

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Asunción Gómez-Pérez
Jérôme Euzenat (Eds.)

The Semantic Web: Research and Applications

Second European Semantic Web Conference, ESWC 2005
Heraklion, Crete, Greece, May 29 – June 1, 2005
Proceedings



Springer

Volume Editors

Asunción Gómez-Pérez

Universidad Politécnica de Madrid

Campus de Montegancedo sn, 28660 Boadilla del Monte, Madrid, Spain

E-mail: asun@fi.upm.es

Jérôme Euzenat

INRIA Rhône-Alpes

655 avenue de l'Europe, 38330 Montbonnot Saint-Martin, France

E-mail: Jerome.Euzenat@inrialpes.fr

Library of Congress Control Number: 2005926291

CR Subject Classification (1998): H.4, H.3, C.2, H.5, I.2, K.4, D.2

ISSN 0302-9743

ISBN-10 3-540-26124-9 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-26124-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2005

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 11431053 06/3142 5 4 3 2 1 0

Preface

This volume contains the papers presented at the 2nd European Semantic Web Conference (ESWC 2005) held in Heraklion, Crete, Greece, from 29th May to 1st June, 2005.

The vision of the Semantic Web is to enhance today's Web via the exploitation of machine-processable metadata. The explicit representation of the semantics of data, accompanied with domain theories (ontologies), will enable a web that provides a qualitatively new level of service. It will weave together an incredibly large network of human knowledge and will complement it with machine processability. Various automated services will help the user to achieve goals by accessing and providing information in a machine-understandable form. This process may ultimately create extremely knowledgeable systems with various specialized reasoning services systems. Many technologies and methodologies are being developed within artificial intelligence, human language technology, machine learning, databases, software engineering and information systems that can contribute to the realization of this vision.

The 2nd Annual European Semantic Web Conference presented the latest results in research and applications of Semantic Web technologies. Following the success of the first edition, ESWC showed a significant increase in participation. With 148 submissions, the number of papers doubled that of the previous edition. Each submission was evaluated by at least three reviewers. The selection process resulted in the acceptance of 48 papers for publication and presentation at the conference (an acceptance rate of 32%). Papers did not come only from Europe but also from other continents.

The selected papers broadly cover the following main topics:

- Semantic Web services
- Semantic Web languages
- ontologies
- reasoning and querying
- search and information retrieval
- user involvement and communities
- natural language for the Semantic Web
- annotation tools
- Semantic Web applications

In addition, ESWC 2005 also featured a special industry forum providing European industry with an opportunity to become more familiar with these technologies. In addition to the main conference, ESWC 2005 included 6 workshops, 5 tutorials, demo and poster sessions. Several European projects had meetings collocated with the main conference.

ESWC 2005 was sponsored by SDK — a group of three European Commission 6th Framework Programme projects known as SEKT, DIP and Knowledge Web.

Together these projects aim to improve world-wide research and standardization in the area of the Semantic Web. The conference also received precious sponsorship from British Telecommunications, DERI Galway and Innsbruck, Ontoprise, AKT and Collexis. Their help was greatly appreciated. We are also grateful to Springer which accepted again this year to publish the proceedings in its Lecture Notes in Computer Science series.

We would like to thank the International Program Committee for its enormous effort in the reviewing process (many reviewers had to evaluate up to 10 papers). In all, 115 additional reviewers were called upon to complete the review process in time.

We owe inexpressible thanks to Miguel Esteban, the Web submission master. Our gratitude also goes to the entire Conference Organization Committee and especially to Christen Ensor for holding things together.

April 2005

Asunción Gómez-Pérez
Jérôme Euzenat

Conference Organization

General Chair	Jérôme Euzenat (INRIA Rhône-Alpes, France)
Program Chair	Asunción Gómez Pérez (UPM, Spain)
Workshops Chair	Wolfgang Nejdl (L3S and Univ. of Hannover, Germany)
Posters Chair	Heiner Stuckenschmidt (Vrije Univ. Amsterdam, The Netherlands)
Demos Chair	Stefan Decker (DERI Galway, Ireland)
Tutorials Chair	Jos de Bruijn (DERI Innsbruck, Austria)
Industry Event	Alain Leger (France Télécom, France)
Publicity Chair	York Sure (Universität Karlsruhe, Germany)
Sponsors Chair	Christen Ensor (DERI Galway, Ireland)
Local Organizer	Christen Ensor (DERI Galway, Ireland)
Local Correspondent	Martin Doerr (FORTH, Greece)
Webmaster	Johannes Breitfuss (DERI Innsbruck, Austria)
Submissions Webmaster	Miguel Esteban Gutiérrez (UPM, Spain)

Program Committee

Dean Allemang (TopQuadrant, Inc., USA)
Jrgen Angele (Ontoprise, Germany)
Sean Bechhofer (University of Manchester, UK)
Richard Benjamins (iSOCO, Spain)
Walter Binder (EPFL, Switzerland)
Paul Buitelaar (DFKI, Germany)
Christoph Bussler (DERI Galway, Ireland)
Fabio Ciravegna (University Sheffield, UK)
Oscar Corcho (iSOCO, Spain)
Hamish Cunningham (University Sheffield, UK)
John Davies (BT, UK)
Ying Ding (DERI Innsbruck, Austria)
John Domingue (The Open University, UK)
Stefan Decker (DERI Galway, Ireland)
Andreas Eberhart (University of Karlsruhe, Germany)
Dieter Fensel (DERI Galway, Ireland and DERI Innsbruck, Austria)
Mariano Fernández-López (CEU, Spain)
Natasha Fridman Noy (Stanford University, USA)
Aldo Gangemi (CNR, Italy)
Mari Georges (ILOG, France)
Fausto Giunchiglia (University of Trento, Italy)

Carole Goble (University of Manchester, UK)
 Christine Golbreich (University of Rennes, France)
 Jeremy J. Carroll (HP, UK)
 Vipul Kashyap (Partners HealthCare System, USA)
 Atanas Kiryakov (Ontotext Lab, Sirma AI, Bulgaria)
 Manolis Koubarakis (Technical University of Crete, Greece)
 Manuel Lama Penin (Universidad de Santiago de Compostela, Spain)
 Alain Léger (France Télécom, France)
 Riichiro Mizoguchi (Osaka University, Japan)
 Dunja Mladenec (J. Stefan Institute, Slovenia)
 Enrico Motta (The Open University, UK)
 Wolfgang Nejdl (LS3 and University of Hannover, Germany)
 Natasha Noy (Stanford University, USA)
 Leo Obrst (MITRE, USA)
 Massimo Paolucci (Carnegie Mellon University, USA)
 Peter Patel-Schneider (Bell Labs, USA)
 Terry Payne (University of Southampton, UK)
 Dimitris Plexousakis (University of Crete, Greece)
 Alan Rector (University of Manchester, UK)
 Marie-Christine Rousset (University of Paris-Sud, France)
 Guus Schreiber (Vrije Universiteit Amsterdam, The Netherlands)
 Daniel Schwabe (PUC-Rio, Brazil)
 Nigel Shadbolt (University of Southampton, UK)
 Steffen Staab (University of Koblenz, Germany)
 Heiner Stuckenschmidt (Vrije Universiteit Amsterdam, The Netherlands)
 Rudi Studer (University of Karlsruhe, Germany)
 York Sure (University of Karlsruhe, Germany)
 Katia Sycara (Carnegie Mellon University, USA)
 Valentina Tamma (University of Liverpool, UK)
 Sergio Tessaris (Free Univ. Bozen, Italy)
 Frank van Harmelen (Vrije Universiteit Amsterdam, The Netherlands)
 Steve Willmott (Universidad Politécnica de Cataluña, Spain)
 Michael Wooldridge (University of Liverpool, UK)

Additional Referees

Sudhir Agarwal	Uldis Bojars	Sam Chapman
Pinar Alper	Kalina Bontcheva	Emilia Cimpian
Anupriya Ankolekar	Stefano Borgo	Paul Clough
Grigoris Antoniou	Amel Boustil	Jesus Contreras
Lora Arroyo	Janez Brank	Monica Crubezy
Alessandro Artale	Christopher Brewster	Olivier Dameron
Jesús Barrasa	Liliana Cabral	Martin Doerr

Alistair Duke	Kyriakos Kritikos	Ken Samuel
Martin Dzbor	Reto Krummenacher	E. Sanches
Marc Ehrig	Vitaveska Lanfranchi	Francois Scharffe
Michael Erdmann	Ken Laskey	James Scicluna
Miguel EstebanGutiérrez	Holger Lausen	Naveen Srinivasan
Cristina Feier	Andrei Lopatenko	Ljiljana Stojanovic
Pablo Fillottrani	Phillip Lord	Nenad Stojanovic
Giorgos Flouris	Francisco	Michael Stollberg
Stefania Galizia	Martin-Recuerda	Umberto Straccia
Raúl García-Castro	Claudio Masolo	Maria del Carmen
Birte Glimm	Peter Mika	Suárez-Figueroa
Francois Goasdoue	Knud Moeller	Martin Szomszor
Juan Miguel Gomez	Matthew Moran	Valentin Tablan
José Manuel	Boris Motik	Merwyn Taylor
Gómez-Pérez	Benjamin Nguyen	Christoph Tempich
Rafael González-Cabero	Barry Norton	Patkos Theodore
Stephan Grimm	Daniel Oberle	Ioan Toma
Peter Haase	Dameron Olivier	Farouk Toumani
Armin Haller	Eyal Oren	Dmitry Tsarkov
Siegfried Handschuh	Jeff Z. Pan	Nikos Tsatsakis
Andreas Harth	Massimo Paoulicci	Daniele Turi
Jens Hartmann	Antonio Pareja Lora	Victoria Uren
Juan Heguiabehere	Nathalie Pernelle	Véronique Ventos
Jan Henke	Livia Predoiu	Laure Vieu
Pascal Hitzler	Irene Polikoff	Johanna Voelker
Laura Hollink	Dnyanesh Rajpathak	Max Völkel
Matthew Horridge	Christophe Rey	Miha Vuk
Uwe Keller	Chantal Reynaud	Holger Wache
Michel Klein	Marc Richardson	Wolf Winkler
Ioanna Koffina	Luis Rodrigo	Jun Zhao
George Kokkinidis	Dumitru Roman	Anna V. Zhdanova
Jacek Kopecky	Sebastian Ryszard Kruk	Kerstin Zimmermann

Conference Organizers



Conference Sponsors

Platinum



Gold



Silver



Table of Contents

Semantic Web Services

Automatic Location of Services

<i>Uwe Keller, Rubén Lara, Holger Lausen, Axel Polleres, Dieter Fensel</i>	1

Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery

<i>Phillip Lord, Pinar Alper, Chris Wroe, Carole Goble</i>	17

Optimally Distributing Interactions Between Composed Semantic Web Services

<i>Ion Constantinescu, Walter Bindrer, Boi Faltings</i>	32

A POP-Based Replanning Agent for Automatic Web Service Composition

<i>Joachim Peer</i>	47

Process-Level Composition of Executable Web Services: “On-the-fly” Versus “Once-for-all” Composition

<i>Marco Pistore, Pierluigi Roberti, Paolo Traverso</i>	62

The OWL-S Editor - A Development Tool for Semantic Web Services

<i>Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, Rukman Senanayake</i>	78

Languages

Temporal RDF

<i>Claudio Gutierrez, Carlos Hurtado, Alejandro Vaisman</i>	93

Multilingual RDF and OWL

<i>Jeremy J. Carroll, Addison Phillips</i>	108

RDFSculpt: Managing RDF Schemas Under Set-Like Semantics

<i>Zoi Kaoudi, Theodore Dalamagas, Timos Sellis</i>	123

REDD: An Algorithm for Redundancy Detection in RDF Models <i>Floriana Esposito, Luigi Iannone, Ignazio Palmisano, Domenico Redavid, Giovanni Semeraro</i>	138
OWL-Eu: Adding Customised Datatypes into OWL <i>Jeff Z. Pan, Ian Horrocks</i>	153
Towards a Fuzzy Description Logic for the Semantic Web (Preliminary Report) <i>Umberto Straccia</i>	167
Consistent Evolution of OWL Ontologies <i>Peter Haase, Ljiljana Stojanovic</i>	182
Ontologies	
Extending HCONE-Merge by Approximating the Intended Meaning of Ontology Concepts Iteratively <i>George A. Vouros, Konstantinos Kotis</i>	198
Soundness of Schema Matching Methods <i>M. Benerecetti, P. Bouquet, S. Zanobini</i>	211
Debugging and Semantic Clarification by Pinpointing <i>Stefan Schlobach</i>	226
An Argumentation Ontology for DIstributed, Loosely-controlled and evolvinG Engineering processes of oNTologies (DILIGENT) <i>Christoph Tempich, H. Sofia Pinto, York Sure, Steffen Staab</i>	241
Towards an Ontology-Based Distributed Architecture for Paid Content <i>Wernher Behrendt, Aldo Gangemi, Wolfgang Maass, Rupert Westenthaler</i>	257
Efficient Semantic Matching <i>Fausto Giunchiglia, Mikalai Yatskevich, Enrico Giunchiglia</i>	272
Ontology-Based Policy Specification and Management <i>Wolfgang Nejdl, Daniel Olmedilla, Marianne Winslett, Charles C. Zhang</i>	290
Web Explanations for Semantic Heterogeneity Discovery <i>Pavel Shvaiko, Fausto Giunchiglia, Paulo Pinheiro da Silva, Deborah L. McGuinness</i>	303

Reasoning and Querying

Approximating Description Logic Classification for Semantic Web Reasoning <i>Perry Groot, Heiner Stuckenschmidt, Holger Wache</i>	318
AIS and Semantic Query <i>Rana Kashif Ali, Steve Cayzer</i>	333
Querying RDF Data from a Graph Database Perspective <i>Renzo Angles, Claudio Gutierrez</i>	346
DRAGO: Distributed Reasoning Architecture for the Semantic Web <i>Luciano Serafini, Andrei Tamilin</i>	361
Dually Structured Concepts in the Semantic Web: Answer Set Programming Approach <i>Patryk Burek, Rafał Graboś</i>	377
Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs <i>Stijn Heymans, Davy Van Nieuwenborgh, Dirk Vermeir</i>	392

Search and Information Retrieval

Product Information Meta-search Framework for Electronic Commerce Through Ontology Mapping <i>Wooju Kim, Dae Woo Choi, Sangun Park</i>	408
Multiple Vehicles for a Semantic Navigation Across Hyper-environments <i>Irene Celino, Emanuele Della Valle</i>	423
Activity Based Metadata for Semantic Desktop Search <i>Paul Alexandru Chirita, Rita Gavriloaie, Stefania Ghita, Wolfgang Nejdl, Raluca Paiu</i>	439
An Ontology-Based Information Retrieval Model <i>David Vallet, Miriam Fernández, Pablo Castells</i>	455
Knowledge Sharing by Information Retrieval in the Semantic Web <i>Neyir Sevilmis, André Stork, Tim Smithers, Jorge Posada, Massimiliano Pianciamore, Rui Castro, Ivan Jimenez, Gorka Marcos, Marco Mauri, Paolo Selvini, Bruno Thelen, Vincenzo Zecchino</i>	471

Users and Communities

Collaborative and Usage-Driven Evolution of Personal Ontologies <i>Peter Haase, Andreas Hotho, Lars Schmidt-Thieme, York Sure</i>	486
Towards Semantically-Interlinked Online Communities <i>John G. Breslin, Andreas Harth, Uldis Bojars, Stefan Decker</i>	500
The Personal Publication Reader: Illustrating Web Data Extraction, Personalization and Reasoning for the Semantic Web <i>Robert Baumgartner, Nicola Henze, Marcus Herzog</i>	515

Natural Language for the Semantic Web

Generating Tailored Textual Summaries from Ontologies <i>Kalina Bontcheva</i>	531
AquaLog: An Ontology-Portable Question Answering System for the Semantic Web <i>Vanessa Lopez, Michele Pasin, Enrico Motta</i>	546
Lexically Evaluating Ontology Triples Generated Automatically from Texts <i>Peter Spyns, Marie-Laure Reinberger</i>	563

AnnotationTools

Pedro Ontology Services: A Framework for Rapid Ontology Markup <i>Kevin Garwood, Phillip Lord, Helen Parkinson, Norman W. Paton, Carole Goble</i>	578
Semantic Annotation of Images and Videos for Multimedia Analysis <i>Stephan Bloehdorn, Kosmas Petridis, Carsten Saathoff, Nikos Simou, Vassilis Tzouvaras, Yannis Avrithis, Siegfried Handschuh, Yiannis Kompatsiaris, Steffen Staab, Michael G. Strintzis</i>	592
RELFIN - Topic Discovery for Ontology Enhancement and Annotation <i>Markus Schaal, Roland M. Müller, Marko Brunzel, Myra Spiliopoulou</i>	608
Semantic Web-Based Document: Editing and Browsing in AktiveDoc <i>Vitaveska Lanfranchi, Fabio Ciravegna, Daniela Petrelli</i>	623

Semantic Web Applications

Semantic-Based Automated Composition of Distributed Learning Objects for Personalized E-Learning <i>Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Azzurra Ragone</i>	633
Orchestration of Semantic Web Services for Large-Scale Document Annotation <i>Barry Norton, Sam Chapman, Fabio Ciravegna</i>	649
Monitoring Research Collaborations Using Semantic Web Technologies <i>Harith Alani, Nicholas Gibbins, Hugh Glaser, Stephen Harris, Nigel Shadbolt</i>	664
Enabling Real World Semantic Web Applications Through a Coordination Middleware <i>Robert Tolksdorf, Lyndon J.B. Nixon, Elena Paslaru Bontas, Duc Minh Nguyen, Franziska Liebsch</i>	679
A Semantic Service Environment: A Case Study in Bioinformatics <i>Stephen Potter, Stuart Aitken</i>	694
Towards B2B Integration in Telecommunications with Semantic Web Services <i>Alistair Duke, Marc Richardson, Sam Watkins, Martin Roberts</i>	710

Invited Papers

SWebB: Semantic Web Browsing <i>Fausto Giunchiglia</i>	725
The Semantic Grid: Past, Present and Future <i>David De Roure</i>	726
Author Index	727

Automatic Location of Services

Uwe Keller¹, Rubén Lara², Holger Lausen¹, Axel Polleres¹, and Dieter Fensel¹

¹ Digital Enterprise Research Institute (DERI) Innsbruck, Austria

`<firstname>.<lastname>@deri.org`

² Tecnología, Información y Finanzas, Madrid, Spain

`rlara@afi.es`

Abstract. The automatic location of services that fulfill a given need is a key step towards dynamic and scalable integration. In this paper we present a model for the automatic location of services that considers the static and dynamic aspects of service descriptions and identifies what notions and techniques are useful for the matching of both. Our model presents three important features: ease of use for the requester, efficient pre-filtering of relevant services, and accurate contracting of services that fulfill a given requester goal. We further elaborate previous work and results on Web service discovery by analyzing what steps and what kinds of descriptions are necessary for efficient and usable automatic service location. Furthermore, we analyze intuitive and formal notions of match that are of interest for locating services that fulfill a given goal. Although having a formal underpinning, the proposed model does not impose any restrictions on how to implement it for specific applications, but proposes some useful formalisms for providing such implementations.

1 Introduction

Current Web service technologies around SOAP [22], WSDL [3] and UDDI [1], only address the syntactical aspects of a Web services and, therefore, only provide protocols and interface descriptions for services in a rigid way that cannot adapt to changing environments without human intervention. The human programmer has to be kept in the loop and scalability as well as economic value of Web services are limited [5]. The vision of semantic Web services is to describe the various aspects of a Web service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web services. Semantic Web technologies are being applied to Web services in order to keep the intervention of the human user to the minimum. The basic idea is to add semantic markup that can be exploited to automate the tasks of discovering services, executing them, composing them and enabling seamless interoperation between them [4], thus enabling intelligent Web services.

The description of Web services in a machine-understandable fashion is expected to have a great impact in areas of e-Commerce and Enterprise Application Integration (EAI), as it can enable dynamic and scalable cooperation between different systems and organizations.

An important step towards dynamic and scalable integration, both within and across enterprise boundaries, is the mechanization of service discovery. Automatically locating available services to perform a given business activity can considerably reduce the

cost of making applications and businesses work together and can enable a flexible integration, where providers are dynamically selected based on semantic descriptions of their capabilities and maybe other non-functional criteria such as trust, security, etc.

Scope of the Paper. In this paper we will address the dynamic location of services that can fulfill a given request. Hereby, we concentrate on the most fundamental aspect of service descriptions for the problem at hand: the service capability, i.e. what *functionality* the service provides. Approaches to automatic service location must precisely analyze what kind of service descriptions can be used for capturing the static and dynamic aspects of a given service, and how such descriptions can be exploited for efficiently and accurately locating a requested service. While a number of proposals are available in our area of interest e.g. [2, 6, 20, 14, 17], none of them has precisely discussed these aspects, but they mainly focused on some specific description languages and frameworks, partly neglecting overall needs. Therefore, we will first define a model that takes into account pragmatic considerations and defines the border line between different steps involved in the process of locating services, namely: goal discovery, goal refinement, service discovery, and service contracting. We will focus on service discovery and service contracting, analyzing the relevant notions of match. Although different notions of match have been studied in the literature (e.g. [14, 23, 17]), some issues involved in the identification of such notions have not been addressed and will be discussed in this paper.

The remainder of this paper is structured as follows: Section 2 discusses static and dynamic aspects of service descriptions and our assumptions on the problem domain providing a general model for the automatic location of services. Service discovery will be discussed in Section 3 where the relevant notions of match are presented. Service contracting will be addressed in Section 4. Related works in the areas of Web service discovery and software component retrieval will be briefly discussed in Section 5. Finally, we conclude the paper and outline future work in Section 6.

2 A Model for the Automatic Location of Services

A workable approach to automatic service location must precisely define its conceptual model and the particular assumptions underlying the proposed solution. For this purpose, we start by providing a common understanding of what a service is and the levels of abstraction in its description based on [18], as well as our assumptions on the elements involved in the location process.

Definition of Service. Recently, it has been pointed out in [18] that the notion of *service* is semantically overloaded. Several communities have different interpretations which makes it difficult to understand and relate single approaches and exchange ideas and results. In order to reach a common understanding of the problem we address here, we need to precisely define the term *service* and, therefore, what kind of entities we aim at locating. In this document, we use the following interpretation for the term *service*, as described in the conceptual architecture for semantic Web services presented in [18]: *Service as provision of value in some domain*. This definition regards a service as a *provision of value* (not necessarily monetary value) in some given domain, independently

of how the supplier and the provider interact. Examples of services in this sense are the provision of information about flight tickets or the booking of a trip with certain characteristics by a tourism service provider.

Usually, a service provider P does not only provide one particular service S , but a set of coherent and logically related services. For instance, a hotel usually does not only provide the possibility to book a particular room at a particular date for a given number of nights, but instead it will offer the general service of booking rooms. Thus, a provider will be interested in advertising *all the services* it is able to provide, i.e. a set $\mathcal{A}_P$ of services. Following the terminology from [18], we call this *collection of services* an *abstract service* offered by a provider. The smallest unit of advertisement is considered to be an abstract service.

In order to deliver a service, a service provider P usually needs certain information from the requester. For instance, a hotel might require the name of the person booking the room, the requested room features, and a valid credit card number as input information in order to book a room. This input data $i_1, \dots, i_n$ will determine what *concrete service* [18] $S \in \mathcal{A}_P$ has to be provided by P .

Description of Requester Needs. Following the approach taken by the Web Service Modeling Ontology (WSMO) [13], a client specifies its needs in terms of what he wants to achieve by a concrete service $S \in \mathcal{A}_P$ of some provider P . Our assumption is that a user will in general care about *what* he wants to get from P , but not about how it is achieved. The conceptual element which formally reflects a desire in WSMO called *goal*. Goals describe what kind of outputs and effects are expected by the client.

A formal Model for Services and Goals. We use a state-based perspective to formalize the concepts involved in the process of automatically locating services. A state $w \in \mathcal{U}$ (where $\mathcal{U}$ is the set of all possible states) determines the properties of the real-world and of the available information at some point in time e.g. the number of rooms currently available in a given hotel. An abstract service $\mathcal{A}$ is considered as a set of state transformations i.e. a relation on the state space $\mathcal{U}$. Each concrete service $S \in \mathcal{A}$ represents a concrete state transformation $S = (w, w')$, with $w, w' \in \mathcal{U}$. In particular, the delivery of a service S determines the outputs and effects which can be observed by the requester; both can be considered as sets of objects (from some universe U), that are attached to w' . Formally, we denote these sets $out_S(w') \subseteq U$ and $eff_S(w') \subseteq U$.

As mentioned above, whether the provision of a service S is possible depends on some information $i_1, \dots, i_n$ provided by the service requester. What information $i_1, \dots, i_n$ is needed is different for each provider P and each abstract service $\mathcal{A}_P$ provided by P . Hence, $\mathcal{A}$ can be considered as a family of relations $\mathcal{A}(i_1, \dots, i_{n_{\mathcal{A}}}) \subseteq \mathcal{U} \times \mathcal{U}$ where each relation of the family is determined by the concrete input information $i_1, \dots, i_{n_{\mathcal{A}}}$ that the service requester provides i.e. the service description must specify *what* can be delivered by the provider *under which circumstances*. Goals can be formally represented as two distinct sets of objects denoting the required set of outputs $out(\mathcal{G}) \subseteq U$ and effects $eff(\mathcal{G}) \subseteq U$.

Eventually, a service requester is interested in finding service providers P that advertised an abstract service $\mathcal{A}(i_1, \dots, i_{n_{\mathcal{A}}})$ such that there is a concrete service $S = (w, w') \in \mathcal{A}(i_1, \dots, i_{n_{\mathcal{A}}})$ that actually resolves the requesters goal $\mathcal{G}$, i.e. the

service S achieves a (final) state $w' \in \mathcal{G}$ in which the sets of requested and provided outputs $out_S(w')$, $out(\mathcal{G})$ as well as the respective effects $eff_S(w')$, $eff(\mathcal{G})$ match. What this precisely means is described in detail in Section 3.

Since each element $S = (w, w')$ of $\mathcal{A}(i_1, \dots, i_{n_A})$ is determined by the respective initial state $w \in dom(\mathcal{A}(i_1, \dots, i_{n_A}))$ ¹ and the input information $i_1, \dots, i_{n_A}$, whether a provider can serve a given concrete service request cannot be determined without knowing $dom(\mathcal{A})$ and $i_1, \dots, i_{n_A}$ ². Unfortunately, we cannot assume that $\mathcal{A}(i_1, \dots, i_{n_A})$ (and therefore $dom(\mathcal{A}(i_1, \dots, i_{n_A}))$) is static over time. In general, the set will dynamically evolve over time: a hotel will not be able to book a room with a single bed on a specific date if all such rooms in the hotel are already booked on this date i.e. this concrete service contained in $\mathcal{A}(i_1, \dots, i_{n_A})$ cannot be provided.

We identify two sources of dynamics for the set $\mathcal{A}$ of concrete services that can be provided by a given service provider: (1) the input information $i_1, \dots, i_{n_A}$ that the requester is able and willing to provide and (2) the respective set of concrete services $\mathcal{A}(i_1, \dots, i_{n_A})$ that can currently be delivered for this input. Such dynamics must be considered for determining matches between $\mathcal{A}$ and a user goal $\mathcal{G}$. Due to these dynamics, some interaction between the parties involved in the matching process – service requester and provider – will be needed in general to determine if a concrete service S fulfilling the requester goal can be provided. This interaction involves communication between the parties and, thus, can be expected to be rather costly. However, it is this communication which enables location results with high precision. We believe that a scalable framework for finding suitable services must address this problem. Our solution to this problem is to split the process into two successive steps, as done in [18]: A first step identifies possible candidate services using less accurate and *static* descriptions of abstract services (so-called *abstract capabilities*), and a second step which applies precise (and possibly dynamic) service descriptions (so-called *contracting capabilities*) and the costly checks (involving communication) of the candidates identified in the first step. We call the first step *service discovery*, whereas the second step is called *service contracting*.

The abstract capability of a service is defined as the set of states that can *potentially* be reached by the provision of such service, independently of the afore-mentioned dynamic factors. It describes only *what* an advertised service can provide but no longer under which circumstances a concrete service S can actually be provided. The contracting capability describes what concrete services can be delivered under what circumstances. It fully describes the family of relations $\mathcal{A}(i_1, \dots, i_{n_A})$. This might involve interaction between both parties for determining if the input available from the requester side can indeed lead to a state w' fulfilling the requester goal.

Assumptions. In order to define a model for the overall location process (including service discovery and contracting), we need to make clear our assumptions on the domain from which we derive the model. Such assumptions are discussed below:

¹ Here $dom(\mathcal{A}(i_1, \dots, i_{n_A}))$ denotes the domain of the state-space relation $\mathcal{A}(i_1, \dots, i_{n_A})$

² More generally, we should consider only input information $i_1, \dots, i_{n_A}$ the requester is able to provide and *willing to disclose* to the provider of an abstract service $\mathcal{A}$. As discussed in [16], this might involve the use of information disclosure policies and a trust negotiation process.

Pre-defined goals. Human service requesters are not expected to have the required background to formalize their goals. Thus, either goals can be expressed in a familiar language (such as natural language for requesters) or appropriate tools should be available which can support requesters to express their precise needs in a simple manner. Hence, we expect that pre-defined, generic, formal and reusable goals will be available to the requester, defining generic objectives requesters may have. They can be refined (or parameterized) by the requester to reflect his concrete needs, as requesters are not expected to write formalized goals from scratch. We assume that there will be a way for requesters to easily locate such pre-defined goals, e.g. by keyword matching.

Abstract capabilities. Abstract capabilities will abstract contracting capabilities in the sense that they abstract from the input information that is provided by the requester, as well as from the dynamics of the available set of concrete services for this input and at a specific point in time. Abstract capabilities are expected to be complete but not always correct [18]: every concrete service S that can be provided will be a model of the description, but there might be concrete services that are models of the description but cannot be provided by P . For example, a tourism service that provides flights within Europe (but not all possible flights) will describe its abstract service as being able to provide any flight within Europe. However, there might be flights that are a model of this description i.e. they are flights within Europe, but that cannot be provided by P for some reason. This incorrectness is a consequence of the abstraction necessary to make descriptions manageable and the matching of candidate services efficient. Therefore, whether a concrete service can indeed be provided will be determined during the contracting phase i.e. during the contracting phase only providers that can actually provide a suitable concrete service S will be matched.

Contracting capabilities. A service provider will describe the concrete services he can provide by describing its *contracting capability*. The contracting capability will also include the description of what conditions have to be fulfilled for a successful service provision, as well as the relation of the required input to the results of the service. The abstract capability might be automatically derived from the contracting capability and both must be consistent with each other.

It is assumed that the requester goal resulting from refining a pre-defined goal will include the information necessary for contracting, such as the input information the requester can or is willing to offer to a provider. We do not impose that this (possibly big) set of information has to be listed for every goal, but it can be made available to the discovery process by other means e.g. an additional service that provides the information that the requester has available and is willing to disclose. A service will not be selected if the requester is not able to provide all the information required by the provider to actually deliver the required concrete service.

Finally, the communication between requesters and providers will be transparent to us in the descriptions of contracting capabilities i.e. we will not describe and deal with service choreographies but only with a logic representation of the communication act.

Conceptual Model for Service Location. Based on our formal model for services and goals, and the assumptions on the domain given above, we provide a conceptual model for the semantic-based location of services that includes the reuse of pre-defined goals,

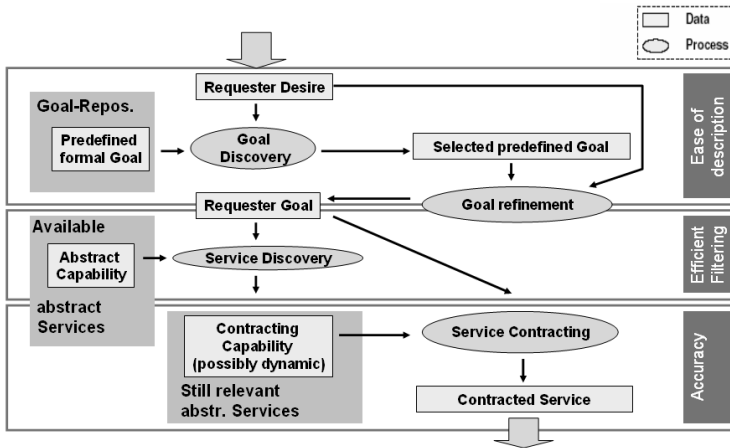


Fig. 1. A conceptual Model for the Discovery Process

the discovery of relevant abstract services, and the contracting of concrete services to fulfill a requester goal. Figure 1 depicts such conceptual model. The different steps of the overall process are:

(1) *Goal Discovery*: Starting from a user desire (expressed using natural language or any other means), goal discovery will locate the pre-defined goal that fits the requester desire from the set of pre-defined goals, resulting on a selected pre-defined goal. Such a pre-defined goal is an abstraction of the requester desire into a generic and reusable goal. (2) *Goal Refinement*: The selected pre-defined goal is refined, based on the given requester desire, in order to actually reflect such desire. This step will result on a formalized requester goal. (3) *Service Discovery*: Available services that can, according to their abstract capabilities, potentially fulfill the requester goal are discovered. As the abstract capability is not guaranteed to be correct, we cannot assure at this level that the service will actually fulfill the requester goal. (4) *Service Contracting*: Based on the contracting capability, the abstract services selected in the previous step will be checked for their ability to deliver a suitable concrete service that fulfills the requester's goal. Such services will eventually be selected.

Let us take as an example a requester who wants to find information about flights from Innsbruck to Madrid on December 21st, 2004. Such requester can express his desire as a text of the form "Search information about flights from Innsbruck to Madrid on December 21st, 2004". This text can be used to perform keyword-based matching of existing pre-defined goals, such as a pre-defined goal for searching flight information.

Once such formal pre-defined goal has been located, it will be refined to reflect the concrete origin and destination given by the requester, as well as the date. This refinement can be done manually (supported by appropriate tools) or automatically from the textual desire.

If a tourism service provider is available and it describes that it can provide flights from any place in Austria to any other place in Europe (as its abstract capability), this service will be considered in the contracting phase. Notice that at this level the dynamics

of the service provision e.g. availability of seats is not considered. Furthermore, we do not expect the service provider to accurately describe all the actual flights it can provide information for, as in general it is not realistic to expect flight information providers to replicate their flight databases in the service description. They will, instead, provide an abstraction of what they can provide.

During the contracting phase it will be checked whether the selected service $\mathcal{A}$ can indeed provides the requested flight information: it will be tested if $\mathcal{A}$ can provide information about flights from Innsbruck to Madrid on the given date. For that purpose, the contracting capability of the service will be used. It might include logic predicates that will actually query the database of the provider to check whether the requested flight information is available. In addition, in case the provider requires extra information from the requester to deliver its service (e.g. personal data of the customer) it will be checked whether the requester can provide such information.

If all the above criteria are fulfilled, the service will be selected and eventually the concrete service provided. In the following sections we concentrate on the notions of match involved in service discovery and service contracting. A more detailed analysis of goal discovery and refinement is beyond the scope of this paper.

3 Service Discovery

As we have sketched in Section 2, the capability of an abstract service can be considered on various levels of abstraction: The most fine-grained perspective on an abstract service $\mathcal{A}$ is to consider it as a family of relations on a state space $\mathcal{U}$. In our discussion, we identified the problem of dynamics in abstract service descriptions when performing accurate and efficient matching and proposed a solution based on a separation of concerns in the overall location process: a service discovery phase based on more abstract and less accurate capability descriptions to identify possible candidates, followed by a service contracting step based on precise capability descriptions which might involve interaction between service requester and provider.

In this section, we discuss the description of abstract services to be used during the first step of the location process, namely *abstract capabilities* of services. An abstract capability of an abstract service is a description which does not depend on dynamic factors, i.e. the current state of the world as well as the requester input needed by the provider. The abstract capability describes only *what* an advertised abstract service $\mathcal{A}$ can potentially deliver but no longer under which circumstances the single services $S \in \mathcal{A}$ can be actually provided.

The proposed modelling of abstract capabilities has been designed in a way such that it provides a formal yet comprehensive model of the description of service capabilities and goals. One particular design goal has been the independence of the formal framework from specific logics. For this reason, we chose a set-based approach for the description of abstract services and goals. How to ground this modelling and discovery approach in logics is shown in [10] (using a slightly different terminology).

Modelling Abstract Services by Means of Abstract Capabilities. A (concrete) *service* S (of an abstract service $\mathcal{A}(i_1, \dots, i_{n_{\mathcal{A}}})$) corresponds to a state transformation

on the state space $\mathcal{U}$: when starting in a specific state $w \in \mathcal{U}$ we end up in a state $w' \in \mathcal{U}$ where the world has changed (some effects are observable) and some output has been provided to the user. Both effects $eff_S(w, i_1, \dots, i_{n_A})$ and outputs $out_S(w, i_1, \dots, i_{n_A})$ can be seen as sets of objects depending on the initial state w and the input information $i_1, \dots, i_{n_A}$ which has been provided to the service provider by the service requester in w . The circumstances under which a service S can be delivered by the provider are represented by w and $i_1, \dots, i_{n_A}$. For example, the description of a concrete service provided by a European airline could be that a business-class flight is booked for the male passenger James Joyce on January 5th, 2005 from Dublin to Innsbruck, and 420 Euros are charged on a MasterCard with number 01233.

If we abstract the description of an *abstract service* $\mathcal{A}$ from the dependency on the contained concrete services, on the provided inputs $i_1, \dots, i_{n_A}$, and on the particular initial states $w \in dom(\mathcal{A}(i_1, \dots, i_{n_A}))$, the description will only specify which objects we can expect from the abstract service as effects $eff_{\mathcal{A}}$ and as outputs $out_{\mathcal{A}}$. For example, an abstract description of a European airline could state that the airline provides information about flights within Europe as well as reservations for these flights, but not what input has to be provided and how this input will determine the results of the service provision. In general, we expect completeness but not necessarily correctness of the abstract capability: every concrete service provided by an abstract service should be covered by the abstract capability, but there might be services which are models of the abstract capability but cannot be delivered as part of the abstract service $\mathcal{A}$ by the provider (since we abstract from the circumstances under which a service can be provided). More formally, we assume $\bigcup_{i_1, \dots, i_{n_A}} \bigcup_{w \in dom(\mathcal{A}(i_1, \dots, i_{n_A}))} eff_S(w, i_1, \dots, i_{n_A}) \subseteq eff_{\mathcal{A}}$ and $\bigcup_{i_1, \dots, i_{n_A}} \bigcup_{w \in dom(\mathcal{A}(i_1, \dots, i_{n_A}))} out_S(w, i_1, \dots, i_{n_A}) \subseteq out_{\mathcal{A}}$. Abstracting further beyond the unions over sets for the single initial states w and input values $i_1, \dots, i_{n_A}$ might in particular be helpful for a provider to simplify the description of abstract capabilities further, since it allows to skip some details on specific constraints of the delivered objects. However, the more abstraction is used beyond these unions (e.g. the airline only specifies to provide tickets for flights all over the world), the less accurate the descriptions of what the service provider is actually able to provide become. *Goals* specify the desire of a client that he wants to have resolved after consuming a service. They describe the information the client wants to receive as output of the service as well as the effects on the state of the world that the client intends to achieve by using the service. This desire can be represented as sets of elements which are relevant to the client as the outputs and the effects of a service provision. According to the WSMO model [13], goals refer to the state which is desired to be reached by service execution.

According to this view, abstract services and goals are both represented as *sets of objects* during the service discovery step. The single descriptions of these sets refer to ontologies that capture general knowledge about the problem domains under consideration. Hence, the objects described in some abstract service description and the objects used in some goal description can or might be interrelated in some way by ontologies. Eventually, such interrelation is needed to establish a match between goals and services.

An important observation in our approach is that the description of a set of objects for representing a goal or an abstract capability can be interpreted in different ways and, thus, the description by means of a set is *not* semantically unique: A modeler

might want to express that either *all* of the elements that are contained in the set are requested (goal) or can be delivered (abstract capability), or that only *some* of these elements are requested (or can be delivered). For this reason, a modeler has to explicitly specify his intention when describing the set of relevant objects for a goal or abstract capability. This intention will strongly affect if we consider two descriptions to match. Therefore, goals as well as abstract capabilities are pairs $\mathcal{D} = (R_{\mathcal{D}}, \mathcal{I}_{\mathcal{D}})$ where $R_{\mathcal{D}}$ is the set of objects which are considered as relevant for the description and $\mathcal{I}_{\mathcal{D}} \in \{\forall, \exists\}$ is the respective (universal or existential) intention. For the sake of simplicity, we will consider in the following only outputs of a service and do not treat effects explicitly. The separation of effects and outputs is conceptual and effects can be dealt with in the very same way. Nonetheless, it is useful to distinguish both since they are conceptually different and we believe that it is beneficial for users to have the ability to apply *different criteria for matching* outputs and effects in a service discovery request. Augmenting the model discussed here accordingly is a straightforward endeavor.

Semantic Matching. In order to consider a goal $\mathcal{G}$ and an abstract service $\mathcal{A}$ to match on a semantic level, the sets $R_{\mathcal{G}}$ and $R_{\mathcal{A}}$ describing these elements have to be interrelated; precisely spoken, we expect that some set-theoretic relationship between $R_{\mathcal{G}}$ and $R_{\mathcal{A}}$ exists. The most basic set-theoretic relationships that might be considered are the following: $R_{\mathcal{G}} = R_{\mathcal{A}}$, $R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$, $R_{\mathcal{A}} \subseteq R_{\mathcal{G}}$, $R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$, $R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$.

These set-theoretic relationships provide the basic means for formalizing our *intuitive understanding of a match* between goals and abstract services. For this reason, they have been considered to some extent already in the literature, for instance in [14] or [17], in the context of Description Logics-based service matchmaking.

On the other hand, we have to keep in mind that in our model these sets only capture *part of the semantics* of goal and service descriptions $\mathcal{D}$, namely the relevant objects for the service requester or service provider. The *intentions* of these sets in the semantic descriptions $\mathcal{D}$ is not considered but clearly affects whether a certain existing set-theoretic relationship between $R_{\mathcal{G}}$ and $R_{\mathcal{A}}$ is considered to actually correspond to (or formalize) our *intuitive understanding* of a match in the real-world. Therefore, we have to consider the intentions of the respective sets as well. Figure 2 gives an overview of the single

Intention of $\mathcal{G} / \mathcal{A}$	$\mathcal{I}_{\mathcal{A}} = \forall$		$\mathcal{I}_{\mathcal{A}} = \exists$	
$\mathcal{I}_{\mathcal{G}} = \forall$	$R_{\mathcal{G}} = R_{\mathcal{A}}$	Match	$R_{\mathcal{G}} = R_{\mathcal{A}}$	PossMatch
	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	Match	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	PossMatch
	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	ParMatch	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	ParMatch
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	ParMatch	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	PossParMatch
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	Nonmatch	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	Nonmatch
$\mathcal{I}_{\mathcal{G}} = \exists$	$R_{\mathcal{G}} = R_{\mathcal{A}}$	Match	$R_{\mathcal{G}} = R_{\mathcal{A}}$	Match
	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	Match	$R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$	PossMatch
	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	Match	$R_{\mathcal{G}} \supseteq R_{\mathcal{A}}$	Match
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	Match	$R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$	PossMatch
	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	Nonmatch	$R_{\mathcal{G}} \cap R_{\mathcal{A}} = \emptyset$	Nonmatch

Fig. 2. Interaction between set-theoretic criteria, intentions and our intuitive understanding of matching

set-theoretical relations as well as their interpretation³ as matches when considering the request and provider intentions. In the table we distinguish several forms of matches: A match (**Match**) means that $\mathcal{A}$ completely satisfies $\mathcal{G}$, a partial match (**ParMatch**) means that $\mathcal{A}$ partially satisfies $\mathcal{G}$ and additional abstract services would be required to completely satisfy the request, a possible match (**PossMatch**) means that there might be an actual match given a more detailed description (at contracting time) of the abstract service, a possible partial match (**PossParMatch**) means that there might be a partial match given more detailed description (at contracting time) of the abstract service or a non-match (**Nonmatch**). Due to space restrictions, we only briefly discuss some entries from the table. A detailed discussion can be found in [10]: (1) $I_{\mathcal{G}} = \forall, I_{\mathcal{A}} = \forall, R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$: The requester wants to get all the objects specified in $R_{\mathcal{G}}$ ($I_{\mathcal{G}} = \forall$), whereas the provider claims that he is able to deliver all the objects specified in $R_{\mathcal{A}}$ ($I_{\mathcal{A}} = \forall$). In this case, the requester needs are fully covered by the $\mathcal{A}$ since all the requested objects $R_{\mathcal{G}}$ can be delivered by the abstract service according to its abstract capability. (2) $I_{\mathcal{G}} = \forall, I_{\mathcal{A}} = \forall, R_{\mathcal{G}} \cap R_{\mathcal{A}} \neq \emptyset$: The requester wants to get all the objects in $R_{\mathcal{G}}$, whereas the provider claims that $\mathcal{A}$ is able to deliver all the objects specified in $R_{\mathcal{A}}$. In this case, the requester needs cannot be fully satisfied by $\mathcal{A}$. At best, the $\mathcal{A}$ can contribute to resolve the desire of the client. Thus, we consider this case as a *partial match*. (3) $I_{\mathcal{G}} = \forall, I_{\mathcal{A}} = \exists, R_{\mathcal{G}} \subseteq R_{\mathcal{A}}$: The requester wants to get all the objects in $R_{\mathcal{G}}$, whereas the provider claims he is only able to deliver some of the objects in $R_{\mathcal{A}}$. In this case, we cannot determine from the given descriptions whether there is a match or not. However, it might turn out when examining a more detailed description there is a match. Such detailed description is available during service contracting (see Section 4). Hence, we consider this as a *possible match*.

Discussion. The proposed modelling approach is based on set theory and ontologies for capturing domain knowledge. By abstracting from dynamic aspects of abstract services, we provide static and general abstract capability descriptions. All the information necessary for checking a match is already available when abstract service descriptions are published, and no interaction with any of the involved parties (requester and provider) is needed for this discovery step. On the other hand, the accuracy we can achieve when is limited. Hence, this discovery step based on such simple descriptions allows an efficient identification of candidate abstract services, but does not guarantee that a matched abstract service will deliver a concrete service fulfilling the requester goal. Abstraction can be used as a means to simplify the description of abstract services by the provider. The overall model is simple, comprehensive and can be implemented in a logical framework [10]. However, the model itself is not based on a specific logical language. The concept of intentions in set-based capability and goal descriptions has not been considered in the literature so far and gives the modeler additional freedom in modelling. Eventually, the use of a set-based model for abstract capabilities can enable the use of Description Logics for classifying and efficiently discovering abstract services to be considered for service contracting. This idea is further elaborated in [12].

³ Please note, that when assigning the intuitive notions we assume that the listed set-theoretic properties between $R_{\mathcal{G}}$ and $R_{\mathcal{A}}$ are the *strongest* ones that actually hold between $R_{\mathcal{G}}$ and $R_{\mathcal{A}}$.

4 Service Contracting

In this section we present service contracting, the last step in the conceptual model presented in Section 3. For service contracting, only the services discovered as discussed in the previous section will be considered. As mentioned above, service contracting will exploit the contracting capability of such services, interpreted as a family of relations $\mathcal{A}(i_1, \dots, i_{n_A}) \subseteq \mathcal{U} \times \mathcal{U}$ where $i_1, \dots, i_{n_A}$ is the input information that has to be made available by the requester.

A contracting capability will describe what input information is required for providing a concrete service and what conditions it must fulfill (i.e. *service preconditions*, denoted by $A_{pre}(i_1 \dots i_{n_A})$), and what conditions the objects delivered fulfill depending on the input given (i.e. *service postconditions*, denoted by $A_{post}(i_1 \dots i_{n_A}, x)$ where x denotes objects that are delivered by a service execution with given input values. We formalize a contracting capability of an abstract service as as follows:

$$\mathcal{A} : \forall x, i_1 \dots i_{n_A}. (as(x, i_1 \dots i_{n_A}) \leftrightarrow A_{pre}(i_1 \dots i_{n_A}) \wedge A_{post}(i_1 \dots i_{n_A}, x)) \quad (1)$$

where $as(x, i_1 \dots i_{n_A})$ represents what objects x the service will deliver for a given input set $i_1 \dots i_{n_A}$. Therefore, the dependency of the service results on the input given is explicitly described. Please note that according to (1) the service is not considered to deliver anything meaningful if the precondition can not be fulfilled by the input provided by the requester. As the dependency on the initial state (e.g. availability of rooms at request time) is dynamic over time, the evaluation of $as(x, i_1 \dots i_{n_A})$ will require interaction with the provider. A goal $\mathcal{G}$ is modelled in terms of a predicate $g(x)$ that describes the relevant objects for the requester.

Single or Multiple Concrete Services. We can enrich the set of matching notions presented in the previous section with an orthogonal dimension: we can express that we can satisfy a particular matching notion wrt. a single concrete service as well as wrt. an arbitrary number of concrete services. This results in additional matching notions that capture additional semantics in a given requester goal. Let us take a requester goal given by the following informal text: "I want to know all flights from Innsbruck to Madrid between 12/10/2004 and 14/10/2004" and an abstract service with the following (informal) capability: "The service provides information about all flights from any place in Austria to any place in Spain on any specific date. Therefore, a single concrete service cannot fulfill the requester goal, but a set of successive concrete services (of the same abstract service) can together fulfill the requester goal: one for each day in the requested period of time. These concrete services correspond to different input information provided by the requester. This can be seen as a simple form of composition, but it can still be captured in our contracting framework and in the definition of the formal proof obligations that have to be checked to determine whether concrete services fulfilling the goal can be contracted.

Extending the Set-Based Modelling. The intuitive notions of match that can be considered at contracting time will be the same as in Figure 2, except for the *possible match* and *possible partial match* notions; as contracting will precisely determine what

concrete services can be provided, we can fully determine whether a service fulfills the requester goal. However, as we introduce the dependency on the input information in the contracting capability, we cannot model our notions of match in terms of set-theoretic relations. Therefore, we replace the set-theoretic relations by the following logical relations, where $\mathcal{O}$ is the set of ontologies that give the terminology used by both descriptions:

1. $R_G = R_A$ for a single concrete service:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \exists i_1, \dots, i_{n_A}. (\forall x. (g(x) \leftrightarrow as(x, i_1 \dots i_{n_A}))) \quad (2)$$

For multiple concrete services:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \forall x. (\exists i_1, \dots, i_{n_A}. (g(x) \leftrightarrow as(x, i_1 \dots i_{n_A}))) \quad (3)$$

2. $R_G \subseteq R_A$ for a single concrete service:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \exists i_1, \dots, i_{n_A}. (\forall x. (g(x) \rightarrow as(x, i_1 \dots i_{n_A}))) \quad (4)$$

For multiple concrete services:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \forall x. (\exists i_1, \dots, i_{n_A}. (g(x) \rightarrow as(x, i_1 \dots i_{n_A}))) \quad (5)$$

3. $R_A \subseteq R_G$ for a single concrete service:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \exists i_1, \dots, i_{n_A}. (\forall x. (g(x) \leftarrow as(x, i_1 \dots i_{n_A}))) \quad (6)$$

For multiple concrete services:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \forall x. (\exists i_1, \dots, i_{n_A}. (g(x) \leftarrow as(x, i_1 \dots i_{n_A}))) \quad (7)$$

4. $R_G \cap R_A \neq \emptyset$ for a single concrete service:

$$\mathcal{A}, \mathcal{G}, \mathcal{O} \models \exists i_1, \dots, i_{n_A}. (\exists x. (g(x) \wedge as(x, i_1 \dots i_{n_A}))) \quad (8)$$

For multiple concrete services the proof obligation for this matching criterion is logically equivalent to the one used for a single concrete service.

If none of the above holds, then the service cannot provide any of the required results, which is similar to the set-theoretic relation $R_G \cap R_A = \emptyset$.

These relations (together with the respective intentions $I_A = \forall$, $I_A = \exists$, $I_G = \forall$, and $I_G = \exists$) can be used for precisely determining if the service fulfills the goal given.

Notice that the input involved in the relations above has to be made available by the requester. This does not impose that the requester has to list for every goal all the information he has available, but he can for example offer a service that provides his available information on demand. In addition, some input information can automatically be extracted from the goal description e.g. if the requester wants to fly from Innsbruck to Madrid we already know that he can provide Innsbruck as the departure location and Madrid as the arrival location. However, how this information is made available to the

contracting process is beyond the scope of the paper, and it is assumed that it will be available in some way during the contracting phase.

In the logical relations above we do not put any restriction on the logical expressivity allowed, as our intention is to formalize the intuition behind service contracting. Similar relations have been formalized using Transaction Logic and implemented using FLORA-2 in [11] and [12].

Contracting based on the above formalizations will in general involve communication with the provider and will be expensive. For example, we can determine whether a given flight can be booked by communicating with the provider, as such availability is dynamic over time and, furthermore, it is not realistic to expect an airline to include its complete flights database as part of the contracting capability description. However, such contracting will be only attempted for abstract services that have been efficiently filtered at discovery time and that are already known as relevant for the goal at hand.

5 Related Work

Software Component Retrieval. The problem of semi-automatically retrieving software components is very similar to the automatic location of services. Specification matching has been proposed in several works e.g. [8, 9, 19, 23] to evaluate how software components relate to a given query i.e. user's need. Specification matching relies on the axiomatization of software components and user queries. A formal (logical) relation is then defined and whether a given query and component satisfy this relation is checked. Such a relation must capture the notion of reusability i.e. if the relation holds for formally specified components and queries, it means that the component can be reused to solve the problem captured by the query.

The work on software component retrieval has not defined a conceptual model for the location of relevant components, but only different notions of match for a given query and a given component have been studied [12]. While such notions of match focus on locating a software component that can be used in the place where the software component represented by the query could, in service discovery we focus on what results can be delivered by the service. Therefore, the notions of match studied for software component retrieval have to be adapted to the Web services domain. Service contracting is not directly considered, as it is outside the application area of software component retrieval. A more detailed account of the work on software component retrieval and its relation to service discovery is given in [12].

Automatic Web Service Discovery. A number of proposals for using Description Logics [15] and OWL-S [4], or similar descriptions for the automatic discovery of services are available [17, 14, 2, 6]. However, none of them provides a conceptual model and they regard discovery as a one step process. In addition, these approaches are not suitable for contracting as they do not employ rules for describing the relation between the results of the service and the input given.

METEOR-S discovery [21] is very similar to the approaches mentioned above, but it uses request templates similar to our pre-defined goals. It also annotates service reg-

istries, specializing them on a given domain and exploiting such annotations during discovery. However, it does not define a conceptual model and it is not suitable for contracting.

LARKS [20] deals with the description of agent capabilities and requests⁴ and the matchmaking of those. The discovery model used in LARKS defines different filters of different complexity and accuracy, allowing the user to select the trade-off between the efficiency and accuracy he needs. However, this model does not address the problem of the different levels of abstraction that are expected in service descriptions, and does not discuss how the requests will be defined by users. Furthermore, it does not consider the contracting of services.

For service discovery, none of the afore-mentioned proposals has discussed the intuitive notions of match a requester or a provider have in mind when requesting or advertising a service i.e. the intentions. The work in [7] discusses variance in service discovery, a complementary aspect to the intentions we have discussed in this paper.

Our previous work on service discovery and contracting [11] already offered a distinction between these two steps. We have built on top of it and examined the conceptual model described in [18] to elaborate a comprehensive conceptual model including both aspects and to discuss the notions of match involved.

6 Conclusions and Future Work

In this paper we presented a model for the automatic location of services that considers the static and dynamic aspects of service descriptions and identifies what notions of match and techniques are useful for the matching of both. Our model presents three important features: ease of use for the requester, efficient pre-filtering of relevant services, and accurate contracting of services that fulfill a given requester goal. We further elaborated previous work and results on Web service discovery by analyzing what steps and what kind of descriptions are necessary for an efficient and usable automatic service location. Furthermore, we analyzed the intuitive and formal notions of match that are of interest for locating services that fulfill a given goal. Although having a formal underpinning, the proposed model does not impose any restrictions on how to implement it for specific applications, but proposes some useful formalisms for providing such implementation. Recently we started with the prototypical implementation of the proposed framework. As soon as the implementation of the prototype will be completed, we will start with the evaluation of our model based on concrete use cases. Further refinement of the model and the respective implementation based on the empirical results obtained from our experiments is part of our future work.

Acknowledgements. We would like to thank all members of the WSMO and WSML Working Groups for fruitful discussions. In particular, Michael Kifer contributed signif-

⁴ Although LARKS is a language for describing agent capabilities, it can equally be applied to Web services.

icantly in discussions of proposed service description and discovery model. This work has been supported by the SFI (Science Funds Ireland) under the DERI-Lion project, the European Commission under the projects DIP, Knowledge Web, SEKT, and SWWS and by FIT-IT under the project RW².

References

1. T. Bellwood, L. Clément, D. Ehnebuske, A. Hatelly, Maryann Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI Version 3.0, July 2002.
2. B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Request rewriting-based Web Service Discovery. In *The Semantic Web - ISWC 2003*, pages 242–257, October 2003.
3. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>, March 2001.
4. The OWL Services Coalition. OWL-S 1.1 Beta Release. July 2004.
5. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.
6. J. Gonzalez-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. In *KI-2001 Workshop on Applications of Description Logics*, September 2001.
7. S. Grimm, B. Motik, and C. Preist. Variance in e-Business Service Discovery. *Semantic Web Services Workshop at ISWC 2004*, November 2004.
8. J.J Jeng and B.H.C. Cheng. Using Automated Reasoning Techniques to Determine Software Reuse. *Intl. Journal of Soft. and Know. Engineering*, 2(4), Dec. 1992.
9. J.J Jeng and B.H.C. Cheng. Specification Matching for Software Reuse: A Foundation. In *SSR'95. ACM SIGSOFT*. ACM Press, 1995.
10. U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Technical report, DERI, November 2004.
11. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Semantic Web Services Workshop at ISWC*, 2004.
12. R. Lara, W. Binder, I. Constantinescu, D. Fensel, U. Keller, J. Pan, M. Pistore, A. Polleres, I. Toma, P. Traverso, and M. Zaremba. Semantics for Web Service Discovery and Composition. Technical report, Knowledge Web, December 2004.
13. H. Lausen, D. Roman, and U. Keller (editors). Web Service Modeling Ontology (WSMO). Working draft, DERI, March 2004. <http://www.wsmo.org/2004/d2/v0.2/>.
14. Lei Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*, Budapest, Hungary, May 2003.
15. D. Nardi, F. Baader, D. Calvanese, D. L. McGuinness, and P. F. Patel-Schneider (eds.). *The Description Logic Handbook*. Cambridge, January 2003.
16. D. Olmedilla, R. Lara, A. Polleres, and H. Lausen. Trust Negotiation for Semantic Web Services. In *SWSWPC Workshop at ICWS 2004*, July 2004.
17. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC*, pages 333–347. Springer Verlag, 2002.
18. Chris Preist. A Conceptual Architecture for Semantic Web Services. In *Proceedings of the International Semantic Web Conference 2004 (ISWC 2004)*, November 2004.
19. E.J. Rollings and J.M. Wing. Specifications as Search Keys for Software Libraries. In *Proceedings of the Eighth International Conference on Logic Programming*, June 1991.

20. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, pages 173–203, 2002.
21. K. Verma, K. Sivashanmugam, A. Sheth, and A. Patil. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 2004.
22. W3C. SOAP Version 1.2 Part 0: Primer, June 2003.
23. A.M. Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6:333–369, 1997.

Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery

Phillip Lord, Pinar Alper, Chris Wroe, and Carole Goble

School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, UK
`p.lord@cs.man.ac.uk`

Abstract. Semantic Web Services offer the possibility of highly flexible web service architectures, where new services can be quickly discovered, orchestrated and composed into workflows. Most existing work has, however, focused on complex service descriptions for automated composition. In this paper, we describe the requirements from the bioinformatics domain which demand technically simpler descriptions, involving the user community at all levels. We describe our data model and light-weight semantic discovery architecture. We explain how this fits in the larger architecture of the *myGrid* project, which overall enables interoperability and composition across, disparate, autonomous, third-party services. Our contention is that such light-weight service discovery provides a good fit for user requirements of bioinformatics and possibly other domains.

1 Introduction

Web Services and Service Orientated Architectures offer the possibility of the composition and orchestration of distributed resources. The *myGrid* project has been seeking to apply these technologies to the bioinformatics domain. To this end, it has contributed to the generation of a rich service layer, has built a workflow engine and workflow development environment [15].

However, there is a difficulty. Even with a good environment, producing complex workflows is difficult, time-consuming and expensive. One of the reasons for this is the sheer number of services which are available for the biologist to use. Commonly, to promote reusability, each bioinformatics service provides a small unit of functionality. A useful task is achieved by combining these services into a workflow. Scientists using *myGrid* currently have access to over 1000 bioinformatics services. In a research domain where many workflows are developed in-house, experimented with, then either thrown away or extensively modified, speed of development is a rate limiting step [16].

There has been a large amount of interest in “Semantic Web Services”. In board outline, this augments standard Service Oriented Architecture with semantic descriptions of the services. These descriptions help agents (whether human or machine) interact with the service during its life cycle including discovery,

composition, execution and monitoring. Considerable work on technologies such as OWL-S [8] and, more recently, WSMO [1], have focused largely on descriptions to enable automated composition of services. The requirement for full automation, and transparency of composition from the user perspective, has resulted in the application of extremely rich service descriptions, using description logics, or F-logic, based formalisms. In keeping with these approaches, the *my*Grid project has built a domain ontology for describing bioinformatics services, described services semantically using that ontology, and used description logic reasoning to support service discovery [19]. However, we found this approach brought considerable complexity to the architecture, without addressing the requirements of our target users (biologists and bioinformaticians). This complexity arose from at least three areas:

- 1) **Service Descriptions.** There are many aspects to a service which can be described at varying levels of detail, depending on the intended audience and use of the description [18]. The description needed to discover a service is different from that needed to configure a service, which may be different from that to invoke a service. Descriptions for unattended agents need to be more formal and explicit than if there is human user involvement. OWL-S recognizes some of this variety by providing a profile for discovery, a process model for orchestration, and a binding for invocation [8]. However, we have found it is not realistic for service providers to provide comprehensive service descriptions as suggested by OWL-S. Sections 2 and 3 examine how the requirements from the bioinformatics discipline allow us to simplify the problem – narrowing the use of description to discovery, and assuming a human is always responsible for final selection.
- 2) **Variety of Services.** The services available for building workflows are heterogeneous. Only a small proportion could be classed as plain web services (i.e. Web Services described using a Web Services Description Language (WSDL) document with no additional conventions). More information on this heterogeneity is given in section 4. We have found that providing detailed formal descriptions of how each type of service varies in its behaviour is unrealistic. Section 5 illustrates how we simplify the problem by describing only an abstraction of a service that captures its functionality (as recognized by the workflow builder) whilst omitting its more technical behaviour. Section 4 explains how other components (particularly the workflow enactor Freefluo) bridge the gap, and loosely bind the abstract description to concrete implementation.
- 3) **Reasoning.** The use of formal semantics within service descriptions allows the use of reasoning. For example, discovery of a service can be supported by description logic (DL) reasoning that matches a requirement for a service against a set of available services. It does so by recasting it as a classification task and testing for those services that are *subsumed by* the requirements. Our experience in implementing such a DL based system is that sophisticated reasoning is computationally expensive, a cost that is not justified by the benefits it provides to users searching for services. The complexity

of deploying reasoning services within the *myGrid* middleware also proved costly. Section 6 describes our discovery component Feta, which allows users to search over simple semantic descriptions of services using simpler reasoning components.

This paper describes our solution for managing (in our experience) the prohibitive complexity of deploying a fully featured semantic web services architecture. By adopting semantic web technologies such as RDF and OWL yet drastically cutting back on the features used, we are able to deploy a manageable middleware system. As both the technology and the tools progress, we leave the door open for adding complexity back into the system, but only as required by users.

2 The Application of Service Orientated Architectures

Bioinformatics involves the application of computational tools to the problems of biology. It is largely reflective of its history, originating in a large number of small molecular biology laboratories. Each lab generally investigates a small area of biology, with only a few labs world-wide working on a particular area. The data and tools generated by these labs has been made available for the community by web publication. More recently, large quantities of data and many tools have been developed and released by relatively few genome centres.

The use of the web as a primary means of data publication has obvious difficulties: HTML enables presentation to humans, not formal data modelling; services are hard to find; interoperability is poor. The primary mechanism for overcoming these problems has been the expert biologist; cutting and pasting between web delivered forms has been the norm [14]. Automated tools for the service composition have largely been built over the top of these web delivered services, often using Perl, and screen-scraping techniques. While this works, it tends to be fragile to changes in the website.

It is against this background that the *myGrid* project has operated. Along with other projects [7], *myGrid* has developed more formal, programmatically accessible middleware, to enable transfer of information and composition of data and tool services into large workflows, addressing the real needs to the lab biologist [15]. This environment comprises of a number of different components including: 1) Soaplab—A toolkit for the presentation of legacy command line applications, which covers the majority of bioinformatics tools, as Web Services. 2) Taverna—A workflow construction environment which enables the composition of Soaplab and other services into, often large and complex, workflows. 3) FreeFluo—A workflow enactment environment which enact Taverna workflows, invoking services, gathering information and returning it to the user.

The *myGrid* environment has simplified the task of accessing the data sets that are available; however, it has been a victim of its own success. At the current time, there are over a 1000 services which can be used by *myGrid*. This leaves the end user with a substantial problem in terms of selecting appropriate services

for use. This is made worse as the user, in this case, is the biologist who may not be highly skilled or knowledgeable about these services [16].

myGrid assumes the contributions of third parties. Initially most of the tools available for use came from within the project, deployed via Soaplab. Currently, the larger part of the services in use come from external, autonomous service providers. This means there is no unified *myGrid* type system determining the structuring of the data passed between services. As a result, while *myGrid* has reduced some of the previous fragility, it has not solve the difficulties of interoperability.

On the face of it, therefore, the problems of *myGrid* seem to be closely aligned to those addressed by the Semantic Web and SW Services. We have a set of services which we wish to compose in ways unanticipated by the providers or those responsible for the middleware.

3 Semantic Web Services in Bioinformatics

While the problems of bioinformatics appear to be closely aligned to those addressed by current SWS efforts, a consideration of the particular nature of the domain suggests to us that this is not the case, due to a set of constraints placed upon us by the domain. In this section, we consider these constraints.

User Transparency: Existing frameworks have focused on the requirement for transparent composition of web services in order to achieve the high level goals given by users (e.g. Make my travel arrangements for the next WWW conference). While this level of automation may be appropriate in B2C applications, it is less desired within bioinformatics, where the user base wish to be involved in service selection. There are two main reasons for this. Firstly, bioinformatics is a scientific endeavour and much depends on the correct selection of services, something that the users may later be forced to justify under peer review. Secondly, they suspect that they will make better decisions than a software agent. In this they are probably correct. Service selection requires significant areas of knowledge, including the technology of bioinformatics, the biological questions being examined and the level of trust the user has in different data sources. Any model of bioinformatics is likely to be wrong, at least in the first instance, particularly if the model is built by the middleware providers rather than those with the biological knowledge.

Messaging Opacity: One of the key difficulties with integration in bioinformatics has been the lack of formal and explicit structuring for its key datatypes. This is true for even relatively simple datatypes, such as DNA sequence which is a simple two-bit code; there are at least 20 different flat file formats for representing this data. The standards that do exist have often come about as a result of many years of collaborative work, so both service providers and consumers have a large investment in these (in)formalisms. Changing these data formats is not an attractive option. The practical upshot of this is that most web services provided for use within bioinformatics do not formally describe their messaging formats with a WSDL document as

the messages are not structured with XML. Within *my*Grid, we have chosen to deal with this by assuming that the information passed around by *my*Grid middleware will be largely opaque to it.

While these two features may appear to be problematic, in practice they simplify the task and scope of the form of semantic service discovery required. As the user is fully involved in the process of service selection, we only require descriptions which reduce the problem space from selection from 1000 services to approximately 10 from which the user can then choose. Other authors have previously noted that this semi-automatic approach simplifies the task of service composition [13], although they see this as a step toward further automation, rather than a strong user requirement. The opacity of the messaging structures means that descriptions do not have to relate to the internal structuring of the data; the best that we can do is describe the existence of a particular datatype¹. However, the absence of formal structuring means that tools such as WSDL2OWLS [12] are of little use; there is little information in the WSDL file which can be mined from it.

4 Coping with Web Service Styles

Of the 1000+ available external bioinformatics services available to *my*Grid users, less than 5% would be considered *plain* web services. Other services consist of approximately:

25% Soaplab services. Soaplab uses web services, but exposes a stateful CORBA-like interface described later in this section.

30% Bio-Moby services. The Bio-Moby project provides a registry and messaging format² for bioinformatics services [17]. This is not described further, but again, imposes additional semantics over normal web service invocation.

30% Web based REST services. The Seqhound [10] sequence retrieval system delivers its services through a Representational State Transfer (REST) style interface, where all the information that is required for the service invocation is encoded in a single HTTP GET or POST request.

10% workflows. *my*Grid allows the incorporation of workflows into larger workflows.

In addition, although not strictly services, local Java applications and Java scripts³ can also be distributed for use within workflows and so need to be discovered and integrated.

¹ Our analogy here is with MIME types. `text/html` tells you little about the datatype, but is useful none the less.

² As described early, messaging formats are opaque in bioinformatics, because most of the data is informally structured. Like *my*Grid, Bio-Moby's messaging format reflects this, consisting of a thin enveloped which simply describes the existence of a given datatype.

³ implemented using the BeanShell (<http://www.beanshell.org>).

Table 1. Two different service interfaces to BLAST, a widely used bioinformatics tool. BLAST operates over a biological sequence, has a number of parameters and returns a single complex BLAST report. The “Document Style” interface has a single method taking a complex set of parameters, while the “Object Style” interface uses object identifiers to provide an *ad hoc* object orientation

Document Style	BlastReport performBlast(Sequence, gap, etc. . .);
Object Style	ObjectIdentifier getInstance(); void setSequence(ObjectIdentifier, Sequence); void setGap(ObjectIdentifier, Gap); ... BlastReport invoke(ObjectIdentifier);

To illustrate further the additional semantics typically found, in Table 1, we give an example of two different presentations of the same service, in this case a BLAST search. This is one of the most widely used tools within bioinformatics. It uses a DNA or protein sequence to search for similar sequences from a database. As such, it takes a sequence as its main input, along with some of a large set of other parameters which modify the search functionality. One standard presentation of this service, which we describe as a “Document Style” approach, has a single operation which takes a sequence as an input parameter and returns a complex BLAST report which is the standard output of this tool. The second provides a much more “Object Style” interface which requires multiple interactions with the service to perform a single BLAST search. This style of interaction is typified by Soaplab.

To describe the object style of service using, for example, OWL-S would require the use of preconditions (**getInstance** must be called before **setSequence**) and effects (**getInstance** uses resources on the server).

Within *myGrid*, however, we have taken an alternative approach. While there are different web service styles, they are only a limited number. Instead of trying to cope with these service styles through the application of semantic descriptions, we have, instead, used an extensible workflow enactment environment, called FreeFluo [11]. The interaction with the web service is handled through a Java interface or *processor*. While this framework is not fully generic, it appears to answer our requirements; support for a new style of service can be added rapidly—in minutes or hours depending on their service complexity. Moreover, the use of widely adopted language such as Java makes this process considerably cheaper than the use of OWL. In effect, we bury the invocation problem by taking advantage of limited families of service patterns with idiomatic patterns of invocation.

The FreeFluo engine then presents a common abstraction over the individual service styles which is used by both the semantic service discovery component, Feta, and the workflow building environment, Taverna. It is this abstraction which we seek to describe as it is this that the users wish to discover.

5 *my*Grid's Data Model of Services

In this section we describe the core data model which we use to describe services. Given the constraints that bioinformatics presents and the support that other parts of the *my*Grid architecture provide for the invocation of services, the key differences between this model and that present within OWL-S are those of omission; we have nothing in this model equivalent to either the grounding or process models and only a subset of the service profile. They are also a few additional features which model the ideas users have about services, but which do not map to the underlying middleware layer.

The majority of the information in the data model was captured in the *my*Grid service ontology described previously [19]. This ontology contains substantial information describing the bioinformatics domain, which acts as an annotation vocabulary including: descriptions of the core bioinformatics data types (e.g. `DNA_sequence`), a characterization of the tasks commonly performed (e.g. `Protein_Analysis`) and a description of the biological entities being investigated (e.g. `homologue`). The core data model is shown as a Conceptual UML class diagram in Figure 1.

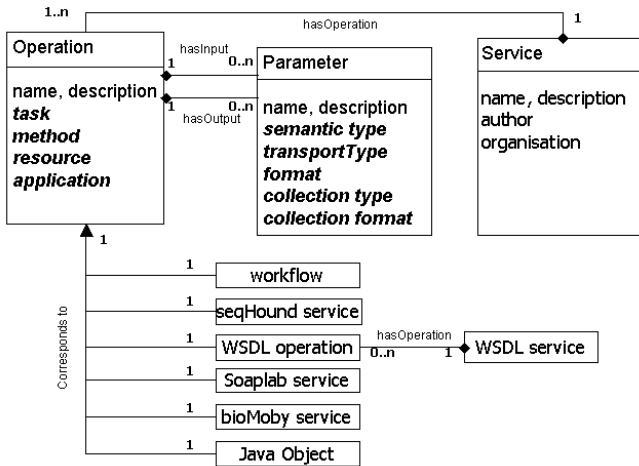


Fig. 1. Feta's Data Model of Services: Those attributes filled with terms from the *my*Grid ontology are marked italicized

Within this data model we distinguish between the core unit of functionality, i.e. the **operation**, and the unit of publication, i.e. the **service**. Our initial analysis of the bioinformatics web services suggested that, in most cases, a Service presented a set of operations providing related but independent functionality. For this reason, the **service** entity encapsulates information only relating to publication; this includes information such as the provider organization

name, the author of the service description, and a free text description of the functionality.

In general, a service may provide one or more service operations. Conventional web services with no state are good examples of this. These operations do not map directly to operations at the WSDL layer. For “object style” services, described in the previous section, the multiple WSDL operations all provide a single unit of functionality from the users perspective. Soaplab services, therefore, are all modelled as a service with a single operation. For other service styles, such as *my*Grid workflows, or Seqhound services, there is no underlying WSDL representation to map to. Again, FreeFluo removes the difficulty of linking between the abstracted service descriptions and these different invocation layers.

The capabilities of **operations**, within Feta, are characterized by the inputs, outputs and several domain specific attributes. All of these attributes use, as fillers, concepts from the *my*Grid domain ontology. These attributes are:

- The overall *task* being performed by the operation. While this attribute has no semantics in the invocation layer, it is an useful description for users who understand the biological *in silico* experiment being performed.
- The underlying *method* being used. Many key bioinformatics tasks can be achieved using more than one algorithm; biologists differ in their level of trust for these different methods, so describing these provides a useful criterion for service selection.
- The *application* to which the service belongs. For example, many service implementations are provided by the EMBOSS project. Again, biologists differ in their trust for these different implementations.
- The *resource* that the service uses. Many tools can operate over different data sets. Services providing access to these tools are likely to provide similar or identical invocation interfaces. This attribute enables the biologist to distinguish between these services.

The inputs and outputs of an operation are modelled through the **Parameter** entity. In addition to its name and textual description, a parameter is described with the following attributes:

- The *semantic type* describes the domain specific data type in question, such as **DNA_sequence**.
- The *format* describes the representation of the data. Many data types can be represented using multiple different formats; some services are agnostic to these formats while some are highly specific.
- The *collectionType* and *collectionFormat* attributes are useful where services return a set of results rather than a single item.
- The *configurationParameter* describes whether the parameter is the “main” input or not. This distinction is common in bioinformatics and can best be

described by analogy to a unix command line⁴: each command has standard input and a set of switches. In general, such parameters have “sensible” default values and can be ignored during service discovery. For this reason, its representation in service descriptions is essential.

6 The Feta System Architecture

In this section, we give an architectural overview of the Feta discovery system, as shown in Figure 2. The key characteristic of this architecture is its relative simplicity: the core components communicate through web services; service descriptions are developed using XML and by applying generic XML tooling; querying is performed with Jena using only RDF(S) entailment rather than DL reasoning. Feta is meant as a light-weight semantic search engine rather than a full service registry, so this functionality is deferred to the standard web services registry, namely UDDI [3]. The core components can be grouped as semantic service publishing components (dark grey in Figure 2), service querying components (light grey) and the *my*Grid service ontology (unshaded) used by both.

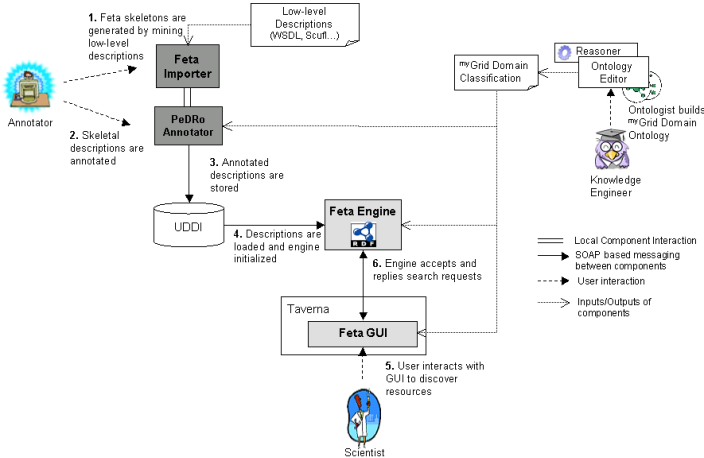


Fig. 2. Architectural Overview of Feta

6.1 Semantic Service Publication

The requirement for Feta to discover multiple styles of service and to reflect the workflow building scientist’s perspective demands manual annotation of the service descriptions. Given that a service’s capability is not reflected by its invocation interface provided at the primary publishing stage, it becomes essential for

⁴ Biologists normally use the term *parameter* specifically to refer to what we call configuration parameters

a secondary publishing stage to take place. It is during publication that the mapping from low-level descriptions of services to the more abstract, user-oriented descriptions takes place.

The absence of formal structuring for most bioinformatics data types (see Section 3), mean that the information which can be obtained from the services themselves is limited to: 1) The Service Name 2) The names and number of service operations. 3) The names and number of operation parameters. For plain web services this information is *imported* from their WSDL files via an XML transformation process. As with the FreeFluo engine, we support an extensibility layer to enable the import of other service styles. In the case of Soaplab services this is achieved by introspective invocation of the service. Other service styles provide information in their own manner and require their own import functionality. In each case, the end product is an XML document conforming to the data model describing in Figure 1, recast as an XML schema. As these documents contain the basic structure for the semantic service descriptions, but little of the information required, we describe them as *skeletons*.

6.2 Service Annotation

Following the generation of skeleton documents, manual annotation of these documents is required to provide full descriptions. This annotation process can take considerable time. In the first instance most descriptions have been developed by expert bioinformaticians from within the *my*Grid project. In our experience, the key difficulty has been poor documentation of the services, requiring experimental invocation of the service with test data. More recent experience with service publishing frameworks such as Soaplab, provide documentation directly associated with services which eases this process considerably.

It is clear that tool support is required for this process to encourage either external service providers, or service consumers to generate their own semantic service descriptions. To this end, we use the PeDRo application [6]. This provides a GUI based interface which allows users to generate XML instance documents conformant to a given XML schema. The tool is also ontology aware and can provide easy access to the vocabulary at the point of use. Annotation is limited to named classes rather than fuller class expressions. In Figure 3, we show this application in the process of annotating a plain web service. The tool is not restrictive in the data that is required for the annotator; it is possible to generate descriptions with minimal information to be augmented at a later date as required.

The use of XML, at this point, provides two key advantages: 1) The use of XML schema validation ensures that Feta documents are internally consistent relieving the need for further error checking at later states of the discovery process. 2) Some service providers will already have access to metadata which can be mapped into the Feta schema and used for service discovery. The use of familiar technology should ease the process should service providers choose to bypass the manual annotation step.

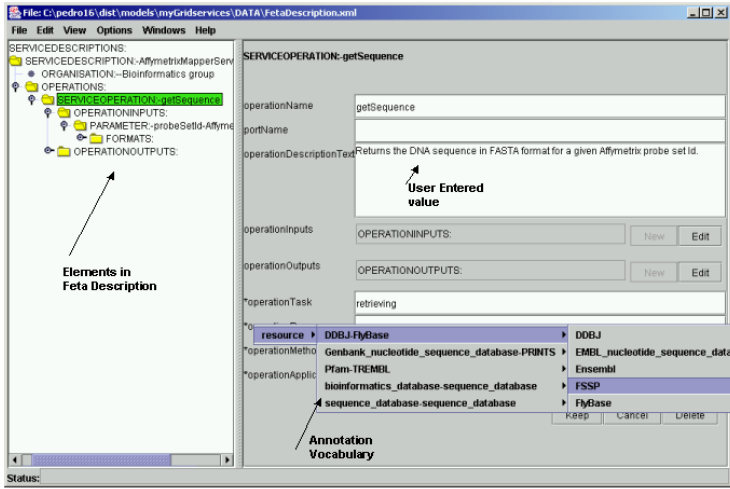


Fig. 3. A screen shot of the XML Data Entry Tool PeDRo

The XML document produced is conformant to the data model, described in Figure 1, containing concepts (represented using URL's) to the ontology described in Section 6.3. Currently, the complexity and time-consuming nature of the annotation phase is one of the key reasons why we do not use complex, OWL-based, service descriptions. Annotation providers are generally not conversant with the use of such technology and are unlikely to make use of the expressive power of OWL. Our experience suggest that even the application of a vocabulary is a demanding process.

6.3 The *myGrid* Domain Ontology

It is clear that the domain ontology is a critical component in ensuring the utility of any semantic service discovery architecture. We have discussed previously the approaches of different projects from within bioinformatics for the development of such an ontology [7]. *myGrid*'s approach has been to develop a seed ontology which will both provide enough utility for initial users of the system. In turn, this should encourage those in the domain to contribute new terms. This process has previously been used highly successfully in bioinformatics [2].

Due to the complexity of the domain, we choose to develop a complex property based ontology using OWL (initially DAML+OIL), which enabled us to take advantage of the reasoning at development time [19]. For use within Feta, we have reasoned over the ontology and then exported it as an RDF(S) hierarchy.

6.4 Querying Feta Descriptions

Following the annotation phase, Feta descriptions are published, making use of a UDDI registry. The Feta Engine engine then imports these descriptions,

along with the RDF(S) version of the domain ontology, from where they can be queried. The decision to avoid the use of OWL and reasoning technologies at query time enables considerable architectural simplicity at this point. The Feta Engine is essentially a set of canned RDQL queries accessible via a web services interface. We currently use Jena [5] as our implementation backend as its query engine provides support for RDF(S) entailment. The canned queries that we currently support include:

- An operation that accepts input of a given semantic type or something more general.
- An operation that produces output of given semantic type or something more specific.
- An operation that performs a given task (or uses method or uses resource or is part of Application) or something more specific.
- An operation that is of type “WSDL based Web Service Operation”, “Soaplab Service”, “Scufl Workflow” etc.
- An operation whose name/description contains a given phrase.

6.5 The Feta GUI Query Tool

The focus of semantic discovery in this paper has been to provide support the workflow building. It is clear, therefore, that the discovery architecture needs to be accessed from within Taverna, our workflow building environment. For this purpose, we provide a plug-in which is shown in Figure 4. The query interface enables the user to build a composite search query using the supported canned queries.

Results of the search are then returned to the user in a results panel shown in Figure 5. Any additional information available about the service is also displayed enabling the user to make the final selection of the most appropriate service. These services can then be added to the workflow by means of drag and drop. Currently returned results are not ranked as most queries narrow the total number of services from which the user can then select manually.

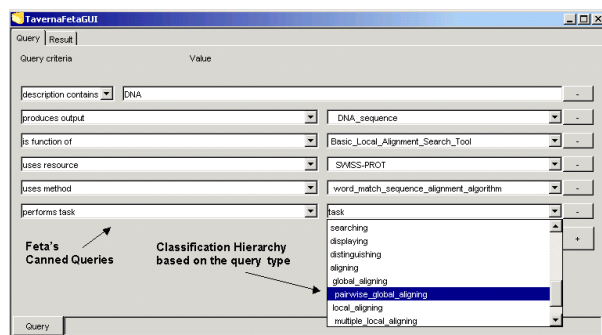


Fig. 4. GUI Panel for Building Search Requests

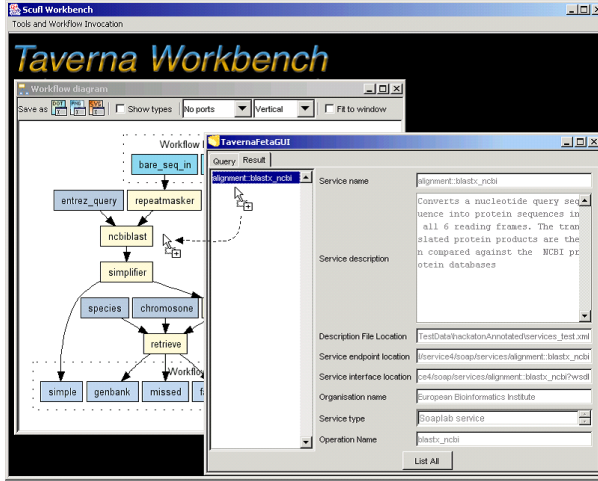


Fig. 5. GUI Panel for Displaying Search Results

We currently consider the query interfaces to be preliminary. Although more expert bioinformaticians are comfortable with this kind of boolean query interface, many biologists are not. We would like to pursue further integration within Taverna to alleviate this need. In particular, we pursue the use of workflow context to filter services. Hiding the explicit use of semantic service discovery should enable it to become a more natural part of the process of workflow building.

7 Discussion

In this paper, we have described our application of Semantic Web technologies to service discovery within bioinformatics. On the whole, the distinctive features of our system are: 1) Its light-weight semantic support 2) Its semi-automatic approach to discovery 3) Its user-oriented capability-based model that enables discovery.

Within Feta we have adopted light-weight semantics, using an RDF(S) classification and entailment, as this appears to be sufficient in this domain. There is an increasing urgent demand for a publicly available registry and associated search facilities. Our choice of an RDF(S) backend has enabled development and deployment of a system with low complexity, without precluding migration to a richer semantic framework based on OWL when required. Currently, we have found that reasoning technologies are useful during construction of a domain ontology [19]. However our assessment is that there is a limited role for reasoning technologies in enabling user oriented service discovery. The level of expressivity that the system can cope with is limited not by the complexity of the reasoning task but by the requirements of the biologist end users who develop both the

service advertisements and discovery queries, and the capabilities of the user interfaces that are used in their generation.

It is possible that satisfiability checking would prove useful during the generation of service descriptions, as the ontology could be used to express constraints on the use of the vocabulary over and above those already provided by the PeDRo user interface. For example, services descriptions with **DNA_Sequence** inputs and outputs, but performing a **Protein_Analysis** task are likely to be erroneous. However, this sort of constraint checking would require a substantial extension to the ontology. It would also require significant ontological engineering knowledge from the curators; this is likely to discourage the community involvement vital to the development of an accurate representation of the domain [2].

Although we have concentrated on providing appropriate user interfaces and tool support for the process of semantic service description and discovery, there are still some areas of weakness. The *myGrid* ontology, in particular, is a key component of the architecture; without an appropriate model of the bioinformatics, we will not be able to provide appropriate service discovery. There is currently no way for service providers and consumers to provide feedback on this model at the point of use. The Bio-Moby project [17] has a more open and collaborative approach to ontology building, but lacks the quality control that *myGrid*'s curatorial approach provides. Better tooling should enable us to take advantage of the best features of both these approaches.

The Semantic Web has always been envisaged to have levels of expressivity—as typified by the Semantic Web Layer Cake [4]. Much of the work on semantic web services has focused on the upper levels of the expressivity. In common with other authors [9], we have found that “being light-weight and flexible trumps other features”. We believe that our XML and RDF(S) based architecture fulfils most of the requirements of the bioinformatics domain while retaining the simplicity, which enables us to adapt service discovery to the specific nature of bioinformatics. We suspect that the use of such light-weight architectures with appropriate data models are likely to be very useful in many other domains.

References

1. S. Arroyo, M. Stollberg, and Y. Ding. WSMO Primer. DERI Working Draft v01, 2004.
2. M. Bada, R. Stevens, C. Goble, Y. Gil, M. Ashburner, J. A. Blake, J. M. Cherry, M. Harris, and S. Lewis. A Short Study on the Success of the Gene Ontology. Accepted for publication in the Journal of Web Semantics, 2004.
3. T. Bellwood. UDDI Version 2.04 API Specification. UDDI Committee Specification, OASIS, July 2002.
4. T. Berners-Lee. Semantic web. XML2000, 2000. <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>.
5. J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. Technical report, HP Labs, December 24th 2003.
6. K. L. Garwood, C. F. Taylor, K. J. Runte, A. Brass, S. G. Oliver, and N. W. Paton. Pedro: a configurable data entry tool for XML. *Bioinformatics*, page bth251, 2004.

7. P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *International Semantic Web Conference*, pages 350–364, 2004.
8. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, Lecture Notes in Computer Science. Springer, July 2004.
9. R. Masuoka, B. Parsia, and Y. Labrou. Task computing - the semantic web meets pervasive computing -. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.
10. K. Michalickova, G. D. Bader, M. Dumontier, H. Lien, D. B. R. Isserlin, and C. W. Hogue. SeqHound: biological sequence and structure database as a platform for bioinformatics research. *BMC Bioinformatics*, 3(32), 2002.
11. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
12. M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura. Towards a Semantic Choreography of Web Services: from WSDL to DAML-S. In *In Proceedings of the International Conference on Web Services (ICWS 2003)*, pages 22–26, 2003.
13. E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*, Angers, France, April 2003.
14. R. Stevens, C. Goble, P. Baker, and A. Brass. A Classification of Tasks in Bioinformatics. *Bioinformatics*, 17(2):180–188, 2001.
15. R. Stevens, H. Tipney, C. Wroe, T. Oinn, M. Senger, P. Lord, C. Goble, A. Brass, and M. Tassabehji. Exploring Williams Beuren Syndrome Using *my*Grid. In *Bioinformatics*, volume 20, pages i303–310, 2004. Intelligent Systems for Molecular Biology (ISMB) 2004.
16. D. Tran, C. Dubay, P. Gorman, and W. Hersh. Applying task analysis to describe and facilitate bioinformatics tasks. *Medinfo*, 2004:818–22, 2004.
17. M. D. Wilkinson, D. Gessler, A. Farmer, and L. Stein. The BioMOBY Project Explores Open-Source, Simple, Extensible Protocols for Enabling Biological Database Interoperability. *Proc Virt Conf Genom and Bioinf*, 3:16–26, 2003.
18. C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating experiments using semantic data on a bioinformatics grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.
19. C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *The International Journal of Cooperative Information Systems*, 12(2):597–624, 2003.

Optimally Distributing Interactions Between Composed Semantic Web Services

Ion Constantinescu, Walter Binder, and Boi Faltings

Ecole Polytechnique Fédérale de Lausanne (EPFL),
Artificial Intelligence Laboratory,
CH-1015 Lausanne, Switzerland
`firstname.lastname@epfl.ch`

Abstract. When information services are organized to provide some composed functionality, their interactions can be formally represented as workflows. Traditionally, workflows are executed by centralized engines that invoke the necessary services and collect results. If services are clustered (e.g., based on geographic criteria), locally routing intermediary results between services in the same cluster can be more efficient.

This paper has several contributions: First, it presents a framework allowing the execution of workflow parts to be mediated by special *execution sites*. Second, we describe a trigger-based mechanism allowing partial results to be routed between execution sites. Finally, we present an approach for optimally computing the distribution of workflow parts to execution sites accordingly to an integrated cost model for workflow execution. The model is created by merging cost-models provided by individual execution sites through a Contract Net task brokering protocol. The models consider cost measures for service activation, parameter transfer, and service execution.¹

Keywords: Service composition, service execution, distributed computing, invocation triggers, workflows.

1 Introduction

A considerable amount of recent research work has focused on the automated composition of agent and web services based on a semantic description of user requirements and service capabilities [7, 3, 4, 9, 2].

Interactions between individual services can be organized according to constraints (e.g., data dependencies) so that they provide some required functionality that none of the individual services could offer alone. The resulting interactions can be represented as a workflow which specifies in which order the individual services have to be invoked and how data has to be passed between these

¹ The work presented in this paper was partly carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European projects KnowledgeWeb (FP6-507482) and DIP (FP6-507483).

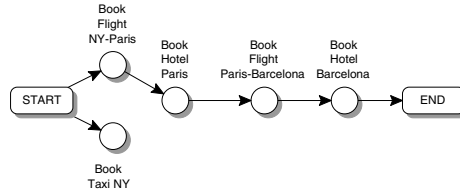


Fig. 1. Travel planning workflow

services. Some workflow specifications may include conditionals [2] or loops [3]. In this paper we consider workflows that do not have conditionals or loops. Also we do not consider transactions and semantic compensation of the effects of service invocations.

Usually, a workflow is executed in a centralized way, either by the client who needs the service or by a server that acts as workflow execution engine. While this approach gives complete control over the workflow execution to a single entity (which may monitor the progress), it may lead to inefficient communication, as all intermediary results are transmitted to the central workflow execution site.

When services are clustered together, transmission of the intermediary results to the client may significantly degrade the performance of the overall workflow execution. Consider the following travel scenario: On behalf of its user living in New York, a Personal Agent (PA) has to book travel arrangements for a trip to Paris and to Barcelona. For this purpose, a workflow as in Fig. 1 has been created (possibly by a semantic web services composition algorithm as in [2]) that books a plane ticket from NY to Paris, arranges a Taxi drive from the home of the user to the airport in NY, books a hotel in Paris, books a plane to Barcelona, and finally reserves a hotel in Barcelona. If we consider that the geographical distribution of the services to be invoked is reflected by the computational cost of the execution of the workflow (e.g., remote interactions between NY and Paris and Barcelona might be slow) the classical solution of centrally executing the workflow might not be the best.

In this paper we propose an alternate solution adapted to this kind of scenarios (clustered services). In our approach, the invocation of services listed in the workflow is not direct but rather mediated by an extra layer of execution sites situated closely to the services to be invoked. These sites are able to provide cost measures regarding the invocation of services and the transfer of partial results between execution sites. For managing the routing of partial results, execution sites use the concept of service invocation triggers, short *triggers* [1]. Each invocation of a service has an associated *trigger* configured on a given execution site. A trigger is able to collect the required input data before it invokes the service, i.e., triggers are also data buffers. Partial results can be sent to one or more triggers on the same execution site or on other execution sites, meaning that triggers also support multicast. According to the cost measures proposed by the execution sites, a workflow is optimally decomposed in several parts. Each

part is assigned to an execution site and triggers are configured for handling the required data-transfers.

Once the trigger for the first services in the workflow have received all input data, the associated services are invoked and the outputs are forwarded to the triggers of subsequent services. Consequently, the workflow is executed in a fully distributed way, the actual service invocations are done with small overhead due to the “closeness” of execution sites, and the data is transmitted directly from the producer sites to all consumer sites.

In the next section we present the formalism that we use for specifying different kinds of workflows and we list some of the assumptions that we use for the rest of the paper. Then in Section 3 we explain our approach to mediated and distributed workflow execution and we introduce the concept of service invocation triggers. In Section 4 we present an overview of the operation of our system, giving the main steps required for computing the optimal distribution of an workflow. Section 5 presents more details regarding how the integrated cost model is computed and in Section 6 we show how the optimal assignment for the workflow distribution is generated from the integrated cost-model. In section 7 we present more details regarding the API for creating and manipulating triggers on execution sites. In section 8 we consider the handling of failures during the execution of a composed service. In section 9 we compare our approach with some related work. Finally, the last section concludes this paper.

2 Formalism and Assumptions

We model our system using a formalism similar to OWL-S (<http://www.swsi.org>) or WSMO (<http://www.wsmo.org>) that describes services and service workflows. We differentiate between services and workflow descriptions that include only functional aspects, *grounded* service and workflow descriptions that specify also pragmatic aspects relevant to the usage of the service, and cost-annotated workflows that specify also cost measures for different operations in the life-cycle of a web service.

Services are described functionally by a set of input (required) and a set of output (provided) parameters. Each input (resp. output) parameter has an associated name that is unique within the set of input (resp. output) parameters. In this paper we do not consider the type of parameters, as we assume that whenever a service receives an input for a particular parameter, the actual type of the passed value corresponds to the formal type of that service parameter. We assume that services are invoked by some sort of remote procedure call (RPC), such as SOAP RPC [8]. The input values for the service parameters are provided in a request message, the output values are returned in a response message. Asynchronous (one-way) calls can be easily mapped to RPC, which is actually the case for SOAP over HTTP.

A workflow is defined by a set of service invocation steps and a set of data dependencies between required parameters and available parameters. The workflow itself can be seen as a composed service with a number of input and output

parameters. We assume that workflows are consistent regarding any semantic and syntactic constraints that appear in the functional service descriptions. This could be straightforward when the workflows are generated by automatic procedures like the type-compatible service composition algorithm described in [2].

A service grounding makes reference to a functional service description and specifies in addition the pragmatic aspects needed for the actual usage of the service, like transport endpoint details.

A grounded workflow makes reference to a non-grounded workflow and to the respective set of service invocation steps and set of data dependencies. In addition, a grounded workflow specifies a set of groundings and a set of bindings which associates a grounding from the set of groundings to each step in the service invocation set.

A cost-annotated workflow extends a normal workflow by introducing a set of groundings and defining execution cost-measures as functions between the groundings and the workflow steps and data-dependencies. A cost-annotated-grounded workflow specifies also the binding of different groundings to different steps, thus allowing for an exact value for the cost of the workflow execution to be computed.

Formally we define a workflow W as a directed acyclic graph $W = \langle S, T \rangle$, where S , the set of nodes, contains steps that refer to functional service profiles and T , the set of edges, contains data dependencies specified as tuples of the form $\langle s_1, s_2 \rangle$ where $s_1, s_2 \in S$. Any workflow graph contains two special nodes s_{start} and s_{end} to model the parameters provided as input to the workflow, respectively required as output results. s_{start} has no data-dependencies and there are no services having data-dependencies on s_{end} .

We defined a grounded workflow GW as a tuple containing the same sets S and T as the workflow W above but extended with a set of service groundings G and a set of bindings B : $GW = \langle S, T, G, B \rangle$. The set G contains groundings for the functional services referred from steps in S . The set of bindings B contains tuples of the form $\langle s, g \rangle$ where $s \in S$ and $g \in G$.

We define a cost-annotated workflow CW as a tuple containing all the sets as the grounded workflow GW excepting the set of bindings B but including three cost functions C_G , C_S and C_T : $CW = \langle S, T, G, C_G, C_S, C_T \rangle$. The three functions correspond to operation costs for each of the three main stages in the life-cycle of a service:

- Service activation $C_G(g) : G \rightarrow \mathcal{R}$. Any service in the frame of a workflow must be activated once, prior to its first invocation. This is independent on how many times a service is used in the workflow. This could correspond to the installation of some infrastructure-required components like local client stubs or third-party applications. The C_G function defines the cost of activating the service specified by the grounding g as a real value. The configuration costs of the triggers presented below can be included in cost returned by the function.
- Parameter transfer $C_T(t, g_1, g_2) : T \times G \times G \rightarrow \mathcal{R}$. In order to be able to execute a service, its parameters must be transferred from the execution site cor-

- responding to the service prior in the workflow graph (possibly s_{start}). The C_T function defines the cost of transferring the parameters specified in the data-dependency t between the execution sites specified in the groundings g_1 and g_2 corresponding to the two services involved in the data-dependency.
- Service execution $C_S(s, g) : S \times G \rightarrow \mathcal{R}$. After the service has been activated and its input parameters have been transferred to the current execution site, the service can be executed. The C_S function defines the execution cost of the workflow step s when the service in the step uses the grounding g . Please note that this cost could include the cost for transferring the parameters between the execution site specified in the grounding g and the actual service.

We define a cost-annotated-grounded workflow as a combination between the grounded and cost-annotated workflows above as the tuple: $CGW = \langle S, T, G, B, C_G, C_S, C_T \rangle$. The cost of this kind of workflow can be uniquely computed as the sum of activation costs for components included in at least one binding plus the sum of the data-transfer costs and execution costs taking into the consideration service groundings as specified by the bindings in B . This can be expressed formally by the formula:

$$\begin{aligned}
 cost(CGW) = & \sum_{g \in G, \exists \langle s, g \rangle \in B} C_G(g) + \\
 & \sum_{\langle s_1, s_2 \rangle \in T, \exists \langle s_1, g_1 \rangle \in B, \exists \langle s_2, g_2 \rangle \in B} C_T(\langle s_1, s_2 \rangle, g_1, g_2) + \\
 & \sum_{s \in S, \exists \langle s, g \rangle \in B} C_S(s, g).
 \end{aligned}$$

3 Service Invocation Triggers

In this section we give an overview of service invocation triggers, a concept that we use to model the management of execution sites. A service invocation trigger, in short trigger, is configured on an execution site and corresponds to one invocation of a service. Basically, the trigger together with the underlying execution site can be considered a specialized proxy for a single service invocation. A trigger plays four different roles:

1. It collects the input values for a service invocation.
2. It acts as a message buffer, as each input value may be transmitted by a distinct sender at a different time.
3. It triggers the service invocation when all required input values are available (synchronization).
4. It defines the routing of service output values. Each output value may be routed to multiple different triggers. That is, a trigger is capable of multi-casting.

With the aid of triggers it is possible to distribute the knowledge concerning the data dependencies of the services within a workflow. The main difference

between the workflow itself and the corresponding triggers is not in the information which is more or less the same, but in the fact that the triggers are distributed on different execution sites. Each trigger defines the service that it will have to invoke. The trigger waits until all required input values are available before it fires (i.e., triggers the service invocation). Moreover, each trigger encapsulates workflow-specific knowledge where the results of the service invocation are needed. As the trigger (enacted by the underlying execution site) behaves as a proxy for the service, it handles the results of the service invocation and forwards them to other triggers or services according to its routing information.

In order to illustrate the advantages of using triggers let's compare how the workflow presented in the introduction would be executed, first in the classic case of a centralized workflow engine and secondly in an environment where triggers and execution sites are used.

In the first case (Fig. 2 (a)) the centralized execution of the workflow would require a number of direct request-reply interactions with the services involved: booking a flight and a taxi in NY, booking a hotel in Paris and a flight to Barcelona, booking a hotel in Barcelona. This will result in 5 request-reply interactions amounting to 10 messages, all possibly carried over slow data-links.

In the second case (Fig. 2 (b)) the results for booking the flight in NY could be directly fed to the hotel booking service in Paris; this service could send its time constraints directly to the service booking the hotel in Barcelona which could locally forward the booking result for booking the Paris-Barcelona flight; in turn the flight booking in Paris could directly send the arrival time to the hotel booking service in Barcelona. Finally all booking results could be returned as an

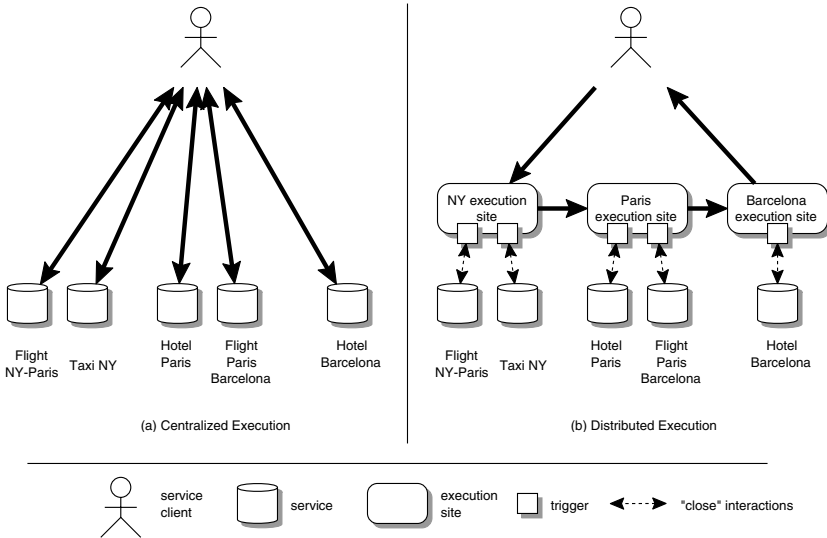


Fig. 2. Centralized and distributed execution of workflows using triggers and execution sites

acknowledgment to the service client. In the right-hand side of the diagram we have figured with a thinner dashed line the interactions between the execution sites and services. We assume that execution sites are very “close” to the services that they have to invoke and thus this invocation costs are much lower than the cost of the long-haul messages exchanged between the service client and the execution sites. In this case only 4 long-haul messages (thick lines) are needed, much less than the 10 required in the previous case.

Using triggers, many different schemes of workflow execution can be implemented. The case that all triggers are to be hosted by the same execution site corresponds to a classic centralized workflow execution model. If each trigger is installed to an execution site “close” to the service that it will invoke, the workflow is executed in a fully distributed way, delivering intermediary results only to those places where they are needed. Our framework does not dictate any of these two extreme settings, allowing for any combination of triggers and execution sites.

Still different distributions of triggers to execution sites might result in workflows that will execute with different performance metrics. In the next section we will present the main contribution of this paper, a generic approach for computing the distribution of triggers over a set of execution sites such that the total workflow execution cost is minimized.

4 Computing Workflow Distributions for Optimal Execution

The system presented here addresses the issue of optimally invoking services in a workflow through a mediation layer, according to some cost measures which reflect the clustering of these services according to some criteria (e.g., geographic distribution). This corresponds to the following problem: Given a workflow, we have to establish how to distribute the triggers corresponding to the invocations of the services in the workflow on a set of available execution sites. The process for achieving that (Fig. 3) comprises the following steps:

1. Send an initial workflow $W = \langle S, T \rangle$ to all relevant execution sites as a Call For Proposals (CFP) (<http://www.fipa.org/specs/fipa00029>). The list of recipients is explicit – each recipient will know all other recipients. Please note that in the case of large numbers of available execution sites some filtering might be used when deciding which execution sites might be “relevant” but we do not address this issue in the current paper.
2. Each execution site returns a cost-annotated workflow CW that presents the execution site’s perspective on how the services in the initial workflow could be grounded. The returned CW will also include the costs of activation, data-transfer and execution, again from the perspective of the current execution site.
3. The groundings and cost functions from cost-annotated workflows received as response to the CFP are merged together.

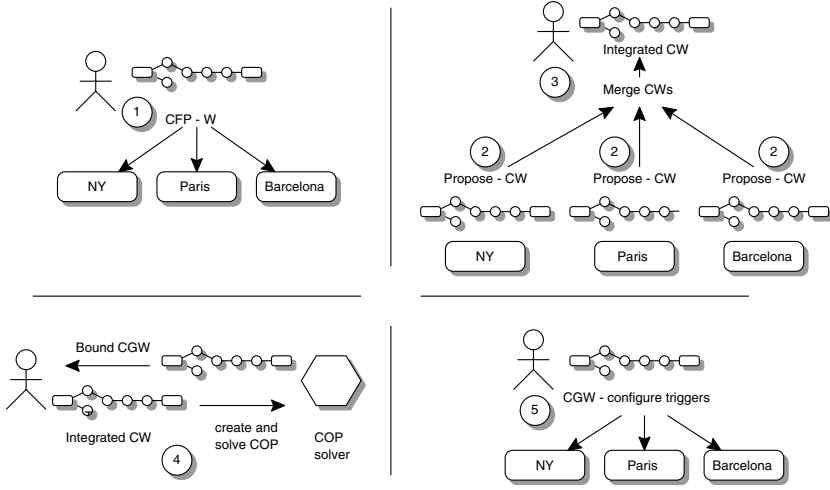


Fig. 3. Computing workflow distributions

4. From the new merged cost-annotated workflow, a Constraint Optimization Problem (COP) is created and solved. The solution of the COP represents a cost-annotated-grounded workflow *CGW* and specifies the optimal binding for the services such that the workflow cost is minimized. From the *CGW* we can determine what triggers have to be created on what execution site and how the results should be routed between execution sites.
5. Finally, the triggers are configured on the concerning execution sites accordingly to the *CGW* obtained as a solution to the COP and the distributed execution of the initial workflow can be started.

5 Generating Cost Measures for Workflow Execution

The first three steps in the process listed above correspond to the generation of a cost model regarding the possible invocation of the services listed in the workflow from any of the possible execution sites.

For this purpose, we use an approach based on a modified version of the FIPA Contract Net Protocol (<http://www.fipa.org/specs/fipa00029>). In the original version agents bid for solving a given task. Our case is slightly different in that the call includes several tasks and the agents/services have to know each other (in order to report peer-wise data-transfer costs). Hence, one shortcoming of our system is that it assumes cooperative services. We consider the case of competitive services as future work.

After receiving the *Call For Proposal (CFP)* with the initial workflow *W*, each execution site creates a cost-annotated workflow *CW* that specifies possible groundings for the services referred in the steps of the original workflow, groundings either locally available or exposed by other execution sites. Also the

service specifies the three cost functions C_G , C_T and C_S for service activation, data-transfer and service execution, costs relative to the current execution site. The service activation and execution function report costs only when the execution site referred in the grounding parameter of the functions is the current execution site. Please note that the execution cost could include the cost of transferring parameter data between an execution site and a “local” service. The data-transfer cost function reports costs only when one of the two execution sites involved in the link is the current site and the other site involved in the link is a remote site. The meaning of the data-transfer cost-function $C_T(t, g_1, g_2)$ is relative to the current execution site: when g_1 is the current execution site and g_2 is a remote execution site, the function result represents the cost of sending the parameter data from the current site remotely; conversely when g_1 is a remote execution site and g_2 is the current execution site the function result represents the cost for the current execution site to receive the parameters data. We consider two different costs for sending and receiving data between execution sites, since communication links are frequently asymmetric both in what concerns the technical parameters and regarding multi-provider business agreements. The data-transfer cost function should reflect some measure of locality (e.g., for an execution site in Paris, the cost for invoking a service in Paris should be lower than the cost for invoking a service in Barcelona).

Finally each execution site returns a cost-annotated workflow CW in the form of a *Propose* message. As the initial CFP might specify also a deadline or timeout, execution sites not providing responses in the given time frame will be discarded from the computations done in the next steps. The process continues if at least one response is received; otherwise a failure is returned.

6 Computing an Optimal Trigger Assignment

For computing the optimal assignment of triggers, we first merge the grounding and cost information in the cost-annotated workflows received as responses to the CFP. Then from the initial set of steps and data-dependencies and the merged sets of groundings and merged cost-functions we create an integrated cost-annotated workflow CW . Finally from the integrated workflow we create and solve a Constraint Optimization Problem (COP), a particular case of Constraint Satisfaction Problem (CSP).

Merging the sets of groundings is straightforward – a new set is created as the union of the grounding sets in the received CG s, where the duplicates are discarded. Merging service activations and execution costs is also straightforward – the new cost functions will just aggregate the costs of activation and execution functions in the received CG s. For data-transfer functions $C_T(t, g_1, g_2)$ the merged cost function returns the sum of the costs of transferring data between execution sites as reported by the two sites specified by the groundings of the link binding. I.e., the total cost of data-transfer will be computed as the sum of the cost reported by the site in g_1 for sending the parameter data to g_2 , plus the cost reported by the site in g_2 for receiving the parameters from the site in g_1 .

Formally, we define a constraint optimization problem (*COP*) as the tuple $\langle X, D, C, R \rangle$ where:

- $X = \{x_1, x_2, \dots, x_n\}$ is a set of n variables.
- $D = \{d_1, d_2, \dots, d_n\}$ is the set of domains of the n variables in X , each given as a finite set of possible values.
- $C = \{c_1, \dots, c_m\}$ is a set of m constraints, where a constraint c_i is given as the list $(x_{i1}, \dots, x_{ik})$ of variables it involves.
- $R = \{r_1, r_2, \dots, r_m\}$ is a set of relations, one for each of the m constraints, where a relation r_i is a function $d_{i1} \times \dots \times d_{ik} \rightarrow \mathcal{R}^+$ giving the cost of choosing each combination of values. Combinations that are not allowed have a very high cost (∞).

A *solution* is a combination of values $v_1 \in d_1, \dots, v_n \in d_n$ such that the sum of the cost of the relations is minimal and different from ∞ . Otherwise we consider that the *COP* has no solution.

We create a *COP* from the annotated workflows received as responses of the CFP as follows:

- X contains a variable for each step of the initial workflow (each execution of a service), for each data-dependency of the initial workflow (each parameter that needs to be transferred between services), and for each grounding in the new set of merged service groundings (each service that might need to be activated).
- The domains of the variables corresponding to executions of services contain as values the possible groundings for the respective service from the merged service grounding set. The domains of variables corresponding to data-dependencies between services contain as values tuples corresponding to all combinations of groundings for the two steps in the data-dependency, corresponding to all reported combinations of senders and receivers. For k execution sites, we have maximum $k(k-1)$ possible values. For example, in the case of three execution sites ES_{NY} , ES_{Paris} , $ES_{Barcelona}$, we have six possible values: $ES_{NY} - ES_{Paris}$, $ES_{NY} - ES_{Barcelona}$, $ES_{Paris} - ES_{NY}$, etc., corresponding to a data transfer between execution sites in NY and Paris, NY and Barcelona, Paris and NY, etc. The domains of service-activation variables corresponding to groundings in the set of merged groundings have two boolean values representing the fact that the service has to be activated or not in the current *COP* solution.
- In C and R we have two kinds of constraints and costs: activation and data transfer. These constraints link execution variables to activation and data-transfer variables, making sure that once a grounding is chosen for executing a service the grounding is going to be activated (the corresponding grounding activation variable is going to be true) and the required input parameters will be available (the values of the variables corresponding to the data-transfer links ending at the current step will have to have the target grounding as the grounding chosen for the execution step). For example, a variable corresponding to a node in the workflow is assigned the value ES_{Paris} . Then

the generated constraints ensure that the respective grounding has been activated and that all the data-dependencies (edges entering the respective node) are fulfilled by being assigned values of the form $* - ES_{Paris}$ (incoming data transfers).

Finally we solve the formulated COP using a state of the art commercial Java solver for constraint optimization problems. Still since there might not be enough responses received from the execution sites by the deadline of the CFP the COP might not have a solution in which case a failure is returned.

7 Defining Triggers

In this section we present more details regarding the installation of triggers. In our description we use the following abbreviations for identifying services, respectively triggers:

- [**SID:**] **Service ID.** Globally unique identifier of a service to be executed. It consists of host, port, protocol, and local service identifier (e.g., service name and version number, depending on the protocol). SIDs are computed from concrete service descriptions (including grounding information).
- [**PID:**] **Parameter ID.** Locally unique identifier for a service input or output parameter.
- [**TID:**] **Trigger ID.** Globally unique trigger identifier. It consists of details regarding the underlying execution site like host and port. It also contains a local trigger identifier (e.g., an integer number referring to an invocation trigger).

Below we present a simple API to deal with triggers in an abstract way:

[**CreateTrigger:**] Creates and installs a trigger.

Arguments:

- Destination of the trigger (host and port). **CreateTrigger** will ask the destination execution site to set up the desired trigger.
- *SID*. The service to be invoked by the trigger.
- Service input parameters to wait for. Each parameter is identified by its *PID*. A parameter may be required or optional. The trigger will fire as soon as all required parameters are available. As for a given parameter multiple values may arrive before the trigger fires (while still some of the required parameters are missing), the client has to define which values to preserve: **preserveLast** or **preserveFirst**. If values for optional input parameters arrive before the trigger fires, they will be passed to the service. After the trigger has fired, arriving inputs are discarded.
- Optional: Input data. For each input parameter a default value may be provided. This value could be transmitted with **SendData** (see below), but including it in **CreateTrigger** may be more efficient and help to reduce network traffic.

- Output routing. For each output value (identified by a PID_O) generated by the service SID , the output routing defines a possibly empty list of pairs (TID_i, PID_i) to forward the output. That is, whenever the service SID returns an output value for the parameter PID_O , the trigger will forward it to all TID_i with the name PID_i , implementing a multicast. If there is a communication problem with a trigger TID_i , the trigger will retry to forward the data several times in order to overcome temporary network problems.
- Desired timeouts:
 1. Timeout to wait for inputs, starting with the installation of the trigger. If not all required input data arrives before this timeout, the trigger will be discarded.
 2. Timeout to wait for service completion, starting with the service invocation.
 3. Timeout to wait for completed forwarding of service outputs, starting when the trigger receives the results of the service invocation.
- Optional: Destination for failure notification message (host, port, protocol). In the case of a failure (i.e., service returning a failure message or expiration of one of the timeouts mentioned before), a failure notification is sent before the trigger is discarded, including information concerning the current state of the trigger. The level of detail of this notification can be configured. The message may simply indicate the reason of the failure, or it may include service inputs resp. outputs the trigger has received so far. This information may help the client to recover from the failure.

Results:

- TID of the installed trigger (if the trigger was accepted).
- Granted timeouts. Each granted timeout may be the desired timeout or shorter.

[RemoveTrigger:] Explicitly removes a trigger. Normally, a trigger is removed automatically if either a timeout occurs or if the output routing task is completed, i.e., all outputs have been forwarded according to the trigger's routing information.

Arguments:

- TID . The trigger to remove.

[SendData:] Sends input data to a trigger. Normally, triggers receive results either by initialization (see the input data of **CreateTrigger**) or through other triggers (forwarded results from another service). However, a client may want to install a trigger and provide input data later on.

Arguments:

- TID . The trigger to send data to.
- Input data. For each input parameter a value may be provided.

[Status:] Returns information concerning the status of a trigger. I.e., whether the trigger is still waiting for required input, which input arguments have been received so far, whether it has already triggered the service, whether it is waiting for the service outputs, etc.

Arguments:

- *TID*. The trigger to ask for its status.

Results:

- Status information.

Three different protocols are involved in the communication with triggers and services:

- The trigger communicates with the service using remote procedure calls (e.g., SOAP RPC [8]). That is, the trigger is transparent to the service, it behaves as any other client.
- The communication between triggers is unidirectional. A trigger forwards results to another trigger. The messages sent from trigger T_A to trigger T_B contains at least one value for an input parameter T_B is waiting for. Even though the communication protocol between triggers need not necessarily comply with standards, SOAP messages are well suited for trigger communication. If the triggered service returns a fault message, the trigger does not forward the message on the normal output routing path, but it may generate a failure notification message (if specified in **CreateTrigger**). Subsequent triggers will notice the failure by a timeout.
- A dedicated, simple protocol supports the API primitives described before. For instance, **CreateTrigger** will try to deploy a trigger on the specified destination platform.

8 Failure Handling

In our approach composed services are executed in a completely distributed way. Therefore, it is not easily possible to monitor the progress of each service invocation. As the client will only receive the final results of the composed service, in general it will notice a failure only after a timeout. In this case, the client may restart the execution of the workflow.

If the used services are not reliable, this approach may result in bad overall performance, since intermediary values may have to be computed multiple times. Hence, the client should make use of the failure notification mechanism in order to collect partial results that had been computed before the failure has happened. Based on the failure notification mechanism, the client could exploit redundant execution plans in order to replace a failed service. As an alternative (but inefficient) solution, if the distributed execution of a composed service fails, the client could simply re-execute the workflow in a centralized fashion (fallback solution).

The client may also use the **Status** primitive of triggers in order to monitor the progress of the execution. Note that **Status** will fail if the trigger has already been removed (i.e., after a timeout or after completing its task). However, **Status** creates additional network traffic, therefore an excessive use of this primitive is not consistent with the principal idea of our approach to minimize the network traffic involving the client.

9 Related Work

The Internet indirection infrastructure i3 (<http://i3.cs.berkeley.edu/>) uses triggers to decouple sender and receiver [6]. In contrast to our triggers, i3 triggers work on the level of individual packets and do not support waiting conditions (synchronization) to aggregate multiple inputs from various locations before forwarding the data. i3 supports only a very limited form of service composition, where individual packets can be directed through a sequence of services. While our triggers are rather transient (used only for a single service invocation) and their placement is explicitly controlled by the client, i3 triggers are more persistent (they act as a longer-term contact point for a service) and are mapped to the Chord peer-to-peer infrastructure, which allows only a limited form of optimizing the routing (by selecting a trigger identifier that will map close to a desired location). Summing up, even though there are some ideas in common, i3 has different goals (indirection, supporting mobility, multicast, anycast) and works at a much lower level than our approach. Our focus is on the efficient routing of intermediary results during the execution of composed services.

Another relevant approach is the SELF-SERV system[5]. The system architecture identifies three kinds of service: elementary, composite, and community. The execution of composite services is managed by coordinators. The concepts are similar to our definition of compositions as workflows and to our infrastructure of execution sites and triggers. The infrastructure presented in [5] is rather complex but no clear details are presented regarding the instrumentation for managing the coordination services or the way of choosing the appropriate coordination service. In this paper we present a simple and clear approach for instrumenting the management of execution mediators. Still the main contribution of this paper is the generic approach for choosing services and execution sites. The distributed execution of the resulting workflow is optimal regarding the costs of service activation, parameter transfer, and service execution.

10 Conclusion

When composed services represented as workflows are executed in a centralized way, partial results might not be efficiently handled. In this paper we describe a multilayered architecture where service invocation is mediated by execution sites equipped with triggers – the equivalent of a service proxy. This approach is advantageous regarding both the number of messages that have to be sent for

executing the workflow and the possibility of having faster interactions between services that are “closely” located.

The main contribution of this paper is an approach for computing the distribution of triggers over a set of execution sites such that for a given cost-model taking into account service activation, parameter transfer, and service execution, the execution cost of the given workflow is minimized.

References

1. W. Binder, I. Constantinescu, and B. Faltings. Efficiently distributing interactions between composed information agents. In *Second European Workshop on Multi-Agent Systems (EUMAS-2004)*, Barcelona, Spain, December 2004.
2. I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *IEEE International Conference on Web Services (ICWS-2004)*, San Diego, CA, USA, July 2004.
3. S. A. McIlraith and T. C. Son. Adapting Golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.
4. S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *11th World Wide Web Conference (Web Engineering Track)*, 2002.
5. Q. Z. Sheng, B. Benatallah, M. Dumas, and E. O.-Y. Mak. Self-serv: A platform for rapid composition of web services in a peer-to-peer environment. In *VLDB*, pages 1051–1054, 2002.
6. I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, Apr. 2004.
7. S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, July 2002.
8. W3C. Simple object access protocol (SOAP), <http://www.w3.org/tr/soap/>.
9. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.

A POP-Based Replanning Agent for Automatic Web Service Composition

Joachim Peer

MCM Institute, University of St. Gallen, Switzerland
joachim.peer@unisg.ch

Abstract. This paper illustrates how a modified version of a modern Partial Order Planner (POP) can be combined with a replanning algorithm to solve planning problems in Web service domains. The contributions of the work are (i) a method of using feedback gained from plan execution for improving plan search and (ii) a novel approach of dealing with nondeterministic Web service operations.

1 Introduction

Web services are distributed software components that can be exposed and invoked over the internet. Commonly, interface description languages such as the Web Service Description Language WSDL [1] are used to describe the syntactical interface of a Web service and the message streams it uses to communicate with its clients. In addition to that, semantic annotations can be created in order to provide software agents with information to reason about the capabilities and consequences of service operations.

In an environment of semantically annotated services, a user who needs to achieve certain goals can be assisted by software agents which automatically identify, invoke, compose and monitor services in order to accomplish the user's goals, which may be either explicitly stated or derived from the user's current situation.

Recently, several papers have investigated the potentials and boundaries of applying *AI planning techniques* to derive Web service processes that achieve the desired goals. It has been pointed out that automatic Web service composition (WSC) differs from classical planning domains in several aspects [2, 3].

First, there is the problem of *incomplete information*: Many planning problems in the Web service domain require the querying of information-providing services, for instance to check the availability of some product at an online retailer. Therefore, the classical planning assumption of a complete initial state description can not be maintained in the WSC domain. Further, there is the problem of *nondeterministic behavior* of services: Web service operations may fail during execution time or yield unexpected results, for instance when a retailer suddenly runs out of stock of some product.

To be practically useful, agents have to be able to handle these contingencies. In this paper, we describe how a modified Partial Order Planning (POP)

algorithm embedded in an execution monitoring agent can deal with the issues mentioned above.

The remainder of this paper is structured as follows: In Section 2, we will briefly describe the conceptual representation schema for Web services our work is based on and in Section 3, we describe the semantics of service executions. In Section 4, we present the structure of the planning problems our system can handle. Then, in Section 5, we lay out our proposed solution and present our contributions to the problem solving algorithms involved. In Section 6, we present examples to illustrate the system and we present our preliminary empirical tests. In Section 7, related work is discussed, followed by a summary of the contributions of the paper in Section 8.

2 Representing Web Service Semantics

Currently, there exist several proposals for markup formats for semantic Web services. Among them are OWL-S [4], WSMO/WSML [5] and the recent SESMA [6] markup format. For the present work we chose SESMA, because it is easy to use and provides convenient support to mark up nondeterministic service operations.

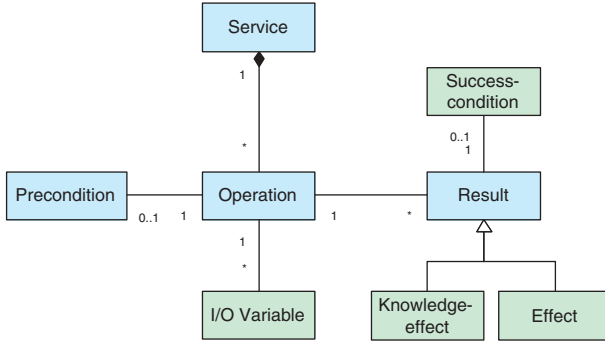


Fig. 1. Service representation in the SESMA model [6]

The syntax and semantics of the SESMA constructs are described in detail in [6]; here we give a brief overview: In the SESMA model, each service consists of a set of operations. Each operation can be written as a quadruple $op = \langle URI, Prec, Res, Var \rangle$. $URI(op)$ denotes the unique identifier of the operation. $Prec(op)$ denotes the optional precondition formula of the operation, which describes conditions that need to be satisfied when the operation is invoked. $Res(op)$ denotes the set of possible results of the operation, which can be *effects* and *knowledge-effects*; both describe conditions that will hold after service execution, but while the world state as a whole may be changed to achieve an *effect*, the world state remains unchanged to achieve *knowledge-effects*; only

```

<op-def name="buyItem" wsdl:portType="ShopA">
  <input>
    <var name="?item" wsdl:part="ean"/>
    <var name="?cc" wsdl:part="ccNr" />
    <var name="?ccexp" wsdl:part="ccExpDate" />
  </input>
  <precondition>
    <and>
      <s:have-creditcard owner="$client"
        nr="?cc" expires="?ccexp"/>
      <s:in-catalog vendor="shopA.com" item="?item"/>
    </and>
  </precondition>
  <output>
    <var name="?result" wsdl:part="buyItemReturn"/>
  </output>
  <effect>
    <!-- to be checked after invocation -->
    <success-condition lang="java">
      !("no".equals(output.get("?result")))
    </success-condition>
    <!-- the desired effect -->
    <s:possess owner="$client" item="?item"/>
  </effect>
</op-def>

<op-def name="getItemList" wsdl:portType="ShopA">
  <output>
    <var name="?item"
      wsdl:part="getItemListReturn"
      wsdl:path="Item/ean"/>
    <var name="?title"
      wsdl:part="getItemListReturn"
      wsdl:path="Item/title" />
    <var name="?desc"
      wsdl:part="getItemListReturn"
      wsdl:path="Item/description"/>
  </output>
  <knowledge-effect>
    <forall>
      <var name="?item" />
      <and>
        <s:in-catalog vendor="shopA.com"
          item="?item" />
        <s:has-title item="?item" title="?title" />
        <s:has-description item="?item"
          description="?desc" />
      </and>
    </forall>
  </knowledge-effect>
</op-def>

```

Fig. 2. SESMA markup for two operations

the *world view* of the agent may be affected by knowledge-effects. Since services often behave nondeterministically, each result $r \in Res(op)$ may have a success condition $SC(r)$, which can be used after service invocation to determine whether or not the invocation did achieve the desired effects.

Further, the set $Var(op)$ defines the variables used in the formulas, whereby input variables are distinguished from output variables. The variable definitions tell the agent how the values represented by the variables are syntactically encoded in the input and output streams sent to/received from the service.

To illustrate the constructs sketched above, we depict the SESMA descriptions of two operations of a Web service *ShopA.com* in Fig. 2:

The operation on the left, *buyItem*, has a **<precondition>** that requires that the client has a credit-card *?cc* and that the item *?item* to be purchased is in the catalog of the shop. The operation has a single **<effect>**, which is that the client will possess the specified product, if the effect's **<success-condition>**, which may be written as a Java(TM) snippet, evaluates to *true* after service execution. The second operation, *getItemList*, has no precondition, but it has a single **<knowledge-effect>** formula. The knowledge effect specifies that, for all *?item*-objects received from the service, there is an *?item* in the catalog with a title *?title* and a description *?desc*.

The **<input>** and **<output>** sections of the operations specify the WSDL message parts and XML paths that identify the location of the variable values in the input and output messages.

3 Semantics of Service Executions

3.1 Representing the World and Its Dynamics

We assume that the world is represented in terms of a state description. Formally, we define a state s as a conjunction of positive atomic formulas (atoms) $L_1 \wedge$

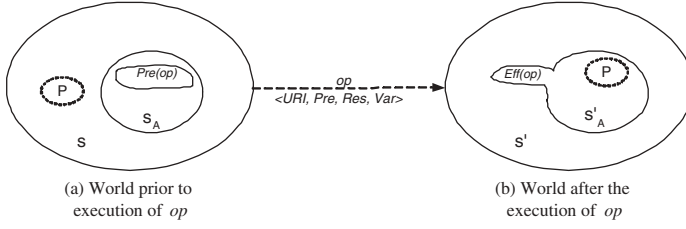


Fig. 3. The world and its dynamics are represented by states and state transitions

$\dots \wedge L_n$. Consequently, for each $L_i \in s$ holds $s \models L_i$ and for each $L_i \notin s$ holds $s \not\models L_i$.

As illustrated by Fig. 3, we distinguish between the global state s of a world and the state knowledge s_A of an agent A who reasons about this world. We assume that $s_A \subseteq s$. This means that an agent may have incomplete knowledge, but we do not assume that it has wrong knowledge.

The dynamics of the world are formalized as a state transition system, which can be thought of as a labeled directed graph: The vertices of the graph are the states, and the edges of the graph are the transitions that may take place, labeled by the operations which cause the state transition. Fig. 3 illustrates a transition from state s to a state s' , triggered by an operation op . As shown in the illustration, the precondition $Pre(op)$ is satisfied by s (and s_A), which allows the agent to invoke the operation. The operation has an effect $Eff(op)$ which is added to s'_A . The operation also has a knowledge-effect; the consequence of the knowledge effect is that the fact P , which the agent was unaware of in its state s_A is added to s'_A , i.e. the agent becomes aware of P after the service execution.

3.2 Conversation Data Sets

Conversation data sets capture the data tokens that are exchanged between clients and services at runtime. Conversation data sets are an important prerequisite for evaluating SESMA formulas.

A conversation data set is a set of substitutions $\{\theta_1, \dots, \theta_n\}$. We denote a substitution as a finite set of the form $\theta = \{x_1/t_1, \dots, x_n/t_n\}$, where each x_i is a distinct variable and each t_i is a constant, such that $x_i \neq t_i$, for all $1 \leq i \leq n$. Note that the variables of an operation are defined in the set $Var(op)$ (cf. Sect. 2). A substituted formula $F\theta$ is a variant of formula F where all variables x in F are replaced by a constant c if there is an element $\{x/c\} \in \theta$.

Let us consider the role of conversation data sets and substitutions in service execution:

- Substituting *input variables*: When invoking an operation op , the agent specifies the values certain input variables should have. This specification is represented by a conversation data set I and each substitution in I contains at most one term t_i for every variable x_i of the input variables defined for the operation.

- Substituting *output variables*: Analogously, the values returned by the service are captured by a conversation data set O , which contains an arbitrary number of substitutions ϑ , where each substitution contains at most one term t_o for every variable x_o of the output variables defined for the operation.

Let us illustrate this using the example markup of Fig. 2: For a particular invocation of the operation `getPrice`, the user might define the input data set $I = \{\{?item/44300\}\}$, and the service might return an output data set $O = \{\{?price/79.00\}\}$. The operation `getItemList` may return an output data set $O = \{\{?item/44300, ?title/CAM300, ?desc/WebCam\}, \{?item/44340, ?title/HS340, ?desc/HeadSet\}\}$, which contains *multiple* substitutions, representing catalog data of an online retailer service.

Next, we define how conversation data sets can be *combined*. We need combined conversation data sets to materialize effects (cf. Sect. 3.5). We start by defining the combination of substitutions: Given two substitutions $\theta = \{x_1/t_1, \dots, x_k/t_k\}$ and $\vartheta = \{x_m/t_m, \dots, x_n/t_n\}$, we define the joint substitution $\theta + \vartheta = \{x_1/t_1, \dots, x_k/t_k, x_m/t_m, \dots, x_n/t_n\}$. Given an input data set I and an output data set O , we define the *combined conversation data set* $C = I \otimes O$ as follows: For each $\theta \in I$ and each $\vartheta \in O$, C contains an element $\{\theta + \vartheta\}$.

3.3 Evaluating Preconditions

The precondition is evaluated against the state s , whereby the values in the conversation data set I provide the substitutions for the variables in the precondition. A precondition P is satisfied iff $s \models P\theta$ holds, for every substitution θ in the input data set I .

Since the agent is only aware of the subset s_A of s it must test the precondition against s_A . As long as the agent can calculate the truth value of the precondition using the literals in s_A , it can conclude that the precondition is satisfied by s as well. However, if the agent has to resort to closed world assumption, i.e. if it assumes $s_A \not\models L^+$ for some positive literal $L^+ \notin s_A$, then there is the possibility that the evaluation against s_A differs from the evaluation against s .

3.4 Evaluating Success Conditions

As mentioned in Section 2, SESMA supports the markup of nondeterministic service results. For instance, a payment Web service may send a note either confirming the transaction or reporting a failure. When dealing with such nondeterministic operations, the agent should only assert the effect of the operation after it has executed it *and* after it received a confirmation of success. For instance, the markup of the operation `buyItem` listed in Fig. 2 defines a success condition which depends on the value of the output variable `?result`.

SESMA success-conditions, which can be written either as declarative SESMA formulas or as boolean Java(TM) expressions, contain the information needed to determine the success or failure of an operation's result(s). We formalize the evaluation as a function $eval(SC(r), s_A, I, O) \mapsto \{true, false\}$. The inputs to

this function are the success condition SC of the result r whose success is to be determined, the known pre-execution state s_A , the input data set I and the output data set O . The output of the evaluation function is a Boolean value, reflecting whether or not the result r did indeed occur and if the effect formula can be materialized. If a result has no success condition, then *eval* will return *true* as a default. Note that it is permitted that one effect of an operation occurs while another effect of the same operation does not.

3.5 Materializing Effects

If an operation is executed without errors, then the successor state can be calculated. All results of the operation, whose success conditions evaluate to *true* or which lack a success condition, are considered; those results whose success conditions evaluate to *false* are ignored.

A result formula E is *materialized* in a successor state s' iff $s' \models E\vartheta$ holds for every substitution ϑ in the combined data set $I \otimes O$.

To formally describe the truth value of an atom A in a given state we introduce the function *val*, which defines for each atom and for each state whether or not the atom is true in that state. Given a state transition from state s to s' by invocation of an operation op , the function is defined as follows:

$$val(s', A) = \begin{cases} true, & \text{if } s \models Pre(op)\theta \text{ for every } \theta \in I \\ & \text{and if there is a result } r \in Res(op) \\ & \text{where } eval(SC(r), s_A, I, O) = true \\ & \text{and } \exists \vartheta \in (I \otimes O) \text{ such that } r\vartheta \models A \\ false, & \text{if } s \models Pre(op)\theta \text{ for every } \theta \in I \\ & \text{and if there is a result } r \in Res(op) \\ & \text{where } eval(SC(r), s_A, I, O) = true \\ & \text{and } \exists \vartheta \in (I \otimes O) \text{ such that } r\vartheta \models \neg A \\ val(s, A), & \text{elsewhere.} \end{cases}$$

The first case states that A is true in the new state s' if (i) the precondition formula (whose variables are substituted by the input values) is satisfied by the pre-execution state and if (ii) A is a consequence of a result formula r (whose variables are substituted by the input and output values) and if (iii) the function *eval* returns *true* for the result. Analogously, the second case describes that A is not true in the new state, if $\neg P$ is a consequence of a result of the operation. The third case describes the situation where A is not affected by the execution of the operation, which addresses the frame problem of logical action descriptions.

From the agent's perspective, a *knowledge effect* is treated (and materialized) just like an *effect*. Nevertheless, there is a difference, which is worth some consideration: A knowledge effect KE of an operation describes an effect without side effects, i.e. it does not contribute to any changes occurring between s and s' , but it may alter the world view s_A of the agent: the statement $[(s' \models KE\vartheta) \Rightarrow (s'_A \models KE\vartheta)] \wedge [(s' \not\models KE\vartheta) \Rightarrow (s'_A \not\models KE\vartheta)]$ holds, for every $\vartheta \in (I \otimes O)$.

4 Planning Goals

Having set out the world the semantic Web service agents act in, we characterize the structure and nature of the goals they are supposed to achieve: We view goal descriptions as logical statements about the desired world state. We allow the specification of goals in the form of literals (positive or negative atomic formulas) and conjunctions and disjunctions of literals, whereby variables are treated with existential quantification. For instance, the following SESMA construct is a valid goal; it is a conjunction of non-ground literals and can be fulfilled by purchasing an *?item* that has a product id (EAN number) “1234”:

```
<and>
  <possess who="client" what="?item" />
  <has-ean item="?item" ean-nr="1234"/>
</and>
```

Further, we support the distinction between goal formulas that describe *achievement* goals and goal formulas that describe *information gathering* goals. Achievement and information gathering goals may be combined to form more complex goal descriptions.

To distinguish information gathering goals from achievement goals, we introduce the goal annotation *find-out*. A plan satisfying a goal formula G that is annotated by *find-out* may not contain an operation that has an *effect* G' , if G and G' unify. Instead, plans that achieve the goal using *knowledge-effects* are sought in this case. This concept is derived from the idea of *maintenance goals* discussed in [7].

As an example, a goal `<find-out> <n:color item='?itm' value='?c' /> </find-out>` may only use operations that do *not* actively affect the color of item *?itm*; we would not want the agent to call an operation that sets the color to some new value and then reports that newly assigned color. Instead, the value should be gathered through a *knowledge effect* of an operation.

Goal expressions that are *not* annotated are interpreted as achievement goals, with the usual semantics, i.e. the agent can do whatever is needed to make the annotated formula true in s' . For instance, the goal `<n:possess item='someID' />` means that the agent can change the world state as necessary to achieve the goal.

5 The Proposed Replanning Agent

In the following we will describe the architecture and algorithms we propose to solve problems in the world formalized as discussed in the last sections.

To deal with the nondeterministic nature of the domain and the arising contingencies, we chose an execution monitoring architecture, where a classical planner – an extension of VHPOP [8] (Versatile Heuristic Partial Order Planner) – is *embedded* in an execution monitoring engine. In Sect. 5.1 we briefly sketch VHPOP and the extension we added to it to fit into the replanning architecture; in Sect. 5.2 we then describe the replanning algorithm itself.

5.1 The POP-Based Kernel

VHPOP uses Partial Order Planning (POP), a well known planning technique where the reasoner searches a space of *plans*. A partially ordered plan is represented as a quadruple $\langle \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L} \rangle$, which consists of the following components: $\mathcal{S}$ is a set of plan steps, i.e. instances of operations. $\mathcal{O}$ is a set of ordering constraints. Each ordering constraint is of the form $s_i \prec s_j$, which means that the step s_i must be executed *before* step s_j . If the set $\mathcal{S}$ of some plan π has at least two steps s_a and s_b where $\mathcal{O}$ neither contains $s_a \prec s_b$ nor $s_b \prec s_a$, then π is a *partially ordered* plan. $\mathcal{B}$ is a set of variable binding constraints on the parameters of action instances: Each variable constraint is of the form $var = x$ or $var \neq x$, where var is a variable of some plan step and x is either a constant value or a reference to a variable of some other plan step. If only ground plan steps are used, then $\mathcal{B} = \emptyset$. Finally, $\mathcal{L}$ is the set of causal links. Causal links are used to keep track why a step was introduced to a plan and to prevent other steps from interfering with that purpose. If a step s_i achieves a proposition p to satisfy a precondition of step s_j , the causal link $s_i \xrightarrow{p} s_j$ is added to $\mathcal{L}$.

Further, the following two derived sets are considered in partial order planning: (1) $\mathcal{OC}$ is the set of open conditions of a plan. An open condition $\xrightarrow{p} s$ emerges when p is a literal that is part of the precondition $Prec(s)$ and when there is no causal link $s_x \xrightarrow{p} s$ in $\mathcal{L}$. In other words, open conditions are preconditions of plan steps which have not yet been addressed by the current plan. (2) $\mathcal{UL}$ is the set of unsafe links. A causal link $s_i \xrightarrow{p} s_j$ is called *unsafe* if there exists a step $s_k \in \mathcal{S}$ such that (i) $\neg p \in Eff(s_k)$ and (ii) $\mathcal{O}$ is consistent with $\{s_i \prec s_k \prec s_j\}$. In such a case, s_k is said to *threaten* the causal link $s_i \xrightarrow{p} s_j$. The union of a plan's open conditions and unsafe links is called the set $\mathcal{F}$ of *flaws* of π , i.e. $\mathcal{F}(\pi) = \mathcal{OC}(\pi) \cup \mathcal{UL}(\pi)$. A plan π that has no flaws is called *complete*.

An open condition $\xrightarrow{p} s$ can be resolved by introducing or reusing a plan step that has an effect achieving p . On the other hand, a *threat* of a causal link $s_i \xrightarrow{p} s_j$ by a step s_k can be possibly resolved either by *demotion*, i.e. by adding an ordering constraint $s_k \prec s_i$ to $\mathcal{O}$ or by *promotion*, i.e. by adding $s_j \prec s_k$ to $\mathcal{O}$. If the planner uses lifted actions, i.e. if it allows action instances with variables in their parameter lists, a threat can also possibly be resolved by *separation*, that is by adding binding constraints such that p and $\neg p$ can not be unified.

The way a planner navigates through plan space, i.e. the strategy it employs to chose the plans to refine and the flaws to remove, determines the efficiency of the planner. VHPOP utilizes a planning graph and several search heuristics to steer the planning process into a fruitful direction; a discussion of these techniques can be found in [8].

We have extended VHPOP by introducing a set $\mathcal{AL}$ (which stands for “Avoid-Links”), which is provided as an additional argument to the main procedure MAKE-PLAN and is subsequently handed over to REFINEMENTS and its various sub-routines, which are described in [8].

$\mathcal{AL}$ is a set of *causal link patterns* that must be avoided by the planner. This means that none of the partial plans $\pi \in \mathcal{P}$ devised by the planner may contain a causal link that matches a causal link pattern in $\mathcal{AL}$.

A causal link pattern is a triple $\langle x, p, y \rangle$ where p is a literal, x is either an operation op which has p in one of its results $Res(op)$ or x is a wildcard (asterisk, $*$), and where y is either an operation op whose precondition is (partially) fulfilled by x (i.e. $p \in Prec(op)$), or a wildcard ($*$). A causal link $s_i \xrightarrow{p} s_j$ matches a causal link pattern $x \xrightarrow{q} y$ iff

1. p unifies with q , and
2. $x = opr(s_i)$ or $x = *$, and
3. $y = opr(s_j)$ or $y = *$,

whereby $opr(s)$ denotes the Web service operation represented by the plan step s . The reasons for maintaining the set $\mathcal{AL}$ of causal link restrictions are (i) to avoid plan structures that already failed in prior executions and (ii) to avoid plan structures that violate the restrictions imposed by *find-out* goals. We will discuss both issues in greater detail below.

5.2 The Replanning Algorithm

In the following we illustrate how to solve planning problems as described in Section 4 by breaking them down into a series of simpler planning problems to be tackled by the modified VHPOP algorithm described in the last section.

The main routine of the replanning agent is SOLVE-PROBLEM (cf. Fig. 4); its arguments are the goal $\mathcal{G}$, the initial situation $\mathcal{I}$, a set *anno* of SESMA-annotations of the available Web services, and the number *threshold* of planning- and-execution cycles to be performed before giving up.

First, the Web service domain and the WSC problem given have to be translated into a PDDL based planning domain description $\mathcal{D}$ to be passed to the VHPOP algorithm (which uses PDDL as input format). This is done by calling the operation BUILD-DOMAIN (cf. line 4). The transformation is straightforward: (i) For each annotated Web service operation a PDDL planning operator is created, whereby the SESMA precondition is transformed into a precondition of the operator. The effect of each planning operator is formed by a conjunction of all results of the corresponding operation, whereby the success conditions are ignored. This encoding is *optimistic*, i.e. it assumes that all results will be achieved. As an example, consider the translation of the SESMA markups from Fig. 2:

<pre>(:action op_buyItem :parameters (?item ?cc ?ccexp ?result) :precondition (and (have-creditcard_ ?cc ?ccexp) (in-catalog_ shop2 ?item)) :effect (possess_ client ?item))</pre>	<pre>(:action op_getItemList :parameters (?title ?desc) :effect (forall(?item) (and (in-catalog_ shop2 ?item) (has-title_ ?item ?title) (has-description_ ?item ?desc))))</pre>

(ii) The WSC planning goal is translated into the precondition of a reserved operator whose effect is the planning goal; this allows us to use variables in goal

```

/* returns a boolean signaling success or failure. */
01 function SOLVE-PROBLEM( $\mathcal{G}$ ,  $\mathcal{I}$ , threshold, anno) {
02   solved  $\Leftarrow \perp$ 
03   attempts  $\Leftarrow 0$ 
04    $\mathcal{D} \Leftarrow \text{BUILD-DOMAIN}(\text{anno})$ 
05    $\mathcal{AL} \Leftarrow \text{EXTRACT-FROM}(\mathcal{G})$ 
06
07   label replan:
08   while(  $\neg \text{solved} \wedge \text{attempts} < \text{threshold}$  ) {
09     Plan  $\pi \Leftarrow \text{MAKE-PLAN}(\mathcal{D}, \mathcal{I}, \mathcal{G}, \mathcal{AL})$  /* invoke VHPOP */
10     if( $\pi = \text{null}$ ) return  $\perp$ 
11     for each step  $s \in \mathcal{S}(\pi)$  ordered by  $\mathcal{O}(\pi)$  {
12       if( $\neg \text{CAUSAL-LINK-VIOLATIONS}(s, \pi, \mathcal{AL})$ ) {
13         EXECUTE( $\text{opr}(s)$ ) /* invoke service operation */
14       } else {
15         attempts++
16         continue replan
17       }
18     }
19   }
20   return (attempts < threshold)
21 }

```

Fig. 4. Main cycle of the replanning algorithm

descriptions. (iii) The fact base s_A of the planning agent is transformed into a set of ground atoms, the description of the initial state.

The next preparation step is to initialize the set $\mathcal{AL}$ with causal link patterns in order to avoid violations of the *find-out* constraints of the goal. This is done by calling the operation EXTRACT-FROM in line 5: For each literal l_g in the goal description $\mathcal{G}$ (which is represented in disjunctive clausal normal form) that has a *find-out*-annotation, all operations in the domain whose effects contain a literal l_{op} that unifies with l_g , a causal link pattern $\langle \text{op}, l_g, * \rangle$ is created and added to $\mathcal{AL}$. This way, the planner will avoid using steps that violate the *find-out*-constraints of the goal.

As described in Fig. 4, the algorithm then enters the main loop (line 8) and first calls the VHPOP routine MAKE-PLAN (line 9). If successful, VHPOP delivers a plan π which (i) is complete, (ii) may be partially ordered, and (iii) does not contain any causal links that match a causal link pattern in $\mathcal{AL}$. If the POP routine fails to come up with such a plan, then the process is terminated and reports failure (line 10).

The complete plan π devised by VHPOP has the *potential* to solve the given problem, but due to the nondeterministic nature of the domain, success is not *guaranteed*. Therefore, the agent is forced to expose the plan to the environment to learn whether or not a potential solution indeed achieves the desired goal. Therefore, the agent chooses a linearization L of $\mathcal{S}(\pi)$ which it aims to execute in a controlled manner (lines 11-18): For each plan step s , the func-

```

/* returns a boolean signaling causal link violations */
01 function CAUSAL-LINK-VIOLATIONS(Step  $s$ , Plan  $\pi$ , Set  $\mathcal{AL}$ ) {
02   boolean foundViolation  $\Leftarrow \perp$ 
03   for each causalLink  $cl = \langle s_i, p, s_j \rangle \in \mathcal{L}(\pi)$  {
04     if ( $cl$  is relevant)  $\wedge \neg(s_i$  achieves  $p)$  {
05        $\mathcal{AL} \Leftarrow \mathcal{AL} \cup cl$ 
06       foundViolation  $\Leftarrow \top$ 
07     }
08   }
09   return foundViolation
10 }

```

Fig. 5. Causal Link Violation Check

tion $\text{CAUSAL-LINK-VIOLATIONS}(s, \pi, \mathcal{AL})$ is called to test whether those preconditions of the remaining steps that should be already achieved are indeed fulfilled (cf. Fig. 5). It does so by testing whether the *relevant* causal links of the plan are fulfilled. A causal link $s_i \xrightarrow{p} s_j$ is *relevant* upon execution of step s if s_i was already performed and s_j is not performed yet, i.e. when $s_i \prec s \prec s_j$ in L .

If all relevant causal links for s (and the steps beyond s) are fulfilled, then the agent will *execute* the operation represented by step s (cf. line 13 in Fig. 4). To execute an operation, the planner first creates the proper input message, where all variables are replaced by the intended values. The message is then sent to the Web service and its response is retrieved and parsed. For each result r in $\text{Res}(op)$, the success condition is tested, if one exists. If no success condition violation is detected, the result (i.e. effect or knowledge effect) gets materialized (cf. Sect. 3.5) and the agent's fact-base is updated. Note that the conversation data sets are used to substitute variables of effect formulas by constants.

After the execution of the current step of the plan is finished, the next step is chosen (cf. line 11 in Fig. 4), and the check for causal link violations is carried out again. If at some point in the plan execution a causal link violation is detected, then the plan execution is aborted (lines 15, 16). The causal links whose violations were detected by $\text{CAUSAL-LINK-VIOLATIONS}$ remain in $\mathcal{AL}$ and will be used in the next planning-cycle.

As long as the threshold value is not reached (line 8), the replanning algorithm continues by generating another plan. Since the set $\mathcal{AL}$ contains information about causal link relationships that have turned out not to yield the desired results, the POP algorithm will avoid to rely on causal links that match any link in $\mathcal{AL}$. It will try to determine an alternative plan that avoids the mistakes of the previous attempt(s). If such an alternative plan is found, then the execution and monitoring of causal link violations starts again (lines 11-18).

Once the method SOLVE-PROBLEM manages to execute a plan *without* running into causal link violations, then the problem is *solved*.

6 Examples and Empirical Tests

For a brief empirical evaluation of the concepts presented in this article, consider a simple “web-shopping” domain, which consists of two online retailers *A* and *B* who offer several operations to browse their catalogs and to purchase items. The services and their WSDL & SESMA descriptions are available online at¹.

Shop *A* offers the following operations: *buyItem* purchases an item; the international EAN number of the item and user’s credit card information must be given. Operation *getPrice* delivers the price of an item, and *getItemList* returns the list of items that are in the shop’s catalog and can be purchased; each catalog item is described by title, EAN number and description. The operations *buyItem* and *getItemList* are depicted in Fig. 2.

Shop *B* has a different interface: it offers the method *getList* to retrieve a list of available items, described by EAN-number, title, price and description. Further, it uses a shopping cart metaphor for purchases: every item to be purchased needs to be put into a shopping cart first, by invocation of a method *addToCart*, which takes the EAN number of the item as input parameter. To remove an item from the cart, the method *removeFromCart* can be used, and to purchase the items in the cart the method *checkout* can be used, which requires credit card information as an input. Only registered users that are logged in and possess a valid session-ID for Shop *B* are permitted to use the *addToCart*, *removeFromCart* and *checkout* operations; to register at Shop *B*, the operation *register* needs to be called with information about the user, and to log in and retrieve a session token, the operation *login* needs to be called with the user credentials received from *register*.

Let us consider a scenario in which the goal $\langle \textit{possess item} = \textit{“123456”} \rangle$ should be fulfilled, whereby the predicate *possess* belongs to the vocabulary used in the markup of the two shopping services. We implemented a prototype of the architecture and algorithms presented in this paper and ran it against 4 different test cases, which are listed in Tab. 1: The cases are different instantiations of the scenario described above; they differ regarding the availability of the product with EAN-number *123456*. In case nr. 1, the item is available at both shops, in case nr. 2, it is only available at Shop *A*, in case nr. 3 it is only available at Shop *B* and in case nr. 4, none of the shops have the item in their article catalog.

As the column 4 of the table shows, the SOLVE-PROBLEM algorithm terminates in all four example cases; even in the unsolvable case it does not take more than three planning attempts for the agent to come to the correct conclusion. Column 5 shows the total time the agent had to spend to solve the problem and col. 6 shows the amount of time used solely for the planning (which excludes the time spent for interaction with the Web services). Column 7 shows the final results of the experiments, i.e. whether or not the agent was able to purchase the desired product.

¹ <http://wsplan.sf.net>

Table 1. Example cases computed on Intel Pentium IV, 1.6 GHz, running Linux 2.4

Case Nr.	Shop A	Shop B	Terminates	Time (s) Total	Time (s) Planning	Result
1	Item available	Item available	Yes, after 1	1.65	0.11	Item purchased
2	Item available	Item not avail.	Yes, after 1	1.65	0.11	Item purchased
3	Item not avail.	Item available	Yes, after 2	1.98	0.17	Item purchased
4	Item not avail.	Item not avail.	Yes, after 3	1.73	0.24	Item not purchased

As an example, we will briefly describe case 3 of our test series, because it illustrates how the replanning algorithm reacts to adverse conditions: At first, an initial plan is created: [*ShopA.getItemList*, *ShopA.buyItem*]. The agent executes the first operation and retrieves the list of items available at Shop A, which does not contain the wanted product. Before it executes the next operation, it checks the causal links and detects that *getItemList* did not manage to achieve the precondition for *buyItem*, which is that the item is in the catalog. Therefore, the planner aborts the plan execution, adds the newly detected causal link violation to $\mathcal{AL}$ and then requests a new plan from the adapted VHPOP planner. The entry in $\mathcal{AL}$ makes it impossible for the planner to find a new plan based on Shop A. It is therefore forced to backtrack and to chose a different planning branch, i.e. to consider shop B. Therefore, in the second attempt, a plan [*ShopB.register*, *ShopB.login*, *ShopB.addToCart*, *ShopB.checkout*] is devised. In case 3, this attempt indeed achieves the goal because Shop B has the product in stock. We can see from the example, that with each planning- and execution-cycle the planner gains feedback about the plan structures that lead to dead ends, which gradually improves its ability to find potent plans that achieve the given goal.

7 Related Work

In recent time, several approaches to applying AI techniques to the WSC problem have been published. A relevant aspect that helps differentiating the various approaches is the way they represent domain knowledge. On the one hand, it is accepted that the proper encoding of domain knowledge is a key requirement for efficient planning. On the other hand, manually encoded domain knowledge may hamper practical adoption, because of the efforts and skills required.

Existing work like [9] and [10] use manually encoded domain knowledge, in the form of HTN method descriptions and GOLOG programs, respectively. Our solution does not use manually encoded domain knowledge, instead it uses knowledge generated by dynamic *interaction* with the domain. In this sense, we follow the tradition of *replanning* agents, where feedback data is gained to better inform the heuristics of the agent (cf. e.g. [11]). It must be admitted, however, that the domain knowledge gained from feedback of plan executions does not keep up to the richness of manually encoded knowledge. Probably a combination of both approaches would suite the WSC problem best.

Another central issue is the representation of *goals*. In this context, our work draws from SADL [12] which introduced goal annotations similar to the ones we use. However, our algorithmic strategy to deal with those goals differs from the existing work: we encode *find-out* goals into a set $\mathcal{AL}$ of forbidden causal link patterns and resort to replanning techniques, while the previous SADL-based approaches like PUCCINI [12] chose to deal with the problem by extending POP's plan-, domain- and problem representation schemes, including the introduction of new link- and threat-types. Our current use cases and experiments justify our strategy, but a systematic comparison of the advantages and disadvantages of each approach has yet to be conducted.

Another novel approach to service composition was presented in [13], which applies the planning as *model checking* (MC) paradigm. In this approach, the goal specifies the conditions that must hold after plan execution which can include conditions about the plan itself. The ability to pose such conditions (e.g. safety and liveness properties) on the plan is an advantage of the planning as MC approach. While our set $\mathcal{AL}$ of *links to avoid* also offers a tool to specify safety conditions, our approach does not allow for liveness properties. However, this solution comes with a certain computational overhead [13] which may lead to long planning times; more work is required to find out which approach serves which problem domains best.

8 Summary

In this paper we described how a modified version of a state-of-the-art partial order planner, embedded in an execution monitoring engine, can be used to automatically solve problems in Web service domains. We have illustrated how the agent can use feedback gained from failed plan executions to avoid doomed plans in the next planning attempts.

We found that the POP model of plans, especially the explicit representation of a plan's causal links provides several benefits: Firstly, it allows for execution monitoring that can detect plan failures even before they become immanent. Secondly, it allows for *precise* learning. For instance, in the example discussed in Sect. 6, the planner does not blindly avoid *all* links between *ShopA.buy* and *ShopA.getItemList*; it only avoids links between the two operations where the literal to be achieved unifies with the literal of the failed link. In practical terms, this means that some action is avoided to buy a product *A* because of a former failure, but an attempt to purchase a product *B* can still be made.

Another contribution of this work is the support for nondeterministic actions using *success-conditions*. These are extra-logical conditions that can be attached to effect descriptions and are evaluated *after* an operation has been carried out. This way, agents can execute nondeterministic actions and easily determine the state they are in afterwards.

In future work we aim to clarify the issues raised in Sect. 7. Further, we plan to study extensions of the current *avoid-links* concept and to investigate *compensation actions* that may be required after failed attempts.

Acknowledgements

The author would like to thank Biplav Srivastava and Jana Koehler for their valuable feedback on earlier versions of the paper, Håkan Younes for the permission to extend VHPOP and the anonymous reviewers for their helpful comments.

References

1. W3C: Web Services Description Language (WSDL) Version 1.2 (2002)
2. Srivastava, B., Koehler, J.: Web Service Composition - Current Solutions and Open Problems. In: Proceedings of the ICAPS'03 Workshop on Planning for Web Services (2003)
3. Carman, M., Serafini, L., Traverso, P.: Web Service Composition as Planning. In: Proceedings of the ICAPS'03 Workshop on Planning for Web Services (2003)
4. OWL-S Coalition: OWL Web Services 1.1, <http://www.daml.org/services> (2004)
5. WSML Working Group: Web Service Modeling Language (WSML), <http://www.wsmo.org/wsml> (2004)
6. Peer, J.: Semantic Service Markup with SESMA. Language Specification, version 0.8, http://elektra.mcm.unisg.ch/pbwsc/docs/sesma_0.8.pdf (2004)
7. Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., Williamson, M.: An Approach to Planning with Incomplete Information. Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (1992)
8. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile Heuristic Partial Order planner. Journal of Artificial Intelligence Research (2003)
9. Hendler, J., Wu, D., Sirin, E., Nau, D., Parsia, B.: Automatic Web Services Composition Using SHOP2. In: Proceedings of The Second International Semantic Web Conference (ISWC) (2003)
10. McIlraith, S., Tran, C.S., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems (2001)
11. Haigh, K.Z.: Situation Dependent Learning for Interleaved Planning and Robot Execution (1998)
12. Golden, K.: Planning and Knowledge Representation for Softbots (1997)
13. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: Third International Semantic Web Conference ISWC'04. (2004)

Process-Level Composition of Executable Web Services: “On-the-fly” Versus “Once-for-all” Composition^{*}

Marco Pistore, Pierluigi Roberti, and Paolo Traverso

University of Trento and ITC-IRST

pistore@dit.unitn.it, {roberti, traverso}@itc.it

Abstract. Most of the work on automated composition of web services has focused so far on the problem of composition at the *functional level*, i.e., composition of atomic services that can be executed in a single request-response step. In this paper, we address the problem of automated composition at the *process level*, i.e., a composition that takes into account that executing a web service requires interactions that may involve different sequential, conditional, and iterative steps. We define two kinds of process-level composition problems: *on-the-fly compositions* that satisfy one-shot user requests specified as composition goals, and a more general form, called *once-for-all compositions*, whose goal is to build a general composed web service that is able to interact directly with the users, receive requests from them, and propose suitable answers. We propose a solution to these two kinds of process-level compositions, and apply the solution to the case of web services described in OWL-S. As a result, we automatically generate process-level compositions as executable OWL-S process models. We show that, while executable on-the-fly compositions can be described as standard OWL-S process models, once-for-all compositions need OWL-S process models to be extended with receive and reply constructs.

1 Introduction

The automated composition of web services is one of the most promising ideas and — at the same time — one of the main challenges for the taking off of service oriented applications: services that are composed automatically can perform new functionalities by interacting with existing services that are published on the web, thus significantly reducing the time and effort needed to develop new web based and service oriented applications. It has been widely recognized that one of the key elements for the automated composition of web services is semantics: unambiguous descriptions of web services capabilities and web service processes, e.g., in standard languages like OWL-S [3] or WSMO [6], can provide the ability to reason about web services, and to automate web services tasks, like web service discovery and composition, see, e.g., [9].

Most of the work on the composition of semantic web services has focused so far on the problem of composition at the *functional level*, i.e., composition of services

^{*} This work is partially funded by the MIUR-FIRB project RBNE0195K5, “Knowledge Level Automated Software Engineering”, and by the MIUR-PRIN 2004 project “Advanced Artificial Intelligence Systems for Web Services”.

which are considered “atomic” components described in terms of their inputs, outputs, preconditions, and effects, and which can be executed in a simple request-response step (see, e.g. [12, 4]). One of the key open problems for semantic web services is composition at the *process level*, i.e., the problem of generating automatically composed web services that can be directly executed to interact with the component services and achieve the composition goal. The problem of *process-level composition* is far from trivial. We need to take into account the fact that, in real cases, component services cannot in general be executed in a single request-response step. Component services are instead stateful processes, and they require to follow an interaction protocol which may involve different sequential, conditional, and iterative steps. For instance, we cannot in general interact with an “hotel booking” web service in an atomic step. The service may require a sequence of different operations including authentication, submission of a specific request for a room, possibly a negotiation of an offer, acceptance (or refusal) of the offer, and finally payment. These different steps may have conditional, or non-nominal outcomes, e.g., authentication can fail, or there may be no rooms available, and so on. Conditional outputs affect the flow of the interaction, e.g., no request can be submitted if the authentication fails, or, in the case no room is available, no request can be submitted. It may also be the case that the same operation can be repeated iteratively, e.g., in order to refine a request or to negotiate the conditions of the offer.

While the details on the exact protocol required to interact with an existing service are not important in functional-level composition, they become essential when we aim at generating composed web services that are executable. For instance, suppose we compose an “hotel booking” service and a “flight booking” service to obtain an composite service implementing a “virtual travel agency”. The composite service has to guarantee properties such as the fact that, if no available hotel is found, then we do not want to book the flights, or that the nights spent in the hotel are compatible with the arrival and departure dates of the flights. All these properties can be guaranteed only by “interleaving” in a suitable way the interaction protocols of “flight booking” and “hotel booking” (e.g., by booking the flights first, so that the arrival and departure dates are known, but by completing the interaction with “flight booking” and confirming the flight only after a suitable hotel room has been found). This example shows that process-level composition needs to deal with descriptions of web services in terms of complex, composite processes, that consist of arbitrary (conditional and iterative) combinations of atomic interactions, in the style, e.g., of OWL-S process models [3] or WSMO interfaces [6].

In this paper, we address the problem of process-level composition of OWL-S process models. Given a set of services that are available on the web and that are described with OWL-S process models, and given a composition goal that describes a set of desired requirements over the behavior of the composed web service to be generated, we generate automatically an executable composed web service, also described with an OWL-S process model. The composed web service, when executed, interacts with the available services in order to satisfy the goal.

More precisely, we address two different forms of process-level composition. In the first form, called *on-the-fly composition*, the composition is created in order to satisfy a specific request of the customer (e.g., a trip to a specific location in a specific period of

time). In this first form of composition, every time the “virtual travel agency” receives a customer’s request, it generates a new composition and executes it, thus obtaining an offer that can be given back to the customer. Moreover, the interactions with the customer of the “virtual travel agency” are forced to follow a simple request/response pattern: after the customer has sent a request, the agency interacts in a suitable way with the hotel and flight services and sends an offer back to the customer.

In the second form, called *once-for-all composition*, the objective of the composition task is to construct a general web service that is able to answer to different requests from customers. For instance, rather than composing the hotel booking and the flight booking services from a request for a specific location and period, the goal is to generate a service that accepts different travel requests, interacts with the customer to propose a suitable travel offer, and finalizes the offer if accepted by the customer. While in on-the-fly compositions the interactions with the customer are restricted to a request/response pattern, in once-for-all composition these interactions can be general. For instance, it is possible to generate a composition that, after receiving a request from the customer, asks the flight and hotel services for possible offers to be combined and sent back to the customer; the customer can now inspect the offers and decide whether to accept or refuse them; the virtual travel agency can confirm or cancel the hotel and flight offers after it receives the feedback from the customer. This way, not only the interactions with the component services (the hotel and flight services) are complex protocols, but also the interactions with the customer become complex, possibly conditional and iterative protocols. Technically speaking, the customer becomes one of the existing component services in input to the composition task.

In this paper we propose a solution to the problem of process-level composition, both in the case of on-the-fly and of once-for-all composition. We define a theoretical framework for automated process-level composition that can be used for both kinds of composition. We compare the on-the-fly composition and once-for-all composition problems, and show that the former is actually a simpler case of the latter. We apply the framework to the case of web services described in OWL-S. In the case of on-the-fly composition, we generate OWL-S process models that can be executed in standard execution engines for OWL-S, e.g., the Mindswap engine [10]. In the case of once-for-all composition, we show that standard OWL-S process models are not adequate to represent executable composed web services. We propose an extension to OWL-S that allows for the execution of web services generated by the automated once-for-all composition task.

2 Motivating Example

In our reference example we have two separate, independent, and existing component services: a “hotel booking” and a “flight booking” service. We aim at composing them into a “virtual travel agency” service that, according to a user travel request, books both hotels and flights.

The **HotelBooking** service accepts requests for booking a room for a given period of time and location, and if at least one hotel is available, it proposes an offer for a given hotel at a certain cost. This offer can be accepted or refused by the external service that

```

<process:CompositeProcess rdf:ID="HotelBooking">
  <process:Sequence>
    <process:AtomicProcess rdf:about="#HotelRequest">
      <process:Input rdf:ID="Period"><process:parameterType rdf:resource="#Period"/></process:Input>
      <process:Input rdf:ID="Location"><process:parameterType rdf:resource="#Location"/></process:Input>
      <process:ConditionalOutput rdf:ID="Cost">
        <process:coCondition rdf:resource="#HotelBookingPossible"/>
        <process:parameterType rdf:resource="#Cost"/>
      </process:ConditionalOutput>
      <process:ConditionalOutput rdf:ID="Hotel">
        <process:coCondition rdf:resource="#HotelBookingPossible"/>
        <process:parameterType rdf:resource="#Hotel"/>
      </process:ConditionalOutput>
      <process:ConditionalOutput rdf:ID="NA">
        <process:coCondition rdf:resource="#NotHotelBookingPossible"/>
        <process:parameterType rdf:resource="#NotAvailable"/>
      </process:ConditionalOutput>
    </process:AtomicProcess>
    <process:CompositeProcess>
      <process:IfThenElse>
        <process:ifCondition rdf:resource="#HotelBookingPossible"/>
        <process:then>
          <process:CompositeProcess>
            <process:Choice>
              <process:AtomicProcess rdf:ID="#AcceptHotelOffer"/>
              <process:AtomicProcess rdf:ID="#RefuseHotelOffer"/>
            </process:Choice>
          </process:CompositeProcess>
        </process:then>
      </process:IfThenElse>
    </process:CompositeProcess>
  </process:Sequence>
</process:CompositeProcess>

<process:condition rdf:ID="HotelBookingPossible">
  <expr:expressionBody>
    #AvailableHotel(#HotelRequest.Period,#HotelRequest.Location) != UNDEF
  </expr:expressionBody>
</process:condition>
...
<process:sameValues rdf:parseType="Collection">
  <process:ValueOf>
    <process:theParameter rdf:resource="#Hotel"/>
    <process:atProcess rdf:resource="#HotelRequest"/></process:ValueOf>
  <process:valueData>
    <expr:expressionBody>
      #AvailableHotel(#HotelRequest.Period,#HotelRequest.Location)
    </expr:expressionBody>
  </process:valueData>
</process:sameValues>
...

```

Fig. 1. OWL-S Process model for the hotel booking service

has invoked the hotel service. The OWL-S process model for the hotel service is shown in Figure 1. It is a composite service consisting of the atomic process `HotelRequest`, `AcceptHotelOffer`, and `RefuseHotelOffer`. `HotelRequest` receives in input a request for a given `Period` and `Location`. The conditional outputs model the fact the service returns an offer including the price (`Cost`) and the name of the hotel (`Hotel`) only if there exists a hotel with available rooms; a “not available” (`NA`) message is returned otherwise. The decision whether there exists a hotel with available rooms is taken according to condition `HotelBookingPossible`, which is defined in terms of function `AvailableHotel( $p, l$ )`: this function returns a hotel name if the booking is possible for a period p and location l , and `UNDEF` otherwise (see definition of condition `HotelBookingPossible`). Function `AvailableHotel( $p, l$ )` is used also to decide the value of output parameter `Hotel` of service `HotelRequest` (see the `process:sameValues` declaration). Similarly, output parameter `Cost` is defined in terms of function `CostOfRoom( $p, h$ )`, returning the cost of a room for period p in hotel h (this declaration has been omitted in Figure 1). If the booking is possible (see the control construct `IfThenElse` and the condition `HotelBookingPossible`), the hotel service waits for a nondeterministic external

decision (control construct `Choice`) that either accepts (`AcceptHotelOffer`) or refuses (`RefuseHotelOffer`) the booking offer.

The `FlightBooking` service is conceptually similar to the hotel booking one: it accepts requests for booking a flight for a given period and location, and if the flight is available, returns an offer with a cost and a flight. This offer can be accepted or refused by the external service that has invoked the flight booking service. Its OWL-S process model has a similar structure too. It is a composition of the following atomic processes: `FlightRequest`, `AcceptFlightOffer` and `RefuseFlightOffer`. For lack of space we omit the OWL-S description.

Let us now consider an example of *on-the-fly composition*. Our goal is to construct a composed service (say `H&F`) that, given in input a specific location and time period, interacts with the hotel and the flight service, and books both a flight and an hotel room whenever possible. A possible OWL-S process model for the `H&F` service is described in Figure 2. The OWL-S specification starts with a declaration of the input (`Period` and `Location`) and output (`Cost`, `Hotel`, and `Flight`, or `NA`) parameters of the composition. Then the body of the composition is defined as a suitable interleaving of the atomic services of the `HotelBooking` and `FlightBooking` services. In this example, we choose a `H&F` service that interacts first with the hotel and then with the flight, therefore, the atomic process `HotelRequest` is called first.¹ If the booking is possible, the condition `HotelBookingPossible` holds and the `H&F` process starts interacting with the flight service (`FlightRequest`). If a flight is available (condition `FlightBookingPossible`) the `H&F` process accepts both the offer of the hotel booking service (`AcceptHotelOffer`) and the one of the flight booking service (`AcceptFlightOffer`). If the flight is not available, `H&F` needs to refuse just the hotel offer (`RefuseHotelOffer`) previously received. We remark that condition `HotelBookingPossible` is defined in terms of the contents of the answer received from atomic service `HotelRequest`. Indeed, the hotel booking is possible if and only if the `NA` does not appear in the answer.

Let us consider now an example of *once-for-all composition*. This time our goal is to construct a composed service (say `H&Fservice`) that interacts with the customer and with the component services as follows. `H&Fservice` gets from the user a travel request for a given location and period. It then interacts with the hotel and flight services to discover whether a hotel and a flight are available; if this is the case, `H&Fservice` interacts again with the user by offering the available hotel and flight at a certain price. At this point, if the user accepts the offer, `H&Fservice` books both the hotel and the flight. If the offer is refused, we assume that the `H&Fservice` service simply cancels both the hotel and the flight requests.

The main conceptual difference of “once-for-all” composition w.r.t. “on-the-fly” composition is the fact that the composed service `H&Fservice` has to interact in a

¹ We can notice that the `Period` parameter passed to the `HotelRequest` process coincides with the `Period` input parameter of the composed process (see the `process:sameValues` declaration at the end of the figure). Similar correspondences, omitted in Figure 2, are also defined for the input and output parameters of the other atomic processes that appear in the composition.

```

<process:CompositeProcess rdf:ID="H&F">
  <process:hasInput rdf:resource="#Period"/>
  <process:hasInput rdf:resource="#Location"/>
  <process:hasResult rdf:resource="#Cost"/>
  <process:hasResult rdf:resource="#Hotel"/>
  <process:hasResult rdf:resource="#Flight"/>
  <process:hasResult rdf:resource="#NA"/>

  <process:Sequence>
    <process:AtomicProcess rdf:about="#HotelRequest"/>
    <process:CompositeProcess>
      <process:IfThenElse>
        <process:ifCondition rdf:resource="#HotelBookingPossible"/>
        <process:then>
          <process:CompositeProcess>
            <process:Sequence>
              <process:AtomicProcess rdf:about="#FlightRequest"/>
              <process:CompositeProcess>
                <process:IfThenElse>
                  <process:ifCondition rdf:resource="#FlightBookingPossible"/>
                  <process:then>
                    <process:CompositeProcess>
                      <process:Sequence>
                        <process:AtomicProcess rdf:about="#AcceptHotelOffer"/>
                        <process:AtomicProcess rdf:about="#AcceptFlightOffer"/>
                      </process:Sequence>
                    </process:CompositeProcess>
                  </process:then>
                  <process:else>
                    <process:AtomicProcess rdf:about="#RefuseHotelOffer"/>
                  </process:else>
                </process:IfThenElse>
              </process:CompositeProcess>
            </process:Sequence>
          </process:CompositeProcess>
        </process:then>
        <process:else>
          <process:AtomicProcess rdf:about="#RefuseHotelOffer"/>
        </process:else>
      </process:IfThenElse>
    </process:CompositeProcess>
  </process:Sequence>
</process:CompositeProcess>

<process:condition rdf:ID="HotelBookingPossible">
  <expr:expressionBody>
    #HotelRequest.NA = UNDEF
  </expr:expressionBody>
</process:condition>
...
<process:sameValues rdf:parseType="Collection">
  <process:ValueOf>
    <process:theParameter rdf:resource="#Period"/>
    <process:atProcess rdf:resource="#HotelAndFlight"/>
  </process:ValueOf>
  <process:ValueOf>
    <process:theParameter rdf:resource="#Period"/>
    <process:atProcess rdf:resource="#HotelRequest"/>
  </process:ValueOf>
</process:sameValues>?
...

```

Fig. 2. OWL-S Process model for on-the-fly composition: the H&F service

complex way also with the user. More precisely, we can assume that the interactions between the user and the virtual travel agency are modeled as a composite OWL-S process model. According to this process model, the user has to call first a `H&FRequest` atomic web service, passing period and location as parameters; if the requested booking is possible, an answer including hotel name, flight number and cost is returned, otherwise a “not available” answer is returned. In case of positive answer, the user can decide whether to confirm the offer (calling service `AcceptH&FOffer`) or to cancel it (calling service `RefuseH&FOffer`).

The OWL-S process model for the `H&Fservice` service is described in Figure 3. It is defined as a suitable combination of the atomic web services corresponding to `HotelBooking`, `FlightBooking`, and to the process defining the virtual travel agency service (`H&FRequest`, `AcceptH&FOffer`, and `RefuseH&FOffer`).

```

<process:CompositeProcess rdf:ID="HotelAndFlightGeneration">
  <process:Sequence>
    <process:Receive rdf:about="#H&FRequest">
      <process:AtomicProcess rdf:about="#HotelRequest"/>
    <process:CompositeProcess>
      <process:IfThenElse>
        <process:ifCondition rdf:resource="#HotelBookingPossible"/>
        <process:then>
          <process:Sequence>
            <process:AtomicProcess rdf:about="#FlightRequest"/>
            <process:CompositeProcess>
              <process:IfThenElse>
                <process:ifCondition rdf:resource="#FlightBookingPossible"/>
                <process:then>
                  <process:CompositeProcess>
                    <process:Sequence>
                      <process:Reply rdf:about="#H&FRequest">
                      <process:Choice>
                        <process:CompositeProcess>
                          <process:Sequence>
                            <process:Receive rdf:about="#AcceptH&FOffer"/>
                            <process:AtomicProcess rdf:about="#AcceptHotelOffer"/>
                            <process:AtomicProcess rdf:about="#AcceptFlightOffer"/>
                            <process:Reply rdf:about="#AcceptH&FOffer"/>
                          </process:Sequence>
                        </process:CompositeProcess>
                      <process:CompositeProcess>
                        <process:Sequence>
                          <process:Receive rdf:about="#RefuseH&FOffer"/>
                          <process:AtomicProcess rdf:about="#RefuseHotelOffer"/>
                          <process:AtomicProcess rdf:about="#RefuseFlightOffer"/>
                          <process:Reply rdf:about="#RefuseH&FOffer"/>
                        </process:Sequence>
                      </process:CompositeProcess>
                    </process:Choice>
                  </process:Sequence>
                </process:CompositeProcess>
              </process:then>
            <process:else>
              <process:CompositeProcess>
                <process:Sequence>
                  <process:Reply rdf:about="#H&FRequest"/>
                  <process:AtomicProcess rdf:about="#RefuseHotelOffer"/>
                </process:Sequence>
              </process:CompositeProcess>
            </process:else>
          </process:IfThenElse>
        </process:CompositeProcess>
      </process:Sequence>
    </process:then>
  <process:else>
    <process:CompositeProcess>
      <process:Sequence>
        <process:Reply rdf:about="#H&FRequest"/>
      </process:Sequence>
    </process:CompositeProcess>
  </process:IfThenElse>
</process:CompositeProcess>

```

Fig. 3. OWL-S Process model for once-for-all composition: the H&Fservice service

Notice that the **H&Fservice** process needs to get the location and period from the user, and then to invoke the hotel and flight service. Similarly, the **H&Fservice** process needs to propose to the user an offer, and then to either cancel or confirm the bookings of the hotel and flight. These kinds of interactions cannot be modeled with usual OWL-S atomic processes. While OWL-S process models allow for defining a suitable combination of invocations of atomic operations provided by the component services, they do not allow for intermixing these invocations with interactions with the user, since OWL-S process models do not allow to model a process that is both “invoked by” and “invoker of” external services. In order to solve this problem, we introduce new constructs that allow the process to **Receive** messages and **Reply** to messages corresponding to atomic services that the composition should provide to the user.

Indeed, the first step of the **H&Fservice** process is to wait for a user request (process:Receive rdf:about="#H&FRequest"). The rest of the process

is the same as the one of the H&F service, but the last steps in the sequence: if the hotel and flight booking is possible, the process replies to the booking request of the user with an offer (`Process:Reply rdf:about="#H&FRequest"`). At this point the H&Fservice waits for a nondeterministic external decision (control construct *Choice*) that either accepts or refuses the offer. If the offer is accepted by the user, the atomic process `AcceptH&FOffer` is activated and then the hotel and flight booking are confirmed. The `Reply` statement communicates to the user that the booking is completed. If the offer is not accepted, the atomic process `RefuseH&FOffer` is activated, and the hotel and flight bookings are refused as well. In this case the `Reply` statement communicates to the user that the booking has been cancelled.

3 Theoretical Framework

Our goal is to automatically generate a new service W (called the *composed service*) that interacts with a set of published web services $W_1, \dots, W_n$ (called the *component services*) and satisfies a given composition goal. More specifically, we start from n process-level descriptions of web services $W_1, \dots, W_n$, e.g., their process models in OWL-S, and we automatically translate each of them into a *state transition system* (STS form now on): $\Sigma_{W_1}, \dots, \Sigma_{W_n}$. Intuitively, each Σ_{W_i} is a compact representation of all the possible behaviors, evolutions of the component service W_i . We then construct a *parallel STS* $\Sigma_{\parallel}$ that combines $\Sigma_{W_1}, \dots, \Sigma_{W_n}$ and represents all the possible evolutions of these component services, without any control by and interaction with the composed service that will be generated. We also formalize the requirements for the composed service as a composition goal, say ρ . Intuitively, ρ describes the functionality that the composed service should have. Given $\Sigma_{\parallel}$ and ρ , we automatically generate a STS Σ_c that encodes the new service W that has to be generated. Σ_c represents a service that dynamically receives and sends messages from/to the component services $W_1, \dots, W_n$ and behaves depending on the responses received from $W_1, \dots, W_n$. Σ_c is such that $\Sigma_c \triangleright \Sigma_{\parallel}$ satisfies the composition goal ρ , where $\Sigma_c \triangleright \Sigma_{\parallel}$ represents all the evolutions of the component services $\Sigma_{\parallel}$ as they are controlled by the composed service Σ_c . The STS Σ_c is then automatically translated into an executable web service, e.g., described as an (extension of a) OWL-S process model.

The formal definition of the process-level composition problem is based on the notion of *state transition system* (STS). STSs are general models for defining dynamic systems that can be in different *states* (some of which are marked as *initial states*) and can evolve to new states as a result of performing some *actions*. Actions are distinguished in *input actions*, which represent the reception of messages, *output actions*, which represent messages sent to external services, and a special action τ , called *internal action*. The action τ is used to represent internal evolutions that are not visible to external services, i.e., the fact that the state of the system can evolve without producing any output, and independently from the reception of inputs. A *transition relation* describes how the state can evolve on the basis of inputs, outputs, or of the internal action τ . Finally, a *labeling function* associates to each state the set of properties $\mathcal{Prop}$ that hold in the state. These properties will be used to define the composition goal ρ .

```

PROCESS HotelBooking;
TYPE  Period; Location; Hotel; Cost; NA;
STATE pc: { start, receiveHotelBooking, checkHotelBookingPossible, isHotelBookingPossible,
            isNotHotelBookingPossible, replyHotelBookingPossible, replyHotelBookingNotPossible,
            choiceAcceptHotelOfferRefuseHotelOffer, replyAcceptHotelOffer, replyRefuseHotelOffer,
            endHotelBookingNotPossible, endRefuseHotelOffer, endAcceptHotelOffer};

    period: Period  $\cup$  { UNDEF };
    loc: Location  $\cup$  { UNDEF };
    hotel: Hotel  $\cup$  { UNDEF };
    cost: Cost  $\cup$  { UNDEF };
    na: NA  $\cup$  { UNDEF };
    AvailableHotel[Period,Location]: Hotel  $\cup$  { UNDEF };
    CostOfRoom[Period,Hotel]: Cost;

INIT  pc = start;
      reg_period = UNDEF;
      reg_loc = UNDEF;
      offer_hotel = UNDEF;
      offer_cost = UNDEF;

INPUT  HotelBooking(Period, Location);
        AcceptHotelOffer();
        RefuseHotelOffer();

OUTPUT HotelBookingAnswer(Hotel  $\cup$  {UNDEF}, Cost  $\cup$  {UNDEF}, NA  $\cup$  {UNDEF});
        AcceptHotelOfferAnswer();
        RefuseHotelOfferAnswer();

TRANS pc = isHotelBookingPossible  $\rightarrow$  [TAU]  $\rightarrow$  pc = replyHotelBookingPossible,
pc = start  $\rightarrow$  [TAU]  $\rightarrow$  pc = receiveHotelBooking;
pc = receiveHotelBooking  $\rightarrow$  [INPUT HotelBooking(period,location)]  $\rightarrow$  pc = checkHotelBookingPossible;
pc = checkHotelBookingPossible  $\wedge$  AvailableHotel[period,location]  $\neq$  UNDEF
     $\rightarrow$  [TAU]  $\rightarrow$  pc = isHotelBookingPossible;
pc = checkHotelBookingPossible  $\wedge$  AvailableHotel[period,location] = UNDEF
     $\rightarrow$  [TAU]  $\rightarrow$  pc = isNotHotelBookingPossible;
pc = isHotelBookingPossible  $\rightarrow$  [TAU]  $\rightarrow$  pc = replyHotelBookingPossible,
    hotel = AvailableHotel[period,location],
    cost = CostOfRoom[period,hotel];
pc = replyHotelBookingPossible  $\rightarrow$  [OUTPUT HotelBookingAnswer(hotel,cost,na)]  $\rightarrow$ 
    pc = choiceAcceptHotelOfferRefuseHotelOffer;
pc = choiceAcceptHotelOfferRefuseHotelOffer  $\rightarrow$  [INPUT AcceptHotelOffer]  $\rightarrow$  pc = replyAcceptHotelOffer;
pc = replyAcceptHotelOffer  $\rightarrow$  [OUTPUT AcceptHotelOfferAnswer]  $\rightarrow$  pc = endAcceptHotelOffer;
pc = choiceAcceptHotelOfferRefuseHotelOffer  $\rightarrow$  [INPUT RefuseHotelOffer]  $\rightarrow$  pc = replyRefuseHotelOffer;
pc = replyRefuseHotelOffer  $\rightarrow$  [OUTPUT RefuseHotelOfferAnswer]  $\rightarrow$  pc = endRefuseHotelOffer;
pc = isNotHotelBookingPossible  $\rightarrow$  [TAU]  $\rightarrow$  pc = replyHotelBookingNotPossible, na  $\in$  NA;
pc = replyHotelBookingNotPossible  $\rightarrow$  [OUTPUT HotelBookingAnswer(hotel,cost,na)]  $\rightarrow$ 
    pc = endHotelBookingNotPossible;

```

Fig. 4. The STS for the HotelBooking process

Definition 1 (State transition system (STS)). A state transition system Σ is a tuple $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{S}^0 \subseteq \mathcal{S}$ is the set of initial states, $\mathcal{I}$ is the set of input actions, $\mathcal{O}$ is the set of output actions, $\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ is the transition relation, and $\mathcal{L} : \mathcal{S} \rightarrow 2^{P^{rop}}$ is the labeling function.

We assume that infinite loops of τ -transitions cannot appear in the system (i.e., the service has to interacting with the environment after a finite number of steps). We also assume that there is no state which originates both input and output transitions.

Figure 4 shows a description of the STS corresponding to the HotelBooking web service (see Figure 1). The set of states $\mathcal{S}$ models the steps of the evolution of the process and the values of its variables. The special variable pc implements a “program counter” that holds the current execution step of the service (e.g., pc has value `receiveHotelBooking` when the process is waiting to receive a booking request, and value `checkHotelBookingPossible` when it is ready to check whether the booking is possible). Other variables like `location` or `cost` correspond to those used in message exchanges. Finally, arrays like `AvailableHotel` or `CostOfRoom` describe predicates and functions expressing properties of the web service (e.g., the fact that there is an hotel available for a given period and in a given location, or the cost of a room in a given hotel for a given period). In the initial states $\mathcal{S}^0$ the pc is set to `START`, while all the other basic variables are undefined. The initial values of the

arrays `AvailableHotel` and `CostOfRoom` are unspecified, since they can assume any value in the domain.

The evolution of the process is modeled through a set of possible transitions. Each transition defines its applicability conditions on the source state, its firing action, and the destination state. For instance,

$$\begin{aligned} &pc = \text{checkHotelBookingPossible} \ \& \\ &\text{AvailableHotel}[\text{period}, \text{location}] \neq \text{UNDEF} \rightarrow [\text{TAU}] \rightarrow \\ &pc = \text{isHotelBookingPossible} \end{aligned} \quad (1)$$

asserts that an action τ can be executed in state `checkHotelBookingPossible`, leading to the state `isHotelBookingPossible`, if there is some hotel with available rooms for the specified period and location. We remark that each TRANS clause of Figure 4 corresponds to different elements in the transition relation $\mathcal{R}$: e.g., clause (1) generates different elements of $\mathcal{R}$, depending on the values of variables `period` and `location`.

According to the formal model, we distinguish among three different kinds of actions. The input actions $\mathcal{I}$ model all the incoming requests to the process and the information they bring (i.e., `HotelBooking` is used for the receiving of the booking request). The output actions $\mathcal{O}$ represent the outgoing messages (in our case, the replies to the input requests). The action τ is used to model internal evolutions of the process, as for instance assignments and decision making.

Finally, the set of properties $\mathcal{Prop}$ of the STS are expressions of the form $\langle \text{variable} \rangle = \langle \text{value} \rangle$ or $\langle \text{array} \rangle [\text{idx}_1, \dots, \text{idx}_n] = \langle \text{value} \rangle$, and the labeling function is the obvious one.

A remark is in order. The definition of STS provided in Figure 4 is parametric w.r.t. the types `Period`, `Location`, `Hotel`, `Cost`, and `NA` used in the messages. In order to obtain a concrete STS and to apply the automated composition techniques described in Section 4, specific ranges have to be assigned to these types. Different approaches are possible for defining these ranges. The simpler approach is to associate finite (and possibly small) ranges to each type. This approach, makes the definition of the STS easy (and allows for an efficient automated composition), however, it has the disadvantage of imposing unrealistic assumptions on the data types handled by the web services. A more realistic (but more complex) approach consists of using abstract models for the data types, avoiding an enumeration of all concrete values that these types can assume, and representing explicitly only those aspects of the data type that are relevant for the task at hand. This second approach will be addressed in future work (see Section 5).

We now formally define the *parallel product* $\Sigma_1 \parallel \Sigma_2$ of the two STSs Σ_1 and Σ_2 . It models the fact that the systems may evolve independently (i.e., without direct communications), and is used to generate $\Sigma_{\parallel}$ from the component web services $W_1, \dots, W_n$: formally, $\Sigma_{\parallel} = \Sigma_{W_1} \parallel \dots \parallel \Sigma_{W_n}$.

Definition 2 (parallel product). Let $\Sigma_1 = \langle \mathcal{S}_1, \mathcal{S}_1^0, \mathcal{I}_1, \mathcal{O}_1, \mathcal{R}_1, \mathcal{L}_1 \rangle$ and $\Sigma_2 = \langle \mathcal{S}_2, \mathcal{S}_2^0, \mathcal{I}_2, \mathcal{O}_2, \mathcal{R}_2, \mathcal{L}_2 \rangle$ be two STSs with $(\mathcal{I}_1 \cup \mathcal{O}_1) \cap (\mathcal{I}_2 \cup \mathcal{O}_2) = \emptyset$. The *parallel product* $\Sigma_1 \parallel \Sigma_2$ of Σ_1 and Σ_2 is defined as:

$$\Sigma_1 \parallel \Sigma_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, \mathcal{S}_1^0 \times \mathcal{S}_2^0, \mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{O}_1 \cup \mathcal{O}_2, \mathcal{R}_1 \parallel \mathcal{R}_2, \mathcal{L}_1 \parallel \mathcal{L}_2 \rangle$$

where:

- $\langle (s_1, s_2), a, (s'_1, s_2) \rangle \in (\mathcal{R}_1 \parallel \mathcal{R}_2)$ if $\langle s_1, a, s'_1 \rangle \in \mathcal{R}_1$;
- $\langle (s_1, s_2), a, (s_1, s'_2) \rangle \in (\mathcal{R}_1 \parallel \mathcal{R}_2)$ if $\langle s_2, a, s'_2 \rangle \in \mathcal{R}_2$;

and $(\mathcal{L}_1 \parallel \mathcal{L}_2)(s_1, s_2) = \mathcal{L}_1(s_1) \cup \mathcal{L}_2(s_2)$.

The automated composition problem consists in generating a STS Σ_c that controls $\Sigma_{\parallel}$ by satisfying the composition requirement ρ . We now define formally the STS describing the behaviors of a STS Σ when controlled by Σ_c .

Definition 3 (controlled system). Let $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$ and $\Sigma_c = \langle \mathcal{S}_c, \mathcal{S}_c^0, \mathcal{O}, \mathcal{I}, \mathcal{R}_c, \mathcal{L}_0 \rangle$ be two state transition systems, where $\mathcal{L}_0(s_c) = \emptyset$ for all $s_c \in \mathcal{S}_c$. The STS $\Sigma_c \triangleright \Sigma$, describing the behaviors of system Σ when controlled by Σ_c , is defined as:

$$\Sigma_c \triangleright \Sigma = \langle \mathcal{S}_c \times \mathcal{S}, \mathcal{S}_c^0 \times \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}_c \triangleright \mathcal{R}, \mathcal{L} \rangle$$

where:

- $\langle (s_c, s), \tau, (s'_c, s') \rangle \in (\mathcal{R}_c \triangleright \mathcal{R})$ if $\langle s_c, \tau, s'_c \rangle \in \mathcal{R}_c$;
- $\langle (s_c, s), \tau, (s_c, s') \rangle \in (\mathcal{R}_c \triangleright \mathcal{R})$ if $\langle s, \tau, s' \rangle \in \mathcal{R}$;
- $\langle (s_c, s), a, (s'_c, s') \rangle \in (\mathcal{R}_c \triangleright \mathcal{R})$, with $a \neq \tau$, if $\langle s_c, a, s'_c \rangle \in \mathcal{R}_c$ and $\langle s, a, s' \rangle \in \mathcal{R}$.

Notice that we require that the inputs of Σ_c coincide with the outputs of Σ and vice-versa. Notice also that, although the systems are connected so that the output of one is associated to the input of the other, the resulting transitions in $\mathcal{R}_c \triangleright \mathcal{R}$ are labelled by input/output actions. This allows us to distinguish the transitions that correspond to τ actions of Σ_c or Σ from those deriving from communications between Σ_c and Σ . Finally, notice that we assume that Σ_c has no labels associated to the states.

A STS Σ_c may not be adequate to control a system Σ . Indeed, we need to guarantee that, whenever Σ_c performs an output transition, then Σ is able to accept it, and vice-versa. We define the condition under which a state s of Σ is able to accept a message. We assume that s can accept a message a if there is some successor s' of s in Σ , reachable from s through a chain of τ transitions, such that s can perform an input transition labelled with a . Vice-versa, if state s has no such successor s' , and message a is sent to Σ , then a deadlock situation is reached. In the following definition, and in the rest of the paper, we denote by $\tau\text{-closure}(s)$ the set of the states reachable from s through a sequence of τ transitions, and by $\tau\text{-closure}(S)$ with $S \subseteq \mathcal{S}$ the union of $\tau\text{-closure}(s)$ on all $s \in S$.

Definition 4 (deadlock-free controller). Let $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$ be a STS and $\Sigma_c = \langle \mathcal{S}_c, \mathcal{S}_c^0, \mathcal{O}, \mathcal{I}, \mathcal{R}_c, \mathcal{L}_0 \rangle$ be a controller for Σ . Σ_c is said to be deadlock free for Σ if all states $(s_c, s) \in \mathcal{S}_c \times \mathcal{S}$ that are reachable from the initial states of $\Sigma_c \triangleright \Sigma$ satisfy the following conditions:

- if $\langle s, a, s' \rangle \in \mathcal{R}$ with $a \in \mathcal{I}$ then there is some $s'_c \in \tau\text{-closure}(s_c)$ such that $\langle s'_c, a, s'' \rangle \in \mathcal{R}$ for some $s'' \in \mathcal{S}_c$; and
- if $\langle s_c, a, s'_c \rangle \in \mathcal{R}_c$ with $a \in \mathcal{O}$ then there is some $s' \in \tau\text{-closure}(s)$ such that $\langle s', a, s'' \rangle \in \mathcal{R}$ for some $s'' \in \mathcal{S}$.

The automated composition task needs to generate a deadlock-free Σ_c that guarantees the satisfaction of a composition goal ρ . This is formalized by requiring that the controlled system $\Sigma_c \triangleright \Sigma_{\parallel}$ must satisfy ρ , written $\Sigma_c \triangleright \Sigma_{\parallel} \models \rho$. The definition of this requirement is technical: it depends on the definition of the executions of $\Sigma_c \triangleright \Sigma_{\parallel}$, and these have to be defined taking into account the special role of τ actions, which describe internal, “unobservable” evolutions of the system. For lack of space, we omit the formal definition of $\Sigma_c \triangleright \Sigma_{\parallel} \models \rho$: the interested reader can find it in [14].

Definition 5 (composition problem). *Let $\Sigma_1, \dots, \Sigma_n$ be a set of state transition systems, and let ρ be a composition goal. The composition problem for $\Sigma_1, \dots, \Sigma_n$ and ρ is the problem of finding a controller Σ_c that is deadlock-free and such that $\Sigma_c \triangleright (\Sigma_1 \parallel \dots \parallel \Sigma_n) \models \rho$.*

4 Automated Process-Level Composition

The automated process-level composition of web services is obtained in four steps:

1. **From Web Services to State Transition Systems.** The process-level descriptions of the available component web services are translated into STSs.
2. **Expressing Composition Goals.** This step consists in the formalization of the goal ρ that defines the functionality that the composed web service should provide.
3. **Synthesis of the Composition.** During this step, the STS implementing the composed web services is automatically generated starting from the STSs constructed in step 1 and from the goal specified in step 2.
4. **Deployment and Execution of the Composed Service.** In this step, the STS generated in step 3 is translated into an executable web service.

In this paper we consider web services described in OWL-S. The translation from OWL-S process models to STS (step 1) is performed along the lines described in Section 3 (see also [16]). More precisely, the STS of Figure 4 is obtained from the OWL-S process model in Figure 1 (the STS has been slightly edited for improving readability).

The definition of the composition goal (step 2) depends on the kind of process-level composition we are interested in. In the case of *on-the-fly composition*, we have to generate a service that satisfies a specific customer’s request. For instance, in the example of the hotel and flight services, the customer’s request specifies a given location l to be visited in a given period of time p . In order to satisfy this request, the composed service has to find a flight and a hotel room compatible with the request of the customer. The composition goal could be the something like “if a travel offer is available, then sell a travel offer to the customer”. In our example, the offer is possible if it is possible to book a flight and a hotel for the period and for the location specified by the customer. The fact that the offer is sold is described by requiring that the *HotelBooking* and the *FlightBooking* processes reach the end state corresponding to an accepted booking offer. The goal condition can hence be specified by the formula:

```

HotelBooking.AvailableHotel[p,l] ≠ UNDEF ∧
FlightBooking.AvailableFlight[p,l] ≠ UNDEF
→ HotelBooking.pc = endAcceptHotelBooking ∧
   FlightBooking.pc = endAcceptFlightBooking ∧
   h = HotelBooking.AvailableHotel[p,l] ∧
   f = FlightBooking.AvailableFlight[p,l] ∧
   c = HotelBooking.CostOfRoom[p,h] +
     FlightBooking.CostOfFlight[f]

```

where c , f , and h describe, respectively, the cost of the offer, and the specific flight and hotel information. This goal has to be interpreted as a condition that must hold at the end of the execution of the composed service.

The case of *once-for-all composition* is more complex. In our reference example, we want to automatically generate a composed service whose requirement is to “sell a travel offer to the customer” that satisfies a generic customer request. This means we want the composed service to reach the situation where an offer has been made to the customer, the customer has confirmed this offer, and the service has confirmed the corresponding (sub-)offers to the **HotelBooking** and **FlightBooking** services. However, the hotel may have no available rooms, the flight may not be possible, the user may not accept the offer due to its cost... We cannot avoid these situations, and we cannot therefore ask the composed service to guarantee this requirement. Nevertheless, we would like the composed service to *try* (do whatever is possible) to satisfy the request. If it is not possible to satisfy the main requirement, i.e., to sell the offer, our requirement for the composed service is to book neither a room nor a flight. Indeed we do not want the service to book and pay for rooms when flights are not available, or to book flights when rooms are not available, or to pay for rooms and flights that will not be accepted by the customer. Our composition goal is therefore a combination of the main requirement (“sell a travel offer to the customer”) and of a secondary requirement (“book neither a room nor a flight”), i.e., something like: “*try* to sell a travel offer to the customer; *upon failure*, do book neither a room nor a flight”. Notice that the secondary requirement (“book neither a room nor a flight”) has a different strength w.r.t. the primary one (“sell a travel offer to the customer”). We write “*do*” satisfy, rather than “*try*” to satisfy. Indeed, in the case the primary requirement is not satisfied, we want the secondary requirement to be guaranteed.

We need a formal language that can express requirements of this kind, including conditions of different strengths (like “*try*” and “*do*”), preferences among different (e.g., primary and secondary) requirements, and failure recovery conditions. For this reason, we cannot simply use a state formula as we did for the case of on-the-fly composition. We use instead the EAGLE language, which has been designed with the purpose to satisfy such expressiveness. A detailed definition and a formal semantics for the EAGLE language can be found in [5]. Here we just explain how EAGLE can express the composition requirement of the running example. The EAGLE formalization of the requirement is the following:

TryReach

```

HotelBooking.pc = endAcceptHotelBooking ∧
FlightBooking.pc = endAcceptFlightBooking ∧
Customer.pc = endAcceptHandFBooking ∧
Customer.h = HotelBooking.AvailableHotel[Customer.p, Customer.l] ∧
Customer.f = FlightBooking.AvailableFlight[Customer.p, Customer.l] ∧
Customer.c = HotelBooking.CostOfRoom[Customer.p, Customer.h] +
  FlightBooking.CostOfFlight[Customer.f]

```

Fail DoReach

```

HotelBooking.pc  $\neq$  endAcceptHotelBooking  $\wedge$ 
FlightBooking.pc  $\neq$  endAcceptHotelBooking  $\wedge$ 
Customer.pc  $\neq$  endAcceptHandFBooking

```

The goal is of the form “**TryReach** c **Fail DoReach** d ”. **TryReach** c requires a service that tries to reach condition c , in our case the condition “sell a travel offer to the customer”. During the execution of the service, a state may be reached from which it is not possible to reach c , e.g., since the flight is not available. When such a state is reached, the requirement **TryReach** c fails and the recovery condition **DoReach** d , in our case “book neither a room nor a flight” is considered.

The core part of the automated composition task is step 3. A formal definition of the problem solved by this task is given in Definition 5. Starting from a set of STSs modeling the existing web services and from a composition goal, a new STS is generated which implements the composed service. We see the process level composition problem as a planning problem, where the parallel STS $\Sigma_{\parallel}$ is the planning domain, ρ is the planning goal, and Σ_c is the generated plan that achieves the goal ρ . Notice that the planning problem is far trivial and cannot be reduced to a classical planning problem. We need planning techniques able to deal with planning under uncertainty, and more precisely with planning in nondeterministic domains and under partial observability. Indeed, it is clear from Section 3 that the planning domain is nondeterministic, i.e., actions executed in the same state can have different outcomes, and partially observable, i.e., only part of the domain information is available at run time, and, at planning time, the planner must therefore deal with the problem that the domain might be in a set of possible states. Moreover, the generated plan Σ_c is not simply a sequence of actions, must represent conditional and iterative courses of actions. Finally, in the case of *once-for-all composition*, we need to deal with extended goals, i.e., with goals that are not simply sets of desired states, but can express complex temporal and preference conditions, as discussed for Step 2. For all these reasons, we generate the STS Σ_c by using the “planning as model checking” technique [2, 5, 1], which has been shown to provide a practical solution to the problem of planning under uncertainty, and has been shown experimentally to scale up to large state spaces. A detailed description of how “planning as model checking” can be applied to solve the composition problem can be found in [14].

In step 4, the automatically generated STS Σ_c is translated into an executable web service. In the case of on-the-fly composition, the code is immediately executed. In the case of once-for-all composition, the code is deployed, and is ready to answer to users’ requests. In this paper, we translate the STS into an OWL-S process model. Examples of (excerpts from) the generated OWL-S process models are given in Figures 2 and 3 for the cases of on-the-fly and once-for-all compositions, respectively.

5 Conclusions and Related Work

There is a large amount of literature addressing the problem of automated composition of web services. However, most of the approaches address composition at the functional level (see, e.g. [12, 4]), and much less emphasis has been devoted to the problem of process-level composition. Different planning approaches have been proposed to ad-

dress the problem of on-the-fly composition, from HTNs [17] to regression planning based on extensions of PDDL, to STRIPS-like planning for composing services described in DAML-S [15]. However, none of these techniques addresses the problem of composing web services with conditional outputs, non-nominal outcomes, and with process models describing interaction protocols that include conditional and iterative steps. In [8, 11, 7], the authors propose an approach to the automated composition of web services based on a translation of DAML-S to situation calculus and Petri Nets. Also in these papers, however, the automated composition is limited to sequential composition of atomic services, and composition requirements are limited to reachability conditions.

As far as we know, the only approach that deals with process-level composition of semantic web services is described in our previous work [16]. In that paper, we proposed a technique for automated composition where the component web services were described as OWL-S composite processes and the generated composed services were emitted as a BPEL4WS programs. In this paper, we provide a substantial contribution with respect to [16], namely composed services are generated as OWL-S process models. In this way, we support the automated composition fully within a semantic web framework, we generate web services enriched with semantic annotations, and we allow the re-use of the generated services as input to the automated composition task. This opens the way to the incremental development of more and more complex web services, within a service oriented development process, where new executable web services are composed and then re-used to compose further services.

As a witness of the increasing interest in executable semantic web services, recent efforts within the semantic web community are addressing the problem of providing execution engines for semantic web services. For instance, the “Mindswap” OWL-S execution engine developed at the University of Maryland [10] can execute (a subset of) OWL-S process models, while there is significant on going work in providing an execution engine, called WSMX, for the WSMO language [6].

In the future, we will address the problem of associating finite ranges to the data types in the generation of the state transition systems from the process-level descriptions of web services. In particular, we intend to investigate techniques for deciding the right size of these ranges, so that the generated web service implements a general solution that can be adopted independently of the actual ranges. We also intend to investigate the so called “knowledge-level” techniques [13] for the composition of web services: these techniques prevent the necessity of fixing a finite range and permit the generation of a general solution. Our future plans also include the integration of the automated composition task with reasoning techniques for discovery and selection of web services, and the extension of the techniques within the WSMO framework [6]. A further important extension of the framework will address how to automatically generate semantic annotations for the generated web services. A possibility for OWL-S will be to exploit logical frameworks, e.g., based on description and dynamic logic, which will be able to derive preconditions and effects of composed services.

References

1. P. Bertoli, A. Cimatti, M. Pistore, and P. Traverso. A Framework for Planning with Extended Goals under Partial Observability. In *Proc. ICAPS'03*, 2003.
2. A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.
3. The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. In *Technical White paper (OWL-S version 1.0)*, 2003.
4. I. Constantinescu, B. Faltings, and W. Binder. Typed Based Service Composition. In *Proc. WWW'04*, 2004.
5. U. Dal Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *Proc. AAAI'02*, 2002.
6. The Web Service Modeling Framework. SDK WSMO working group - <http://www.wsmo.org/>.
7. S. McIlraith and R. Fadel. Planning with Complex Actions. In *Proc. NMR'02*, 2002.
8. S. McIlraith and S. Son. Adapting Golog for composition of semantic web Services. In *Proc. KR'02*, 2002.
9. S. McIlraith, S. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
10. Mindswap. Maryland Information and Network Dynamics lab Semantic Web Agents Projects - <http://www.mindswap.org/>.
11. S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW'02*, 2002.
12. M. Paolucci, K. Sycara, and T. Kawamura. Delivering Semantic Web Services. In *Proc. WWW'03*, 2002.
13. R. Petrick and F. Bacchus. A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In *Proc. AIPS'02*, 2002.
14. M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proc. ICAPS'05*, 2005.
15. M. Sheshagiri, M. desJardins, and T. Finin. A Planner for Composing Services Described in DAML-S. In *Proc. AAMAS'03*, 2003.
16. P. Traverso and M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. ISWC'04*, 2004.
17. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proc. ISWC'03*, 2003.

The OWL-S Editor – A Development Tool for Semantic Web Services

Daniel Elenius, Grit Denker, David Martin,
Fred Gilham, John Khouri, Shahin Sadaati,
and Rukman Senanayake*

SRI International, Menlo Park, California, USA
`firstname.lastname@sri.com`

Abstract. The power of Web Service (WS) technology lies in the fact that it establishes a common, vendor-neutral platform for integrating distributed computing applications, in intranets as well as the Internet at large. Semantic Web Services (SWSs) promise to provide solutions to the challenges associated with automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based systems. One of the barriers to a wider adoption of SWS technology is the lack of tools for creating SWS specifications. OWL-S is one of the major SWS description languages. This paper presents an OWL-S Editor, whose objective is to allow easy, intuitive OWL-S service development and to provide a variety of special-purpose capabilities to facilitate SWS design. The editor is implemented as a plugin to the Protégé OWL ontology editor, and is being developed as open-source software.

1 Introduction

Web Services (WS) were invented to bring a new level of integration to the computing industry and its networked communities. Ideally, service-based applications should be able to interoperate despite being developed in different programming languages, at different times, by different people, with designs based on different assumptions. Standard protocols for service interface descriptions (WSDL¹) and service invocation (SOAP²), coupled with a global data format (XML³), were introduced to turn this vision of the Service-Oriented Architecture [1] into reality.

* Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168 to SRI, and in part by Vinnova (grant no. 2002-00907) and The Swedish Research Council (grant no. 621-2003-2991).

¹ Web Service Definition Language, <http://www.w3.org/TR/wds1>

² Simple Object Access Protocol, <http://www.w3.org/TR/soap12-part0/>

³ Extensible Markup Language, <http://www.w3.org/XML>

Web Services have met with very strong initial success, mostly in the area of integration within, and (to a lesser extent) between, businesses. An increasing number of organizations are endorsing WS technology as a standardized infrastructure for interoperation of disparate software components within the organization, fulfillment of transactions between organizations, and sharing of corporate resources with customers and partners. However, this integration has been achieved only through costly efforts in manually programming and designing these WSs. Developers spend time searching for the right services and adding adapter components between incompatible services, and the resulting applications cannot adapt dynamically to changes in their environment.

Although second-generation WS specifications are under development, such as WS-CDL⁴ and BPEL4WS⁵, to enhance the usability, scope, and expressiveness of WSs, there is an increasing realization that technologies from the Semantic Web (SW) [2] can also make crucial contributions to WS frameworks. Semantic Web Services (SWSs) [3] take up on this idea, introducing ontologies to describe, on the one hand, the concepts in the services' domains (e.g., flights and hotels, tourism, e-business), and on the other hand, characteristics of the services themselves (e.g., control flow, data flow) and their relationships to the domain ontologies (via inputs and outputs, preconditions and effects, and so on). These semantically rich descriptions enable automated machine reasoning over service and domain descriptions, thus supporting automation of service discovery, composition, and execution, and reducing manual configuration and programming efforts. The three most prominent SWS specification approaches currently under development are OWL-S [4], WSMO⁶, and SWSL⁷.

The field of SWSs is still in an early stage, and adoption has been slow. A limiting factor has been the lack of tool support. The objective has been to enable machines to manipulate services, yet so far arduous human work has been necessary to create the semantic service descriptions. While tools to create and edit SW ontologies in general do exist [5], modeling SWSs requires additional functionality and developer support in order to be practically feasible.

Tools that make the SWS technology accessible to a broad audience with diverse needs are a crucial factor in the success of SWS technology. Tools are needed to facilitate tasks such as service definition and annotation, execution and monitoring, and service registration and discovery. The OWL-S Editor is aimed at providing a flexible, yet powerful editor for OWL-S service definitions. This paper describes the design and current functionality of the OWL-S Editor as well as its future directions.

The remainder of the paper is organized as follows. A brief introduction to OWL-S is presented in Section 2. Section 3 forms the main part of this paper,

⁴ Web Services Choreography Description Language, <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041012>

⁵ <http://www-128.ibm.com/developerworks/library/ws-bpel/>

⁶ <http://www.wsmo.org>

⁷ <http://www.daml.org/services/swsl/>

and outlines the main features of the tool. An overview of related work is given in Section 4. Future work in this area is discussed in Section 5, and Section 6 concludes with a brief summary.

2 OWL-S Overview

OWL-S is an ontology of service concepts. OWL-S organizes a service description into four conceptual areas: the *process model*, the *profile*, the *grounding*, and the *service*.

A process model describes how a service performs its tasks. It includes information about inputs, outputs (including a specification of the conditions under which various outputs will occur), preconditions (circumstances that must hold before a service can be used), and results (changes brought about by a service). The process model differentiates between composite, atomic, and simple processes. For a composite process, the process model shows how it breaks down into simpler component processes, and the flow of control and data between them (see Sections 3.3 and 3.4). Atomic processes are essentially “black boxes” of functionality, and simple processes are abstract process descriptions that can relate to other composite or atomic processes.

A profile provides a general description of a WS, intended to be published and shared to facilitate service discovery. Profiles can include both *functional* properties (inputs, outputs, preconditions, and results) and *nonfunctional* properties (service name, text description, contact information, service category, and additional service parameters). The functional properties are derived from the process model, but it is not necessary to include all the functional properties from the process model in a profile. A simplified view can be provided for service discovery, on the assumption that the service consumer would eventually look at the process model to achieve a full understanding of how the service works.

A grounding specifies how a service is invoked, by detailing how the atomic processes in a service’s process model map onto a concrete messaging protocol. OWL-S allows for different types of groundings to be used, but the only type developed to date is the WSDL grounding (see Section 3.5), which allows any WS with a WSDL definition to be marked up as a SWS using OWL-S.

A service simply binds the other parts together into a unit that can be published and invoked. It is important to understand that the different parts of a service can be reused and connected in various ways. For example, a service provider may connect its process model with several profiles in order to provide customized advertisements to different communities of service consumers. A different service provider, providing a similar service, may reuse the same process model, possibly as part of a larger composite process, and connect it to a different grounding. The relationships between service components are modeled using properties such as **presents** (Service-to-Profile), **describedBy** (Service-to-Process Model), and **supports** (Service-to-Grounding).

3 OWL-S Editor: Design and Features

There are two main tasks in the development of OWL-S services. The first task is to define the service's domain ontologies in terms of OWL classes, properties, and instances. The second task is to create an OWL-S description of the service, relating this description to the domain ontologies. An OWL-S service description consists of *instances* of OWL-S classes such as **Service**, **Process**, **Input**, and **Output**. In some cases, the OWL-S ontology is also *extended* to handle specific modelling situations.

In order to best facilitate these tasks, we built the OWL-S Editor on top of the Protégé OWL Ontology Editor [5]. Protégé allows editing of domain ontologies out-of-the-box. However, efficient development of services requires additional features. Our strategy has been to leverage the existing functionality of Protégé and to utilize Protégé's pluggable architecture to extend it where we judged it would be helpful for the SWS developer. The result is a SWS development environment where the domain ontologies are well integrated with the service descriptions.

The main user interface to the OWL-S Editor is a so-called *tab widget*. Figure 1 shows the OWL-S Editor tab, which provides service-specific design capabilities as described in the following sections.

Our design also makes it easy to extend the OWL-S ontologies. A common scenario is to create subclasses of the OWL-S **Profile** class, creating a *profile hierarchy* with profiles specific for different domains. Figure 2 shows an example of a custom profile in such a hierarchy.

In addition, building our tool on top of Protégé means that users can take advantage of the many other existing Protégé plugins, e.g. for querying and visualizing the Knowledge Base (KB), and to export the KB to different formats. These different plugins coexist gracefully, all working on the same KB (see Figure 1).

The icons in the toolbar on the top left of the OWL-S Editor tab provide parameter management, generating an OWL-S service from a WSDL specification, graphical overview, and additional options. In the following sections we discuss these and other features in more detail.

3.1 Managing the Top-Level Ontology

As explained in Section 2, a number of properties connect the different components of OWL-S services. It is very important to be able to get a good overview of these relationships when developing an OWL-S service. The OWL-S Editor tab widget provides a customized view for managing instances of the OWL-S subontologies. Along the left side of the OWL-S tab are four *instance panes* (see Figure 1), one each for services, profiles, processes, and groundings. Each pane lists all instances of the corresponding type. For the process instance pane we also use small icons next to the process names to distinguish the different types of processes (e.g., “a” for atomic and “c” for composite). An ontology containing multiple service descriptions would have several instances

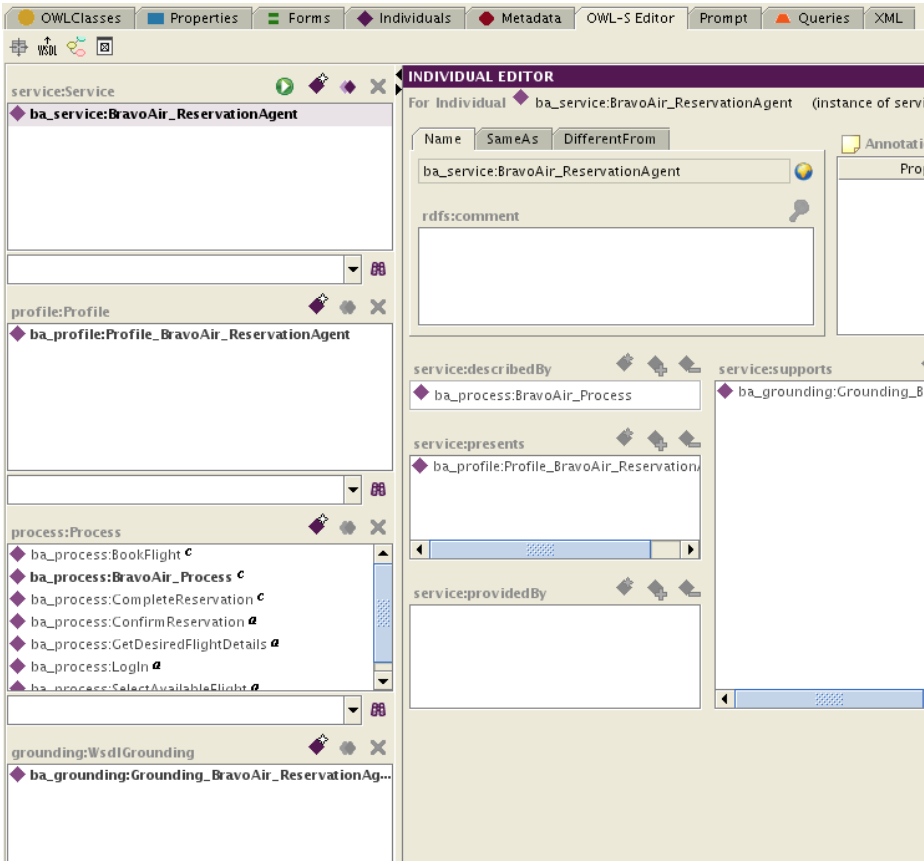


Fig. 1. The OWL-S Editor, a tab-widget plugin for Protégé is shown here next to the standard Protégé-OWL tabs to the left, and, to the right, other tab-widget plugins for ontology management, queries, and XML management

in each pane. To provide the user with an overview of how different service components fit together and which instances are related to one another (via **presents**, **describedBy**, and so on), the user can select an instance in one of the instance panes, and all instances that are directly related to the selected instance are emphasized in boldface in the other panes. In Figure 1 the service **ba_service:BravoAir_ReservationAgent** was highlighted in the service instance pane. As a result, its profile **ba_profile:Profile_BravoAir_ReservationAgent**, its top-level process **ba_process:BravoAir_Process**, and its grounding **ba_grounding:Grounding_BravoAir_ReservationAgent** are boldfaced in the other instance panes.

We have also implemented a graphical overview functionality. By selecting an instance and clicking a button in the toolbar, the user gets a graph view of the same information (see Figure 3).

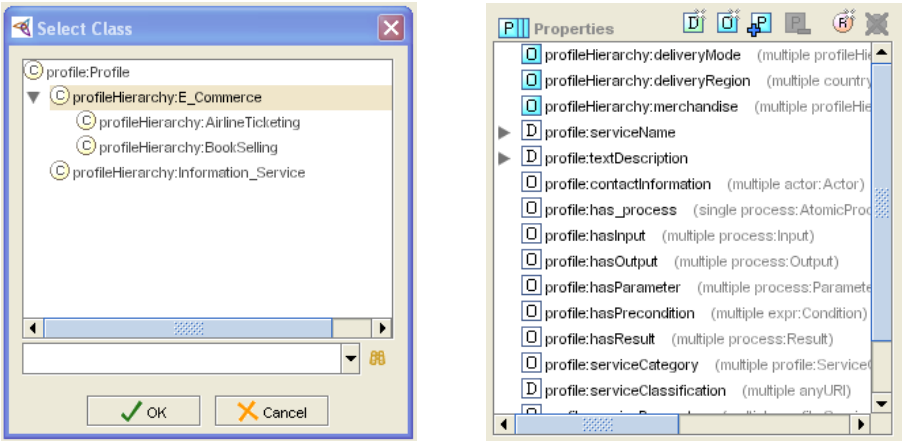


Fig. 2. A profile hierarchy for e-commerce that defines additional properties such as delivery mode and merchandise

When the user clicks an instance in one of the instance panes, the space to the right of the four panes changes to show a detailed *editing pane* for the selected instance. For example, if the user selects a profile instance, then the right window will show all properties of the profile. For some instances, such as processes, we have designed a layout on the right that provides a pictorial visualization of subprocesses, control constructs, and data flow (see Sections 3.3 and 3.4).

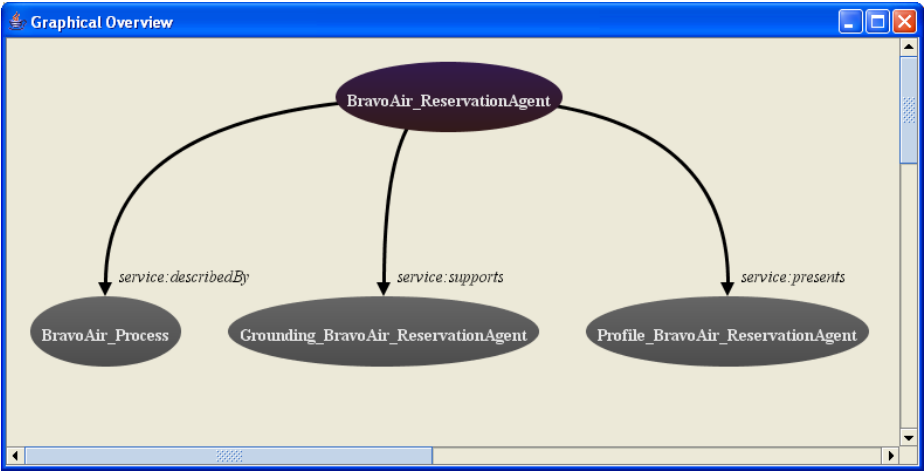


Fig. 3. Graphical Overview of the Bravo Air service

3.2 Managing Parameters

Inputs, outputs, preconditions, and results (IOPRs) are important parts of services. Both profiles and processes have a set of properties to relate them to their IOPRs: `hasInput`, `hasOutput`, `hasPrecondition`, and `hasResult`. As mentioned above, a profile usually includes a subset of the IOPRs of the process to which it is related. For this reason, it is often convenient to compare a profile side-by-side with the related process, and have them both in view when making decisions about the values of the IOPR properties. In addition, we sometimes want to relate the IOPRs of two profiles or processes (e.g., a composite process and an associated simple process, or two processes of different services).

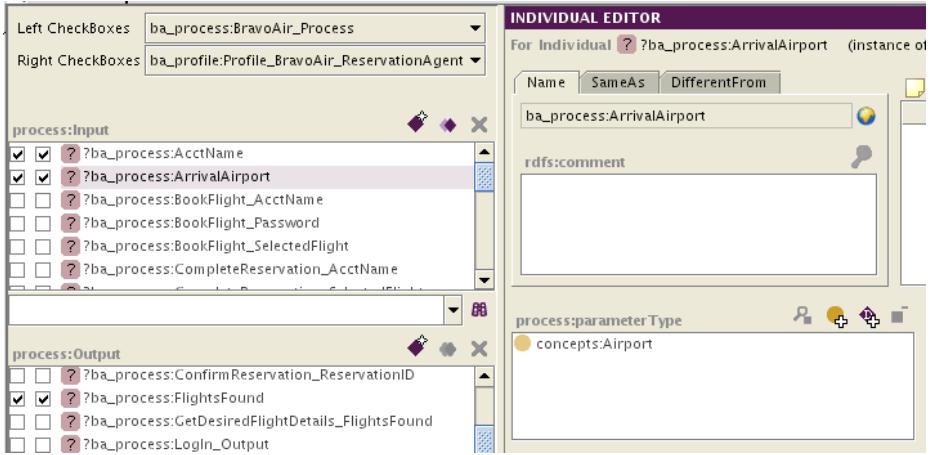


Fig. 4. IOPR Manager

To support efficient management of these IOPRs, we designed the *IOPR Manager*, which visualizes IOPR relationships in a very compact way (see Figure 4). Clicking a toolbar button brings up the IOPR Manager window, which is somewhat similar to the main tab widget of the OWL-S Editor. Like the tab widget, it provides four instance panes to the left, and an editing pane to the right (not shown here). The instance panes of the IOPR Manager show all the IOPRs in the KB, and allow the user to create and delete IOPRs. The user can also edit IOPR properties in the editing pane. In addition, two combo boxes at the top of the window allow users to select two processes and/or profiles that are to be compared with regard to their IOPRs. Associated with each combo box is a column of checkboxes, one for each IOPR. The user can simply check or uncheck these boxes to add or remove instances of the corresponding properties (`hasInput`, `hasOutput`, and so on).

As an example, if we select `ba_process:BravoAir_Process` and its profile `ba_profile:Profile_BravoAir_ReservationAgent` in the combo boxes, then the

checkboxes show that they both have the input `ba_process:ArrivalAirport`. In addition, the editing pane shows us that the `parameterType` of this input is the `Airport` class (which is imported from a domain ontology of flight-related concepts).

Groundings also refer to inputs and outputs. However, groundings do not refer to preconditions or effects, and the relationship with inputs and outputs is somewhat different from that of profiles and processes. For these reasons, we chose not to include the groundings in the IOPR Manager. Instead, we implemented separate support for editing groundings (see Section 3.5).

3.3 Control Flow

A powerful feature of OWL-S is the ability to model composite processes. A composite process is constructed from subprocesses that can in turn be composite, atomic, or simple. The control flow of a composite process is defined using control constructs, such as If-Then-Else, Sequence, and Repeat-Until. These constructs can be nested to an arbitrary depth.

These control flows are particularly difficult to generate by hand or in a plain ontology editor not designed for this task. The OWL-S editor visualizes these control flows graphically, in a style similar to UML Activity Diagrams, using boxes for subprocess invocation (called Performs in OWL-S), diamonds for conditional nodes (e.g., for If-Then-Else constructs), and arrows showing the flow of execution. Being able to view these “work flow” graphs was a high priority for us. OWL-S control flows have more structure than arbitrary flow charts or UML activity diagrams, however. Therefore, we do not allow users to directly “draw” the work flow. Instead, we take advantage of the fact that all OWL-S control flows are trees in the graph-theoretical sense. We let the user model the control flow in a GUI tree component, with full drag-and-drop support, whereas the corresponding work flow graph is updated to reflect any changes to this tree (see Figure 5).

The view in Figure 5 is the editing pane for composite processes. If a user selects a composite process in the process instance pane, the editing pane to the right has the shown layout. If the user does a right-click on the process graph view of a composite process, a menu will pop up, offering zooming, printing, and SVG exporting capabilities.

As a further note, the process modeling that forms a part of the service semantics has reaped interest outside of the area of SWSs (e.g. in [6]). The process modeling part of our tool can be used to create process descriptions not necessarily related to WSs. However, this is not the primary goal of the OWL-S Editor.

3.4 Data Flow

In addition to control flow, composite processes can specify their data flow. For example, we can state that a certain input of Process B should be taken from a certain output of Process A. The goal in OWL-S (and our tool) is to also be able

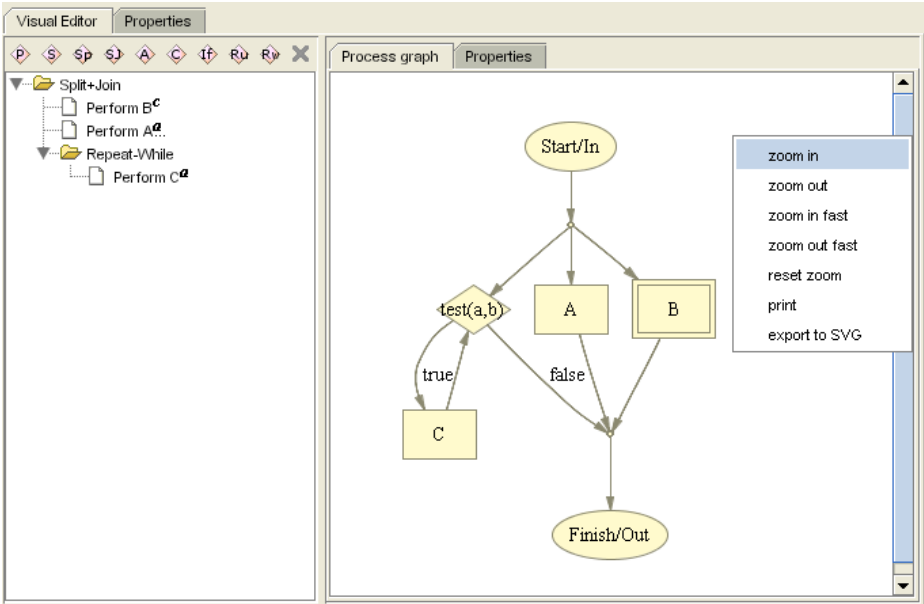


Fig. 5. A composite process, its tree structure shown to the left, and its graph representation to the right

to define more complex things, such as the input of Process B being the sum of the outputs of Processes A and C. The details of this remain to be worked out, but the simple one-to-one mappings should be sufficient for many applications.

Data flow is another area that is complicated to do by hand, but that can take great advantage of a graphical representation and specialized editing support. Both are supplied by the OWL-S Editor (see Figure 6).

Data flow definitions relate two parameters of different processes with each other. Either one associates a parameter of the parent process with a parameter of one of its component processes, or one relates two parameters of two component processes (atomic or complex) in the same parent process. If the user clicks on one of the Perform boxes in the process graph, a popup window (shown in the lower part of Figure 6) appears. This popup window shows the properties of that Perform, including any incoming data flow. Here, the user selects an input of the process (left part in the figure), and a source from which to take the value (right part). For the source, the user needs to first select the process (“From Perform”) and then a parameter in that process (“From Parameter”) to create a data flow declaration.

3.5 Grounding and WSDL Import

We have already mentioned that OWL-S descriptions can relate to WSDL files through groundings. OWL-S processes can relate to WSDL files in several ways

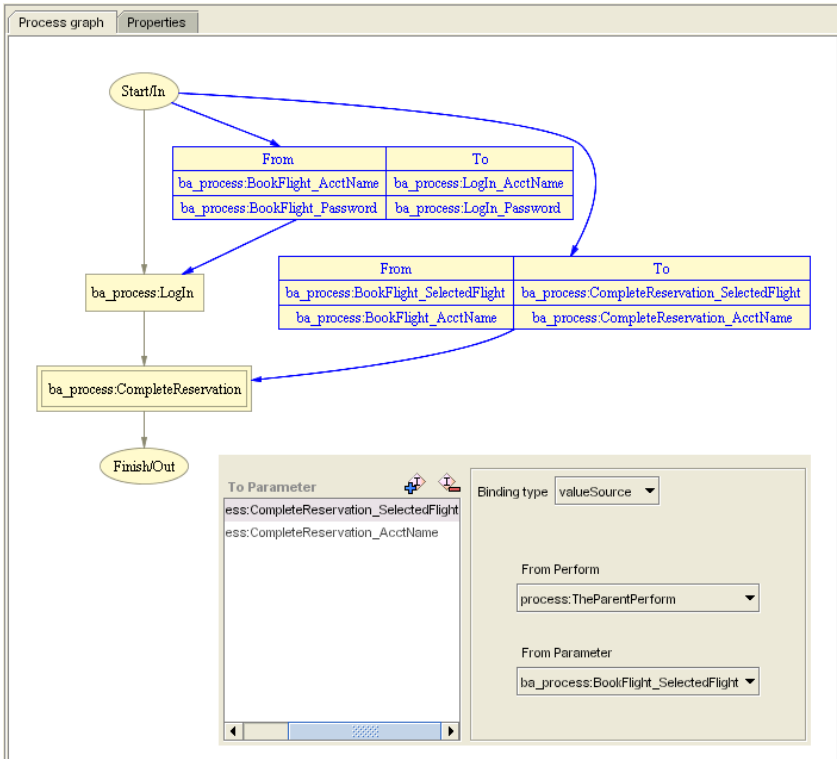


Fig. 6. Example of data flow between processes, and the popup window for editing data flow declarations

(see Figure 7), making it somewhat complicated to model this. In the simplest case, there is a one-to-one correspondence between an OWL-S input parameter and a message part of a WSDL input message as well as a one-to-one correspondence between a WSDL operation output message part and an OWL-S output parameter. These correspondences are defined in the grounding of a service through so-called *WsdMessageMaps*. In either of the two one-to-one correspondences, the WSDL service accepts serialized OWL, or the ontology operates on XSD[7] data types. Often, however, a transformation has to take place, in order to map between concepts in the ontology and complex XSD types on the WSDL side. We have added rudimentary support for this task in the OWL-S Editor, but complex mappings still have to be written manually. It is our goal to make it straightforward and easy to declare these mappings in the OWL-S Editor.

In many cases, it will be desirable to create a “skeletal” OWL-S description based on a preexisting WSDL file. Parts of the OWL-S description can be generated automatically based on the inputs and outputs defined in the WSDL file. To this end, we have integrated the WSDL2OWLS code, part of the OWL-S API

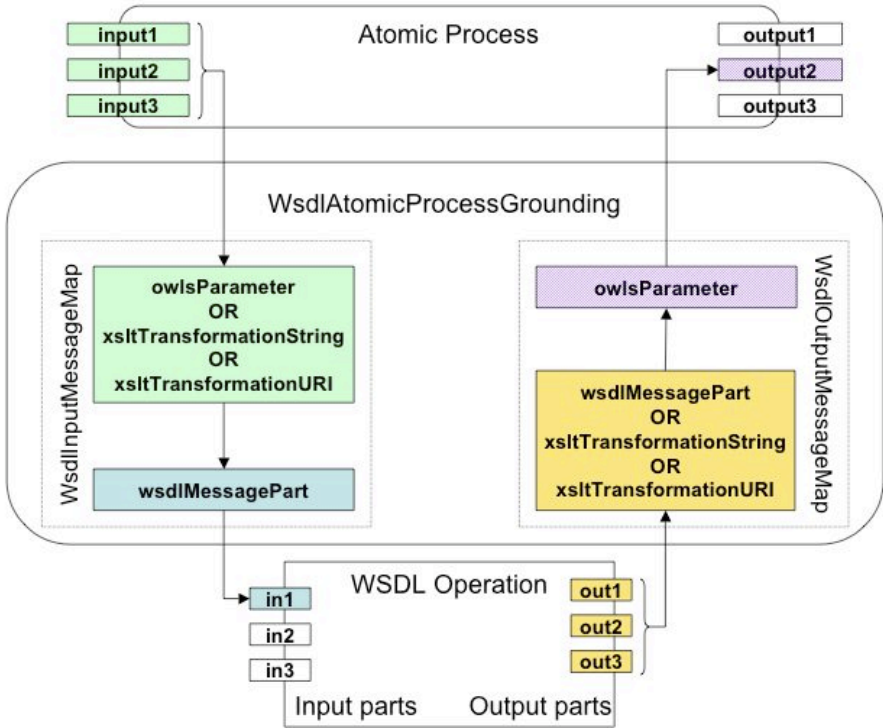


Fig. 7. Grounding: WSDL Message Maps

from Mindswap⁸, to the OWL-S Editor. This allows users to perform this type of OWL-S generation from a WSDL file by clicking one of the toolbar buttons (see Figure 1).

3.6 Execution

An exciting feature of the OWL-S Editor is the ability to actually *execute* services inside the editing environment. Selecting a Service instance and clicking the 'play' button (see Figure 1) will execute that service, provided that it has a WSDL grounding which is hooked up to a real web service. The user is presented with a window (see Figure 8) where he/she gets to choose the values of the input parameters (or create new instances for them) based on the parameter types defined in the service's process model. This functionality is work in progress, but we aim to support composite as well as atomic processes, and users will be able to take the results returned from services and add them into the Protégé knowledge base.

⁸ <http://www.mindswap.org/2004/owl-s/api/>

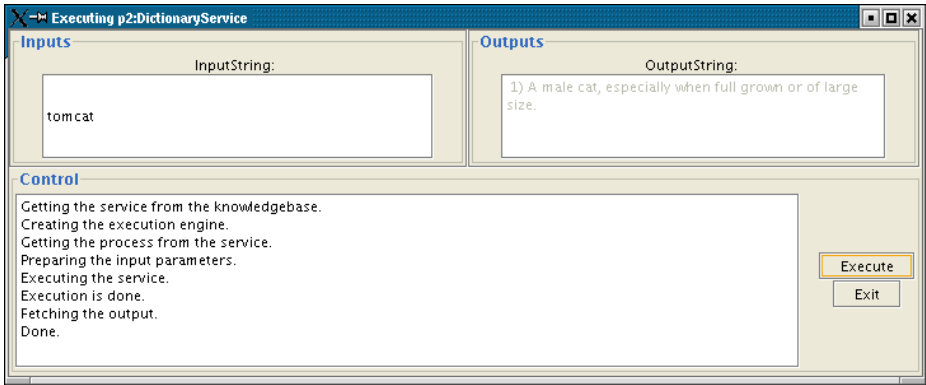


Fig. 8. Execution of a Semantic Web Service

4 Related Work

The OWL-S IDE project⁹ is also concerned with the development of OWL-S services. The OWL-S IDE is a plugin for Eclipse¹⁰, which attempts to integrate the semantic markup with the programming environment. Developers can write their Java code in Eclipse, and run a Java2OWLS tool to generate an OWL-S “skeleton” directly from the Java sources.

The idea of integrating SWSs more closely with the programming environment used to develop the service implementations is a good one. However, Eclipse does not support ontology editing, and there is no KB from which to choose the domain concepts to which the OWL-S files should relate. Furthermore, it will often be more useful to generate the semantic markup *before* the Java (or other) code, as the semantic descriptions can be seen as a higher level of abstraction of the programming modules. The OWL-S IDE does not provide any graphical visualization of services or processes.

There are plans to integrate Protege with Eclipse in the future, so perhaps we will have the best of both worlds—tight integration with the programming environment, as well as ontology editing and KB integration, all in the same IDE.

Another OWL-S Editor [8] has been developed at the University of Malta. It is a stand-alone program, providing WSDL import as well as a graphical editor and visualization for control flow and data flow. Not being integrated with an ontology editor, it shares some of the drawbacks of the OWL-S IDE, without gaining the advantage of programming-language integration.

ODE SWS is a tool for editing SWSs “at the knowledge level” [9], describing services following a Problem-Solving Methods (PSMs)[10] approach. OWL-S

⁹ <http://projects.semwebcentral.org/projects/owl-s-ide/>, formerly known as CODE

¹⁰ <http://www.eclipse.org>

plays a subordinate role in this environment, whereas the OWL-S descriptions are the main focus of our work.

IRS-3 [11] also follows the PSM approach, but lacks the graphical tools of ODE SWS or the OWL-S Editor, and does not support OWL-S, favoring WSMO instead.

To the best of our knowledge, none of the projects above have released their source code, whereas the source code for the OWL-S Editor has been available from the beginning, at <http://owlseditor.semwebcentral.org>.

5 Future Work

This tool represents early work in SWS design and development. In the following, we present some areas that we plan to work on in the future.

A limitation in Protégé is that it is not designed for concurrently working with multiple ontologies. However, it is often useful to be able to do so when working with SWSs. One often wants to edit service components spread across different subontologies, and the domain ontologies are normally separated from the service descriptions. Fortunately, the Protégé developers are working to implementing this functionality.

One aspect of OWL-S services not covered in this paper is the editing of preconditions and effects of processes, and conditions associated with control constructs such as If-Then-Else. In OWL-S, these are normally described in the SWRL language¹¹. Currently, we simply provide a text box where users can enter these SWRL expressions. However, we plan to provide more user-friendly editing capabilities. Protégé has recently been enhanced with native support for SWRL, including a SWRL expression-builder, which will serve as the basis of this work.

A feature not yet implemented is online search for services. A central idea in OWL-S is reusability. The separation of service descriptions into process models, profiles, and groundings, means that these components can be re-used in other services. An online search capability for service components inside the OWL-S Editor would greatly facilitate such reuse. Such a search facility could also be used to find entire services, to be included as parts of a composite process that the user is working on in the OWL-S Editor. Ideally, the user should be able to give detailed search criteria, and find a service that matches her current needs (e.g. to find a service with inputs matching the outputs of previous processes in a composite process model). Various approaches to online searching could be implemented, ranging from brute-force Google¹² search for `.owl` files, via semantic search engines such as Swoogle¹³, to service-specific systems such as semantically enhanced UDDI registries[12].

¹¹ <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>

¹² www.google.com

¹³ www.swoogle.org

We are also working on improvements to the graph overview presented in Section 3.1 to (1) show the entire “forest” of OWL-S instances simultaneously, and (2) allow users to change the relationships between the service components, i.e. add or remove instances of the properties **presents**, **describedBy**, etc.

We are also interested in the generation of OWL-S descriptions from BPXL4WS files. We are closely following work in this area¹⁴ for possible inclusions in our tool.

6 Concluding Remarks

We have argued in this paper that Semantic Web Services (SWSs) could enable radically improved integration of businesses and networked communities, by automating service discovery, composition, and execution. SWSs thus promise great potential gains, but uptake has so far been slow. Lack of tool support has been a limiting factor for adoption of SWS technology.

OWL-S represents an emerging standard for SWSs, providing the concepts necessary to create detailed service descriptions. This paper has introduced the OWL-S Editor, a development tool for OWL-S services. This tool allows engineering of all aspects of SWSs, providing specialized views and design features wherever deemed necessary. The tool is well integrated with the Protégé OWL ontology editing framework. This integration means that developers can load, edit, and create domain ontologies, and subsequently relate their services to domain concepts in an easy and intuitive way.

Among the main features are graphical editing and visualization of control flow and data flow; the ability to easily maintain a large number of services; functionality to manage the relationships between service components and parameters; and generation of “skeletal” OWL-S descriptions from WSDL files.

A number of desirable extensions to the OWL-S Editor, such as online searching capabilities, and an integrated execution environment for services, have been discussed. In addition, we also plan to investigate the SWS software development process as a whole. This could involve such things as best practices for developing SWSs, “design patterns”[13] for SWSs, and the relationships between OWL-S and other representations and methodologies such as UML[14], Model-Driven Architectures[15], and PSL[16].

In providing this tool to the community, our aim is to make it easier to understand the concepts of SWSs, and to create semantic descriptions of services. We believe that this can bring a fruitful cross-pollination between practice and theory. As more people start developing SWSs, important feedback on using the service ontologies in various projects, and on design and implementation aspects of SWSs, could benefit the knowledge in this field.

The OWL-S Editor is available for download in both binary and source formats on <http://owlseditor.semwebcentral.org>. We welcome all feedback on our mailings list.

¹⁴ <http://www.it.swin.edu.au/centres/cicec/bpel2owls.htm>

References

1. Erl, T.: *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, Upper Saddle River, NJ, USA (2004)
2. Berners-Lee, T., Hendler, J., Lassila, O.: *The Semantic Web*. Scientific American (2001)
3. McIlraith, S., Song, T., Zeng, H.: Semantic Web services. *IEEE Intelligent Systems*, Special Issue on the Semantic Web **16** (2001) 46–53
4. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing semantics to Web Services: The OWL-S approach. In: *Proc. First Intern. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, July 6-9, 2004, San Diego, California, USA. (2004) <http://www.daml.org/services/owl-s>.
5. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open development environment for semantic web applications. In: McIlraith, S., Plexousakis, D., van Harmelen, F., eds.: *Proc. 3rd Intern. Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, November 2004, Springer (2004) 229–243 LNCS 3298.
6. Schlenoff, C., Barbera, T., Washington, R.: Experiences in developing an intelligent ground vehicle (IGV) ontology in protégé. In: *Proceedings of the 7th International Protégé Conference*. (2004) <http://protege.stanford.edu/conference/2004/abstracts/Schlenoff.pdf>.
7. Fallside, D.C., (eds.), P.W.: *XML Schema part 0: Primer second edition* (2004) <http://www.w3.org/TR/xmlschema-0/>.
8. Scicluna, J., Abela, C., Montebello, M.: Visual modelling of OWL-S services. In: *Proceedings of the IADIS International Conference WWW/Internet*, Madrid, Spain, October 2004. (2004) <http://www.daml.org/services/owl-s/pub-archive/Visual-Modeling-of-OWL-S-%-Services.pdf>.
9. Gómez-Pérez, A., González-Cabero, R., Lama, M.: Development of semantic web services at the knowledge level. In: *European Conference on Web Services (ECOWS)*, Erfurt, Germany. (2004)
10. Fensel, D.: *Problem Solving Methods*. Springer-Verlag Telos (2000)
11. Domingue, J., Cabral, L., Hakimpour, F., Sell, D., Motta, E.: IRS-III: A platform and infrastructure for creating wsmo-based semantic web services. In: *Proceedings of the Workshop on WSMO Implementations (WIW 2004)* Frankfurt, Germany, September 29-30, 2004. (2004) *CEUR Workshop Proceedings*, ISSN 1613-0073.
12. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: *Proceedings of the 1st International Semantic Web Conference (ISWC2002)*. (2002)
13. Gamma, E.: *Design Patterns*. Addison-Wesley, Boston, MA, USA (1995)
14. Brooch, G.: *Object-Oriented Analysis with Design and Applications* (2nd ed). Addison-Wesley, Boston, MA, USA (1993)
15. Raistrick, C., Francis, P., Wright, J.: *Model Driven Architecture with Executable UML*. Cambridge University Press, Cambridge, UK (2004)
16. Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., Lee, J.: *The Process Specification Language (PSL): Overview and version 1.0 specification*. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD. (2000)

Temporal RDF

Claudio Gutierrez¹, Carlos Hurtado¹, and Alejandro Vaisman²

¹ Department of Computer Science,
Universidad de Chile

{cgutierr, churtado}@dcc.uchile.cl

² Department of Computer Science,
Universidad de Buenos Aires,

avaisman@dc.uba.ar

Abstract. The *Resource Description Framework (RDF)* is a metadata model and language recommended by the W3C. This paper presents a framework to incorporate temporal reasoning into RDF, yielding *temporal RDF graphs*. We present a semantics for temporal RDF graphs, a syntax to incorporate temporality into standard RDF graphs, an inference system for temporal RDF graphs, complexity bounds showing that entailment in temporal RDF graphs does not yield extra asymptotic complexity with respect to standard RDF graphs and sketch a temporal query language for RDF.

1 Introduction

The *Resource Description Framework (RDF)* [14] is a metadata model and language recommended by the W3C for building an infrastructure of machine-readable semantics for the data on the Web, a long-term vision known as *Semantic Web*. In the RDF model, the universe to be modeled is a set of *resources*, essentially anything that can have a *universal resource identifier*, URI. The language to describe them is a set of *properties*, technically binary predicates. Descriptions are *statements* very much in the subject-predicate-object structure. Both subject and object can be anonymous objects, known as *blank nodes*. In addition, the RDF specification includes a built-in vocabulary with a normative semantics (RDFS) [4]. This vocabulary deals with inheritance of classes and properties, as well as typing, among other features allowing the descriptions of concepts and relationships that can exist for a community of people and software agents, enabling knowledge sharing and reuse.

Although some studies exist about addressing changes in an ontology [15], or the need for temporal annotations on Web documents [22], little attention has deserved the problem of representing, updating and querying temporal information in RDF. Time is present in almost any Web application. Indeed, as pointed out by Abiteboul [1] the modeling of time is one of the key primitives needed in a query language for Web and semistructured data. Thus, there is a clear need of applying temporal database concepts to RDF to allow metadata navigation across time.

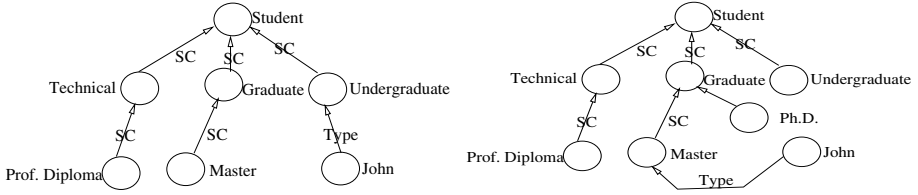


Fig. 1. Initial RDF graph (left) and after some changes (right)

Consider an RDF graph describing information about a university, as of its creation time, Figure 1 (left). Students were classified as technical, graduate or undergraduate, and the only graduate programs offered were at the level of ‘Master’ studies, like MBA or MSc; ‘Professional Diploma’ was the only program offered at the technical level. As the university evolved, a Ph.D program was created. Figure 1 (right) illustrates the new situation. Notice the dynamics of this example: students (e.g., John) can enroll in one program (e.g., Undergraduate), then shift to another one (e.g., Master), and so on. The figures show that the impact of disregarding the time dimension is twofold: on the one hand, when a change occurs, a new metadata document must be created (and the current document dropped). On the other hand, queries asking for past states of the metadata cannot be supported. For instance, we cannot ask for the programs offered at the time when the university was created.

1.1 Problem Statement: Introducing Time into RDF

Generally speaking, a temporal database is a repository of temporal information. Although temporal databases were initially studied for adding the time dimension to relational databases, as new data models emerged, temporal extensions to these models were also proposed (see Section 1.2). We next discuss the main issues that arise when extending RDF with temporal information.

Versioning Versus Time Labeling. There are two mechanisms for adding the time dimension to non-temporal RDF graphs: labeling and versioning (following the *timestamp* and *snapshot* models, respectively). The former consists in labeling the elements subject to changes (*i.e.* triples). The latter is based on maintaining a snapshot of each state of the graph. For instance, each time a triple changes, a new version of the RDF graph is created, and the past state is stored somewhere. Although both models are equivalent, versioning appears to be not suitable for queries of the form: “all time instants where Φ holds in the database”.

There are at least two temporal dimensions to consider when dealing with temporal databases: *valid* and *transaction* times. *Valid* time is the time when data is valid in the modeled world; *transaction* time is the time when data is actually stored in the database. The versioning approach captures transaction time, while labeling is mostly used when representing valid time. The approach we present in this paper supports both time dimensions.

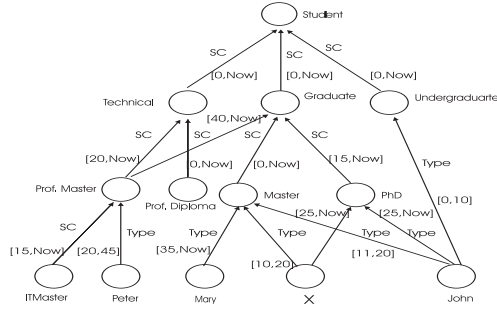


Fig. 2. A temporal RDF graph accounting for the evolution of the university ontology

In summary, we believe that for RDF data, labeling is better than versioning, because (a) it preserves the spirit of the distributed and extensible nature of RDF, and (b) in scenarios where changes are frequent and only affecting a few elements of the document, creating a new physical version of the graph each time an update occurs may lead to large overheads when processing temporal queries that span multiple versions.

Time Points Versus Time Intervals. We will work with the point-based temporal domain for defining our data model and query language, but we will encode time-points in intervals when possible, for the sake of clarity. We will consider time as a discrete, linearly ordered domain, as usual in virtually all temporal database applications. An ordered pair $[a, b]$ of time points, with $a \leq b$, denotes the closed interval from a to b . Figure 2 shows a temporal RDF graph for the university example above. The arcs in the graph are labeled with their interval of validity.¹ For example, the interval $[0, \text{Now}]$ says that the triple (technical, sc, student) is valid from the document’s creation time to the current time. Also, note that the figure shows that John was an Undergraduate student in the interval $[0, 10]$, and now he is a PhD student.

Vocabulary for Temporal Labeling. Temporal labeling can be implemented within the RDF specification, making use of a simple additional vocabulary, as Figure 3 shows. As we adopted the point-based, discrete and linearly ordered temporal domain, the left and right hand sides of Figure 3 are equivalent. We will use both representations indistinctly. Moreover, we define constructs that allow moving between intervals and time instants as follows: the instants depicted in Figure 3 (left) can be encoded in an interval as shown in Figure 3 (right). Both alternatives will be used in the query language.

¹ Note that the standard graph(ical) representation of an RDF graph is not the most faithful to convey the idea of statements (triples) being labeled by a temporal element. Technically, temporal labels should be attached to a whole subgraph $u \xrightarrow{p} v$, and not only to an arc.

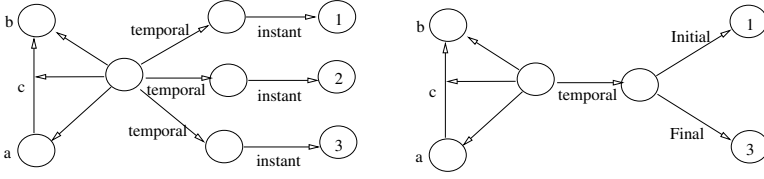


Fig. 3. Point-based labeling (left) and Interval-based labeling (right)

Temporal Entailment. An RDF graph can be regarded as a knowledge base from which new knowledge, *i.e.*, other graphs, may be entailed. Entailment in a temporal setting is a slightly more involved in the RDF case than in the standard database case. In principle, one may be tempted to define the semantics as in temporal relational databases, *i.e.*, defining the temporal database as the union of all of its snapshots. (A *snapshot* at time t of a temporal RDF graph G is the corresponding subgraph formed by triples labeled by and instant t .) Blank nodes impose some constraints to this naive approach. For example, each of the three snapshots of Figure 4 (right) entails the corresponding snapshots of Figure 4 (left). However, the whole graph of Figure 4 (left) cannot be entailed by the graph of Figure 4 (right). Indeed, the graph of Figure 4 (left) states that there is an anonymous object X in the triple (a, b, X) at times 3 and 4, which is not the case for the other graph.

Temporal Query Language. Regarding query languages in temporal databases, basically two choices for defining the temporal domains exist: the *point-based* and the *interval* based temporal domains, yielding different query languages [20, 3]. In the point-based approach, temporal variables in query languages refer to individual time instants, while in the interval-based domain, variables in the queries range over intervals, making queries more complicated and unnatural. Anyway, one can move easily between these two domains.

1.2 Related Work

The RDF model was introduced five years ago as a W3C recommendation [14]. Formal work in RDF includes the study of formal aspects of RDF data and query languages [10, 21], considering RDF features like the entailment, presence

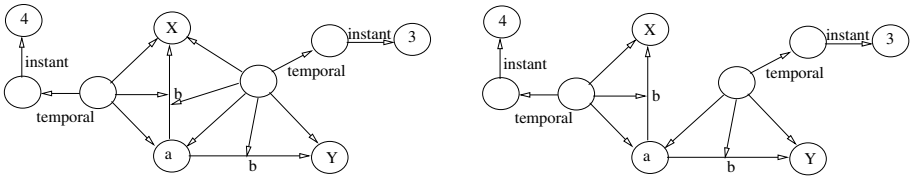


Fig. 4. Temporal entailment: for each t the corresponding snapshots at t are equivalent, but the graph on the left is not entailed by the graph on the right

of blank nodes, reification, premises in queries, and the RDFS vocabulary with predefined semantics. Several languages for querying RDF data have been proposed and implemented. Some of them in the lines of traditional database query languages (e.g. SQL, OQL), others based on logic and rule languages. Good surveys are [13, 16]. To the best of our knowledge, there is still no formal study of temporality issues in RDF graphs and RDF query languages.

Temporal database management has been extensively studied, including data models, mostly based on the relational model and query languages [19], leading to the TSQL2 language [18]. Beyond the relational model, managing historical semistructured data was first proposed by Chawathe *et al* [6], who extended the Object Exchange Model (OEM) with the ability to represent updates and to keep track of them by means of “deltas.” Later, Dyreson *et al* [7] allowed annotations on the edges of the database graph. In the XML world, Amagasa *et al* [2] introduced a temporal data model based on XPath for the first time. Dyreson [8] proposed an extension of XPath with support for transaction time by means of the addition of several temporal axes for specifying temporal directions, focusing on document versioning over the web in the absence of explicit time stamps. Chien *et al* [5] proposed update and versioning schemes for XML through an edit-based schema in which the most current version of the document is maintained, and reverse edit scripts allow moving backward in time. Gao *et al* [9] introduced τ XQuery, an extension to XQuery supporting valid time while maintaining the data model unchanged. Mendelzon *et al* [17] proposed a temporal model for XML, a temporal extension to XPath, and a novel indexing strategy for temporal XML documents. Like in our approach, they use labeling, and a point-based temporal domain and query language. Finally, Visser *et al* [22] proposed a temporal reasoning framework for the Semantic Web, which has been applied in BUSTER, an ontology-based prototype developed at the University of Bremen, supporting the so-called *concept@location in time* type of query.

1.3 Contributions

In this paper we present a framework to incorporate temporal reasoning into RDF, yielding *temporal RDF graphs*. In particular, we present the following contributions: (a) a semantics for temporal RDF graphs in terms of the semantics of non-temporal RDF and RDFS graphs; (b) a study of properties of temporal RDF graphs and the interplay between timestamp and snapshot semantics in temporal RDF graphs; (c) a syntax to incorporate this framework into standard RDF graphs, which includes a vocabulary and rules. The syntax uses the standard RDF vocabulary plus temporal labels; (d) a sound and complete inference system for temporal RDF graphs; (e) complexity bounds which show that entailment in temporal RDF graphs does not yield extra asymptotic time complexity with respect to standard RDF graphs; (f) a sketch for a temporal query language for RDF. For the sake of space, we do not include proofs in this version of the paper.

2 RDF Preliminaries

In this section we present a streamlined formalization of the RDF model following the W3C documents [14, 12, 4], along the lines of [10].

2.1 RDF Graphs

Assume there is an infinite set U (RDF URI references); an infinite set $B = \{N_j : j \in \mathbb{N}\}$ (Blank nodes); and an infinite set L (RDF literals). A triple $(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple*. In such a triple, v_1 is called the *subject*, v_2 the *predicate* and v_3 the *object*. We often denote by UBL the union of the sets U , B and L .

An *RDF graph* (just graph from now on) is a set of RDF triples. A *subgraph* is a subset of a graph. The *universe* of a graph G , $\text{universe}(G)$, is the set of elements of UBL that occur in the triples of G . The *vocabulary* of G is the set $\text{universe}(G) \cap (U \cup L)$. We will use letters $N, X, Y, \dots$ to denote blank nodes, and $a, b, c, \dots$ for URIs and literals. A graph is *ground* if it has no blank nodes. Graphically we represent RDF graphs as follows: each triple (a, b, c) is represented by the labeled graph $a \xrightarrow{b} c$. Note that the set of arc labels can have non-empty intersection with the set of node labels.

A *map* is a function $\mu : \text{UBL} \rightarrow \text{UBL}$ preserving URIs and literals, i.e., $\mu(u) = u$ and $\mu(l) = l$ for all $u \in U$ and $l \in L$. Given a graph G , we define $\mu(G)$ as the set of all $(\mu(s), \mu(p), \mu(o))$ such that $(s, p, o) \in G$. A map μ is *consistent* with G if $\mu(G)$ is an RDF graph, i.e., if s is the subject of a triple, then $\mu(s) \in UB$, and if p is the predicate of a triple, then $\mu(p) \in U$. In this case, we say that the graph $\mu(G)$ is an *instance* of the graph G . An instance of G is *proper* if $\mu(G)$ has fewer blank nodes than G . This means that either μ sends a blank node to an URI or a literal, or identifies two blank nodes of G . We will overload the meaning of map and speak of a *map* $\mu : G_1 \rightarrow G_2$ if there is a map μ such that $\mu(G_1)$ is a subgraph of G_2 .

Two graphs G_1, G_2 are *isomorphic*, denoted $G_1 \cong G_2$, if there are maps μ_1, μ_2 such that $\mu_1(G_1) = G_2$ and $\mu_2(G_2) = G_1$.

We define two operations on graphs. The *union* of G_1, G_2 , denoted $G_1 \cup G_2$, is the set theoretical union of their sets of triples. The *merge* of G_1, G_2 , denoted $G_1 + G_2$, is the union $G_1 \cup G'_2$, where G'_2 is an isomorphic copy of G_2 whose set of blank nodes is disjoint with that of G_1 . Note that $G_1 + G_2$ is unique up to isomorphism.

2.2 RDFS Vocabulary

There is a set of reserved words defined in the RDF vocabulary description language, RDF Schema [4], –just *rdfs-vocabulary* for us– that may be used to describe properties like attributes of resources (traditional attribute-value pairs), and also to represent relationships between resources. It defines classes and properties that may be used for describing groups of related resources and relation-

ships between resources.² *Classes* are sets of resources. Elements of a class are known as *instances* of that class. To state that a resource is an instance of a class, the property `rdf:type` may be used. The following are the most important classes (in brackets the name we will use in this paper) `rdfs:Resource` [**res**], `rdfs:Class` [**class**], `rdfs:Literal` [**literal**], `rdfs:Datatype` [**datatype**], `rdf:XMLLiteral` [**xmlLit**], `rdf:Property` [**property**]. *Properties* are binary relations between subject resources and object resources. The built-in properties are: `rdfs:range` [**range**], `rdfs:domain` [**dom**], `rdf:type` [**type**], `rdfs:subClassOf` [**sc**], `rdfs:subPropertyOf` [**sp**].

3 Temporal RDF Graphs

In this paper we extend RDF graphs by allowing temporal elements to label triples. A *temporal label* is a temporal element t labeling a triple (a, b, c) . For simplicity, without loss of generality, we will work with single intervals instead of temporal elements. In an RDF graph, given a triple (a, b, c) , the temporal element t represents the time period when the triple was valid, *i.e.* the *valid time* of the triple. At this time we do not deal with *transaction time*, which can be addressed in an analogous way.

3.1 Basic Definitions

In this section we define the notion of temporal RDF at a conceptual level.

Definition 1 (Temporal graph).

1. A temporal triple is an RDF triple with a temporal label (a natural number). We will use the notation $(a, b, c) : [t]$. The expression $(a, b, c) : [t_1, t_2]$ is a notation for $\{(a, b, c) : [t] \mid t_1 \leq t \leq t_2\}$.
2. A temporal graph is a set of temporal triples. A subgraph is a subset of the graph.

For a temporal graph G , define the snapshot at time t as the RDF graph

$$G(t) = \{(a, b, c) \mid (a, b, c) : [t] \in G\}$$

The underlying RDF graph of a temporal RDF graph G , denoted $u(G)$, is $\bigcup_t G(t)$, the union of the graphs $G(t)$.

For an RDF graph, define G^t as the temporalization of all its triples by a temporal mark t , that is, $G^t = \{(a, b, c) : [t] \mid (a, b, c) \in G\}$.

The above definitions give the following elementary consequences about the relationship between RDF graphs and temporal RDF graphs.

² We omit in this paper vocabulary intended to describe lists, collections, some variations on these, as well as vocabulary to help document and describe other functionalities for which there is no normative semantics. The complete vocabulary can be consulted in [4].

Lemma 1. *Let G be an RDF graph, and G' be a temporal RDF graph. Then:*

(1) $G^t(t) = G$; (2) $(G'(t))^t \subseteq G'$, and (3) $G' = \bigcup_t (G'(t))^t$.

Several issues on the definition of temporal RDF graph are in order:

- Recall we use a temporal model where an interval $[a, b]$ is of the form $[a, a + 1, \dots, b]$ for a given unit of time that we will assume to be universal in this paper. The natural way to approach this issue is to specify, together with the temporal mark, the unit of time it represents. All the results given here extend without difficulties to this setting.
- Temporal triples do not belong to the RDF syntax. In the next section we introduce an RDF-complying syntax for temporal triples, using a small temporal vocabulary.
- Source of a temporal statement: Due to the extensible nature of the RDF model, it is possible to include the source of a temporal statement (i.e. who is the author of the temporal statement), and other properties that apply. Although our model (see next section) allows this, we will not study the semantic consequences of this extra information in this paper, but rather stay in the classic setting of temporal models.

3.2 Semantics

In what follows, we present the semantics for the notion of entailment for temporal graphs based on the corresponding notion for RDF graphs.

Definition 2 (Temporal Entailment). *Let G_1, G_2 be RDF temporal graphs. Define*

- *For ground temporal RDF graphs G_1, G_2 define $G_1 \models_t G_2$ as $G_1(t) \models G_2(t)$ for each t ;*
- *For general graphs, $G_1 \models_t G_2$ iff there exist ground instances $\mu_1(G_1)$ and $\mu_2(G_2)$ such that $(\mu_1(G_1))(t) \models (\mu_2(G_2))(t)$ for each t .*

Note that the definition for ground graphs resembles classical temporal definitions:

Proposition 1. *Let G_1, G_2 be temporal graphs. Then, $G_1 \models_t G_2$ implies $G_1(t) \models G_2(t)$ for all t , and the converse is true for ground graphs.*

In fact, the problems for general graphs are introduced by blank nodes and the notion of entailment. For example, $G_1(t) \models G_2(t)$ for all t does not imply $G_1 \models_t G_2$ (see Figure 4). We have the following issues:

- A blank node represents the same (unnamed) resource throughout the time range, rather than a sequence of different resources. This makes the behavior of temporal marks in Temporal RDF different from the classical setting. Temporal marks here –contrary to temporal XML for example– are not only a relation among *fixed objects*, but also among *time-varying objects*, the blank nodes. See example in Figure 4.

- The notion of entailment for temporal RDF needs a basic arithmetic of intervals in order to combine the notion of temporality and deductive properties. For example if we have $(a, \mathbf{sc}, c) : [2, 3]$, $(c, \mathbf{sc}, d) : [2]$, then we should be able to derive $(a, \mathbf{sc}, d) : [2]$, but not $(a, \mathbf{sc}, d) : [3]$.

In the rest of this section, we show that the notions of closure, lean graph, core –fundamental to define notions of normalization of this data– can be extended without difficulty to the temporal setting. (Compare discussion in [10]).

The *closure* of a temporal graph G , denoted $\text{tcl}(G)$, is a maximal set of temporal triples G' over $\text{universe}(G)$ plus the RDF vocabulary such that G' contains G and is equivalent to it.

Proposition 2 (Entailment for Temporal graphs).

Let G, G_1, G_2 be temporal RDF graphs. Then

1. $\text{tcl}(G) = \bigcup_t (\text{cl}(G(t)))^t$;
2. $G_1 \models_t G_2$ if and only if $\text{tcl}(G_1) \models_t G_2$.

A temporal graph G is *lean* if and only if there is no proper temporal subgraph G' of G such that $G \models_t G'$. The *core* of G , $\text{core}(G)$ is a lean subgraph of G equivalent to it. With these notions, for a temporal RDF graph G we can define –as in the case of RDF graphs– a notion of *normal form*, denoted by $\text{nf}_t(G)$, as follows: $\text{nf}_t(G) = \text{core}_t(G')$ for a temporal closure G' of G .

The computational complexities of computing the core and testing whether a graph is lean, are asymptotically the same as the case of standard RDF graphs [10].

Proposition 3. Let G, G' be graphs.

1. The problem of deciding if G' is the closure of G is DP-complete.
2. The problem of deciding if G' is the normal form of G is DP-complete.

4 Syntax and Deductive System for Temporal Graphs

We present a deductive system for temporal RDF. It is based on a sound and complete set of rules given in [12], plus three rules capturing temporal issues.

4.1 RDF Syntax of Temporal Triples

Definition 3 (Temporal vocabulary). The temporal vocabulary is the following: *temporal* (abbreviated as *tpl*), *instant*, *interval*, *initial* and *final*, all of type *property*, and *now* of type *plain literal*. The range of *instant*, *initial* and *final* is the set of natural numbers.

We will use the following notation shortcuts: $\text{reif}(a, b, c, X)$: the set of triples $(X, \text{tsubj}, a), (X, \text{tpred}, b), (X, \text{tobj}, c)$ (a kind of “temporal reification” of (a, b, c)).³

³ We could have used here the standard reification vocabulary of RDF. We chose not to in order to stress the fact that the notions presented in this paper are independent of any view one may have about the concept of reification in RDF.

Definition 4 (Temporal triples and graphs). *Temporal triples are the following graphs using the temporal vocabulary.*

- $(a, b, c), \text{reif}(a, b, c, X), (X, \text{tpl}, Y), (Y, \text{instant}, n)$ where n is a natural number; we will summarize this as $(a, b, c) : [X, Y, n]$;
- $(a, b, c), \text{reif}(a, b, c, X), (X, \text{tpl}, Y), (Y, \text{interval}, Z), (Z, \text{initial}, I), (Z, \text{final}, F)$; where I, F are natural number; we will summarize this as $(a, b, c) : [X, Y, I, F]$;

A temporal graph will be defined as a merge of a set of temporal triples.

Because RDF is extensible, nothing prevents the use of the blank nodes included in the definition as target or source of other properties beyond the temporal vocabulary. We want to have a definition of temporal triple independent of the blank nodes occurring in the proposed syntactic definition of temporal triples, e.g., we want that $(a, b, c) : [X, Y, n]$ be essentially equivalent to $(a, b, c) : [n]$. Both previous issues are overcome in our syntax by adding certain rules, which regulate the temporal vocabulary.

4.2 Rules

The set of rules is arranged in four groups. Groups A, B, C, and D are intended to describe the classical RDFS semantics, and we follow the approach in [10]. We omit another group of rules that has to do with internal relationships of the RDF model itself and that we do not consider in this paper.

The novelty here is Group T (temporal rules), whose main objective is to be able to standardize the interval version and the instant version as well as help defining “absolute” temporal marks.

GROUP A (Existential) For a map $\mu : G' \rightarrow G, \quad \frac{G}{G'}$

GROUP B (Subproperty)

$$\frac{(a, \text{type}, \text{property})}{(a, \text{sp}, a)}, \quad \frac{(a, \text{sp}, b) \ (b, \text{sp}, c)}{(a, \text{sp}, c)}, \quad \frac{(a, \text{sp}, b) \ (x, a, y)}{(x, b, y)}$$

GROUP C (Subclass)

$$\frac{(a, \text{type}, \text{class})}{(a, \text{sc}, a)}, \quad \frac{(a, \text{sc}, b) \ (b, \text{sc}, c)}{(a, \text{sc}, c)}, \quad \frac{(a, \text{sc}, b) \ (x, \text{type}, a)}{(x, \text{type}, b)}$$

GROUP D (Typing)

$$\frac{(a, \text{dom}, c) \ (x, a, y)}{(x, \text{type}, c)} \quad \frac{(a, \text{range}, d) \ (x, a, y)}{(y, \text{type}, d)}$$

GROUP T (Temporal)

$$(i2t) \frac{\{(X, \text{tpl}, Y), (Y, \text{instant}, n) : n \in [t_1, t_2]\}}{(X, \text{tpl}, Y), (Y, \text{int}, Z), (Z, \text{initial}, t_1), (Z, \text{final}, t_2)}$$

$$(t2i) \frac{(X, \text{tpl}, Y), (Y, \text{int}, Z), (Z, \text{initial}, t_1), (Z, \text{final}, t_2)}{\{(X, \text{tpl}, Y), (Y, \text{instant}, n)\}}, \quad n \in [t_1, t_2]$$

$$(abs) \frac{(a, b, c) : [X_1, Y_1, n_1], (a, b, c) : [X_2, Y_2, n_2]}{(a, b, c) : [U, V, n_1], (a, b, c) : [U, V, n_2]}, \quad U, V \text{ fresh}$$

Rules (i2t) (interval to instants) and (t2i) are needed to standardize the interval version and the instant version, by making them equivalent. Rule (abs) essentially says that marks (instants) can be collected in a single node. This permits to concentrate on temporal marks independent of other contexts in which the variables involving temporal vocabulary are immersed.

The definition behaves well in the sense of the following lemma.

- Lemma 2.** 1. *There exist blanks X, Y such that $G \models_t (a, b, c) : [X, Y, t_1, t_2]$ if and only if there exist blanks U, V such that $\forall j$ with $t_1 \leq j \leq t_2$ $G \models_t (a, b, c) : [U, V, t_j]$*
2. *There exist blanks X_1, Y_1, X_2, Y_2 such that $G \models_t (a, b, c) : [X_1, Y_1, t_1]$ and $G \models_t (a, b, c) : [X_2, Y_2, t_2]$ if and only if there exist blanks U, V such that $G \models_t (a, b, c) : [U, V, t_1]$ and $G \models_t (a, b, c) : [U, V, t_2]$.*

For a temporal RDF graph G , define G^* as the RDF graph $\{(a, b, c) : [X_t, Y_t, t] \mid (a, b, c) : [t] \in G\}$, where X_t, Y_t are free blank variables, different for each t . Conversely, for each RDF graph G with temporal vocabulary, define G_* as the temporal graph defined as $\{(a, b, c) : [t] \mid \exists X \exists Y (a, b, c) : [X, Y, t] \in G\}$.

- Theorem 1.** 1. *Let G_1, G_2 be temporal RDF graphs. Then $G_1 \models_t G_2$ implies $G_1^* \models G_2^*$.*
2. *Let G_1, G_2 be RDF graphs with temporal vocabulary. Then $G_1 \models G_2$ implies $(G_1)_* \models_t (G_2)_*$.*
3. *Let G be a temporal RDF graph, and G' an RDF graph with temporal vocabulary. Then $(G^*)_* = G$ and $G' \models (G'_*)^*$.*

Now we can show that the syntax introduced captures the semantics of temporal RDF. The following deductive system based on the rules presented, is sound and complete for entailment of RDF graphs with rdts vocabulary.

Definition 5. *Let G be a graph. For each rule $r : \frac{A}{B}$ above, define $G \vdash_r G \cup \mu(B)$ iff there is a map $\mu : A \rightarrow G$. Also define $G \vdash_s G'$ if and only if G' is a subgraph of G .*

Define $G \vdash G'$ if there is a finite sequence of graphs $G_1, \dots, G_n$ such that (1) $G = G_1$; (2) $G' = G_n$; and (3) for each i , either, $G_i \vdash_r G_{i+1}$ for some r , or $G_i \vdash_s G_{i+1}$.

The following theorem shows that one can give a syntactic characterization over RDF graphs with temporal vocabulary for entailment of temporal RDF graphs:

- Theorem 2.** *For any pair of temporal RDF graphs G_1, G_2 :*
 $G_1 \models_t G_2$ *if and only if* $G_1^* \vdash G_2^*$.

Note that that we cannot establish the theorem in its complete generality, namely, to prove that for RDF graphs G_1, G_2 with temporal vocabulary, $G_1 \vdash G_2$ if and only if $(G_1)_* \models_t (G_2)_*$. (Both graphs in Figure 4 have identical $(\)_*$ -images but are not $\vdash$ -equivalent.)

The previous theorem permits to concentrate for the following sections in temporal RDF (instead of diving into syntactic issues).

5 Query Language

In this section we present query language for temporal RDF graphs, along with its semantics. We also present a brief study of the complexity of query processing.

5.1 The Query Language by Example

We will give the flavor of the query language using our running example, the database of Figure 2. Let us begin with a simple query: “Find students who have taken a Master course between year 2000 and now and return them qualified by 21-century-student”. This query can be expressed as:

$$(\text{?X, type, 21-century-student}) \leftarrow \\ (\text{?X, takes, ?C}) : [\text{?T}], (\text{?C, type, Master}) : [\text{?T}], 2000 \leq \text{?T}, \text{?T} \leq \text{Now}.$$

This example query illustrates the need of a built-in arithmetic language in order to reason about time and intervals. Another important observation is that temporal queries may output non-temporal RDF graph, as the previous query does.

For the query asking for a snapshot of the graph at t_1 , we have:

$$(\text{?X, ?Y, ?Z}) \leftarrow (\text{?X, ?Y, ?Y}) : [t_1].$$

Now consider the query “Students taking Ph.D courses together, and the time instants when this occurred.” For simplicity we expressed this as a point-based query. The translation of the result into intervals is straightforward.

$$(\text{?X, together, ?Y})[\text{?T}] \leftarrow (\text{?X, type, Ph.D}) : [\text{?T}], (\text{?Y, type, Ph.D}) : [\text{?T}].$$

Next, we give examples of queries that use temporal triples with intervals. The query “Time intervals when the IT Master was offered” can be expressed as follows:

$$(\text{X, interval, Y}), (\text{Y, initial, } T_i), (\text{Y, final, } T_f) \leftarrow \\ (\text{IT Master, sc, Prof.Master}) : [T_i, T_f].$$

Observe that the previous query returns a set of intervals. In order to retrieve maximal intervals we need a more subtle query, since their computation do not

follow from the temporal rules. For the query “Compute the maximal interval when the triple (a, b, c) holds”, we need aggregate operators MAX and MIN.

$$(a, b, c) : [?T_1, ?T_2] \leftarrow (a, b, c) : [?T_i, ?T_f], ?T_1 = \text{MIN}(?T_i), ?T_2 = \text{MAX}(?T_f)$$

For a query asking for “Students applying for jobs at time t after finishing their Ph.D. program in no more than 4 years”, we have:

$$(?X, \text{apply}, \text{job}) \leftarrow (X, \text{type}, \text{Ph.D}) : \|t_i, t_f\|, t_f - t_i < 4, t_f < t.$$

Here, the notation $\|t_i, t_f\|$ stands that t_i and t_f match with the maximal interval for the corresponding triple computed with the query given above.

5.2 Semantics and Complexity

Let V be a set of variables (disjoint from UBLT). Individual variables will be denoted $?X, ?Y, ?Z$, etc. There is also a set of temporal variables $V_t \subset V$.

The query language we define is analogous to the one presented by Gutierrez et al. [11]. A query is a *temporal tableau*, which is a pair $(H, B \cup A)$, where H and B are temporal RDF graphs with some elements of UBL replaced by variables in V , and with some elements of T replaced with variables in V_t , B has no blank nodes and all the variables in H occur also in B . The set A has the usual arithmetic built-in predicates such as $<, >, =, .$ over elements in V_t and T .

We adopt the usual notion of *safe rule* from Datalog to prevent operations on infinite predicates. A rule is *safe* if all its variables are limited. A variable is *limited* if one of the following hold: a variable appears as an argument in a non-built-in predicate of the body; the variable X appears in a subgoal $X = t$ (or $t = X$), where t is a constant in T ; or the variable X appears in a subgoal $X = Y$ (or $Y = X$), where Y is limited.

The semantics is the usual in these cases. Given a temporal tableau $(H, B \cup A)$ and a temporal RDF graph G , for each matching of the graph pattern B in G , pick up the values of the variables and check whether they satisfy the built-in predicates in A . If this is the case, construct a pre-answer, which is the graph resulting by substituting the values of the variables in the head. Finally, the answer of the query is the union of all pre-answers.

We end this section by showing that the additional time dimension in our model does not play any relevant role in the complexity of query answering, that is, the query language preserves the tractability of answers. In order to do this, we consider the simpler problem of testing emptiness of the query answer set in the following forms: (1) Query complexity version: For a fixed database D , given a query q , is $q(D)$ non-empty? (2) Data complexity version: For a fixed query q , given a database D , is $q(D)$ non-empty?

Theorem 3. *The evaluation problem is NP-complete for the query complexity version, and polynomial for the data complexity version.*

The previous result shows that the temporal labeling over the triples does not introduce any complexity overhead. This is consistent with previous works

in temporal databases. As Toman [20] showed, a point-based temporal query language has the same properties than a first order query language, in spite of the temporal variable.

6 Conclusions

We have proposed an RDF vocabulary to assert the times when triples are valid in RDF graphs. This allows an explicit treatment of time inside RDF. We have also offered a complete and sound inference procedure for temporal RDF graphs, and a query language for them. Our framework allows to browse, query, and reason across different versions of RDF graphs.

There are several aspects left for future work. Among the most important are the definition of a built-in arithmetic, aggregate functions, and a unified semantic for the two classes of RDF answers –temporal and plain– which would allow closeness and full query composition in a temporal query language for RDF. Probably one of the challenging issues open is the handling of anonymous times. For example, one may want to say that a triple holds in a particular time inside an interval, but do not know the exact valid time of the triple. Anonymous times may help in the specification of triples without temporal labels, which is a form to specify incomplete temporal information.

Acknowledgments. This research was supported by Millennium Nucleus, Center for Web Research (P01-029-F), Mideplan, Chile. C. Gutierrez and C. Hurtado were supported by FONDECYT 1030810.

References

1. S. Abiteboul. Querying Semi-Structured Data. *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*. Delphi, Greece, 1997.
2. T. Amagasa, M. Yoshikawa, S. Uemura, *A Temporal Data Model for XML Documents*, Proceedings of DEXA Conference, 2000, 334-344.
3. M. Bölen98, R. Busatto, C.S. Jensen *Point- Versus Interval-based Temporal Data Models*, Proceedings of IEEE/ICDE, 1998.
4. Dan Brickley, R.V. Guha Eds., *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Working Draft 23 January 2003.
5. S. Chien, V. Tsotras, C. Zaniolo, *Efficient Management of Multiversion Documents by Object Referencing*, Proceedings of the 27th International Conference on Very Large Data Bases, 2002, Rome, Italy, 291-300.
6. S. Chawathe, S. Abiteboul, J. Widom, *Managing Historical Semistructured Data*, Theory and Practice of Object Systems, Vol 5(3), 1999, 143-162.
7. C.E. Dyreson, M.H. Bolen, C.S. Jensen, *Capturing and Querying Multiple Aspects of Semistructured Data*, Proceedings of the 25th VLDB Conference, 1999, 290-301.
8. C.E. Dyreson, *Observing Transaction-time Semantics with TTXPath*, Proceedings of WISE 2001, 2001, 193-202.
9. C. Gao, R. Snodgrass, *Temporal Slicing in the Evaluation of XML Queries*, Proc. 29th Int. Conference on Very Large Data Bases, 2003, 632-643, Berlin, Germany.

10. C. Gutierrez, C. Hurtado, A.O. Mendelzon, *Formal aspects of querying RDF databases*, Proc. SWDB 2003, 293-307.
11. C. Gutierrez, C. Hurtado, A.O. Mendelzon, *Foundations of Semantic Web Databases*, 23rd. Symp. on Principles of Database Systems, PODS 2004, 95-106.
12. Patrick Hayes Ed., *RDF Semantics, W3C Working Draft, 1 October 2003*
13. P. Haase HAASE, J. Broekstra, A. Eberhart, R. Volz. *A comparison of RDF Query Languages*. International Semantic Web Conference, 2004.
14. O. Lassila, R. Swick Eds., *Resource description framework (RDF) model and syntax specification*, Working draft, W3C, 1998.
15. A. Maedche, B. Motik, L. Stojanovic, R. Studer, R. Volz *Establishing the semantic web 11: An infrastructure for searching, reusing, and evolving distributed ontologies*, Proc. of the 12th. Int. Conference on World Wide Web, 2003, 439-448.
16. A. Magkanaraki et al. *Ontology Storage and Querying*, Technical Report No. 308, April 2002, Foundation for Research and Technology Hellas, Institute of Computer Science, Information System Laboratory.
17. A.O. Mendelzon, F. Rizzolo, A. Vaisman, *Indexing Temporal XML*, Proc. 30th Int. Conference on Very Large Data Bases, Toronto, Canada, 2004, 216-227.
18. R. Snodgrass, *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.
19. A. Tansel, J. Clifford, S. Gadia Eds., *Temporal Databases: Theory, Design and Implementation*, Benjamin/Cummings, 1993.
20. D. Toman, *Point vs. Interval-based Query Languages for Temporal Databases*, 15 th. Symposium on Principles of Database Systems, PODS 1996, 58-67.
21. G. Yang, M. Kifer, *On the Semantics of Anonymous Identity and Reification* Proc. First International Conference on Ontologies, Databases and Applications of Semantics (ODBASE), 2002, 1047-1066.
22. Ubbo Visser, *Intelligent Information Integration for the Semantic Web*, Lecture Notes in Artificial Intelligence Volume 3159, Springer-Verlag, 2004.

Multilingual RDF and OWL

Jeremy J. Carroll¹ and Addison Phillips²

¹ HP Labs, Bristol, UK

`jjc@hpl.hp.com`

² Quest Software, Irvine, CA, US

`addison.phillips@quest.com`

(formerly with webMethods)

Abstract. RDF uses the RFC3066 standard for language tags for literals in natural languages. The revision RFC3066bis includes productive use of language, country and script codes. These form an implicit ontology of natural languages for marking-up texts. Relating each language tag with classes of appropriately tagged literals allows this implicit ontology to be made explicit as an ontology in OWL in which every class in the ontology is a datarange. The treatment extends to XML Literals, which may have multiple embedded language tags. Further features of RFC3066bis such as the relationship with deprecated codes, language ranges and language tag fallback can be expressed in OWL. A small change to the RDF model theory is suggested to permit access to the language tag in the formal semantics, giving this ontology a precise formal meaning. Illustrative use cases refer to use of English, Japanese, Chinese and Klingon texts.

1 Introduction

RDF [1], the foundation of the Semantic Web standards, has some provision for the use of natural language text from multiple languages, distinguished by use of language tags. This is a minimal requirement, made explicit in [2], for a knowledge representation language which is intended for interoperable use on a World Wide scale. This provision depends on XML, which in turn depends on “*RFC 1766 or its successors*”: the IETF is in the process of updating that track with RFC 3066bis [3], [4]. Limitations with the support for multiple languages in the Semantic Web are discussed in [5].

RFC 3066bis contains the significant advance of a generative ontology of language identification, and this paper explores the natural step of expressing that ontology using the Web Ontology Language (OWL). In addition, RFC 3066bis (and the earlier versions) connect various registries etc. Correct use depends on some detailed expert knowledge. We show how that expert knowledge can be expressed within OWL, making it easier for non-experts to correctly formulate OWL expressions for text with specific linguistic properties.

We use three example use cases throughout the paper, and demonstrate how the advances we propose help construct better Semantic Web applications, in which linguistic knowledge is captured in OWL, rather than application code.

This paper introduces a few properties and a large number of classes, using a variety of namespaces. We use the following namespace prefixes: `rdflg:` `rdfl-dflt:` `core-lang:` `xmllit-all-lang:` `xmllit-some-lang:` `xmllit-main-lang:` `presentable-lang:` `lang-range:`, but omit their binding to mutually distinct URIs. Such a binding is assumed.

2 Three Use Cases

2.1 Appropriate Display of Labels

This use case is described in the OWL Requirements [2]: a Semantic Web application has data to be displayed to an end user. Many of the resources in the knowledge base are to be displayed using one of the values of `rdfs:label`, selected to make a good match between the linguistic capabilities of the end-user and the language tag associated with that particular text string. A simplified example of such labels taken from the OWL Test Cases [8] is:

```
<owl:Class rdf:ID="ShakespearePlay">
  <rdfs:label xml:lang="it">Opere di Shakespeare</rdfs:label>
  <rdfs:label rdf:parseType="Literal"><span
    xml:lang="ja">シェイクスピアの<ruby>
      <rbc><rb>演</rb><rb>劇</rb></rbc>
      <rtc><rt>えん</rt><rt>げき</rt></rtc>
    </ruby></span></rdfs:label>
</owl:Class>
```

We consider a specific end user: Brian who is a mother tongue English speaker, with a good knowledge of Japanese, can read Kanji, and hence can make some sense of any language written in traditional Chinese characters; he still remembers some of his schoolboy French.

2.2 Finding all Klingon Text in a Knowledge Base

A Star trek fan wishes to search an RDF knowledge base for all the Klingon text in it, and then to explore the knowledge base from these resources.

2.3 Multilingual Knowledge Base Construction

An open source Semantic Web knowledge base is developed. The project started in the US, and all the natural language text strings in it have been tagged as "en-US" (following RDF Concepts [1] and RFC3066bis [3]). Other plain literals, with text that is not intended as natural language are marked up with the empty language tag. Gradually groups of Chinese developers (some from the mainland and other groups from Taiwan) become involved. Their typical interest is in using some subset of the knowledge base, possibly with some additional axioms. Moreover, each group has a

specific application in mind, involving specific queries which return literal values for end-user presentation. Also, depending on the intended users of the application, the presented text must be available in English or traditional Chinese or simplified Chinese or some combination. Clearly, some of the developers will need to add traditional Chinese literals corresponding to the original US English literals; others will need to add simplified Chinese; but precisely when it is *necessary* to translate which literals can only really be determined by asking an OWL reasoner the relevant queries and comparing the results for the various natural languages.

3 Language and Text in the Semantic Web Recommendations

RDF and OWL use literal nodes for natural language text. Literal nodes are either plain literals or typed literals.

3.1 Plain Literals in RDF

A plain literal is a Unicode string paired with an optional language tag [1]. For natural language text, the tag should be used in accordance with RFC 3066 (or its successors). A plain literal without a tag is simply a string and, like data of type `xsd:string`, it is not appropriate for natural language text, which may mean different things in different languages. There is no support within RDF or OWL semantics for language tags, other than the ability to distinguish literals that differ in language tag (case insensitively).

3.2 XML Literals in RDF

For some natural language text it is beneficial to use additional markup, for example for indicating bi-directional text or Ruby [9] markup (seen in the earlier Japanese example). These are typed literals and hence do not have an explicit language tag, however language information may be embedded within the literal using `xml:lang`. However, there may be more than one such `xml:lang`, each of which will have different parts of the XML literal in its scope; or even if there is exactly one `xml:lang`, there may be some text that is outside its scope.

3.3 OWL DataRanges

OWL does provide support for describing classes consisting entirely of literals. These are classes with type `owl:DataRange`. A significant difference from RDF datatypes is that plain literals as well as typed literals may belong to a datarange.

There is only explicit support for finite sets of literals, and no encouragement to use class expressions. In OWL Full, however, it is possible to construct class expressions from dataranges using `owl:unionOf`, `owl:intersectionOf` and `owl:complementOf` and dataranges can be infinite and can be named. We will make extensive use of these capabilities.

4 Language Tags and Their Generative Capacity

RFC 3066bis [3] is (currently) an Internet-Draft that expands and redefines the language tags used by `xml:lang` and other applications. The basic goal of a language tag is to identify the natural language of content in a machine accessible manner and the design of RFC 3066bis improves this capability.

The structure of all RFC 1766 and successor language tags is a series of subtags with an assigned meaning for each type of subtag. The combination of subtags forms a very rough ontology that approximates the historical, geographical, and linguistic distinction of various natural languages and their dialects.

Prior to RFC 3066bis, all language tags fell into two categories: 1) generative tags made up of a language subtag and optional country code; 2) registered tags that must be considered as a singular unit.

RFC 3066bis changes the structure of the registry so that the generative mechanism is always applicable and greatly restricts the ability to register tags. The highly generative structure makes language tags into a much more robust ontological structure on which to base applications (such as the ones described here).

Tags under the new scheme are comprised of five subtag types, plus user-defined extensions. Each subtag must appear in a specific position in a tag and has unique length and content restrictions. Thus it is always possible to identify each subtag and assign it meaning (even without access to the underlying standards). The four subtag types are: 1) language subtags defined by ISO 639-1 or ISO 639-2; 2) extended language subtags which are reserved for possible use with ISO 639-3 in the future; 3) script subtags defined by ISO 15924; 4) region subtags defined by either ISO 3166 or by the UN M.49 region codes; and 5) variant subtags which are defined either by registration with IANA or by private use subtags for specific expert use.

Thus an example RFC 3066bis tag looks something like:

```
en-Latn-US-boont-x-anExtension
zh-s-min-s-nan-Hant-CN
```

The structure of a tag is described by this BNF notation:

```
Lang* [-s-extlang] [-script] [-region] [-variant] [-x* [-extension]]
```

5 The Use Cases with Current Technology

To motivate the rest of the paper, we show the limitations of the current recommendations when faced with these simple tests.

5.1 Appropriate Display of Labels

Brian has clear, but complex linguistic preferences: 1) English, from whatever geographical region, with a preference for British English. 2) Japanese, in whatever script. 3) Any language in traditional Chinese script 4) French.

Each of these corresponds to a number of possible language tags; the process of matching language tags is covered by the Internet Draft [4] which is published along

with RFC 3066bis. All but the third of Brian's preferences can be described using language ranges, which are also present in the current RFC 3066. The third is an *extended language range* in the terms of [4]. However, there is no support for language ranges in RDF or OWL. So, despite the ontological capabilities of both RFC 3066bis and OWL, someone, (either Brian or the application developer) needs to turn each of these preferences into either a list of explicit alternatives, or a wildcard matching; and then the application developer needs to write custom code in the display routine that selects all the known labels of a resource and find the best match.

This best match needs to be able to cope with both plain literals, in which the language tag is an explicit component, and XML Literals, in which language mark-up is embedded within the literal value (see the Japanese in the example). It is unlikely that Brian will be able to reuse his expression of preference for one Semantic Web application in a second application.

5.2 Finding all Klingon Text in a Knowledge Base

Searching for Klingon text has an additional complexity: there are two possible tags for Klingon, the preferred primary tag of "tli" a recent addition to the ISO 639-2 registry, or the older IANA tag of "i-klingon", grandfathered in RFC 3066bis.

Even if the Semantic Web knowledge base provides support for searching by language tag, or better by language range, and even if that support covers both plain literals and XML Literals, the Star Trek fan still needs to know that there are two tags, and has to perform two searches in place of one.

5.3 Multilingual Knowledge Base Construction

One approach is for each group to check every literal in the subset of the knowledge base that they are using and ensuring that an appropriate translation is available. This involves unnecessary translation work in that some of the literals may never be relevant to the queries being asked, and others may already have a translation available, but only through the application of an OWL reasoner. A second approach would be to ask each relevant query of the knowledge base, and then have custom code to examine the language tags in the literals returned, and to have further custom code to determine whether one such literal meets the linguistic requirements of the application. This custom code should be able to look inside XML Literals for embedded language information. It may also need to support the concepts of language range and language tag fallback from [4].

6 The Formal Machinery

OWL provides a language for describing ontologies: much of this paper shows how OWL can be used to describe the ontology of RFC 3066bis. In this case, it is attractive to integrate this description of an ontology in with the lower level machinery of RDF, and RDF's use of language tags. This section defines a few properties and classes that, with suitable extensions to RDF semantics, serve to

expose the language information in RDF literals in a way that can then be integrated with OWL's class definition mechanisms.

6.1 Properties

The minimal requirement to use OWL's ontological capabilities to capture the relationships between language tags and literals using these tags, is that it should be possible to identify within RDF and OWL the tag of a literal. The current recommendations provide no relationship between the language tag of a literal and any other feature of the language. The approach we take is to define a new property `rdflg:lang` (where `rdflg:` is a new namespace used in this paper).

This property is such that the triple $(x \text{ rdflg:lang } y)$ is true if and only if one of the following:

1. x is a plain literal with language tag z (normalized to lowercase), and y is `z^^xsd:language`
2. x is an XML literal containing non-white character data (a text node or attribute value) in-scope of an `xml:lang= 'z'` (z normalized to lowercase), and y is `z^^xsd:language`

To express this formally as a semantic extension, the above can be read as a definition of the property extension *text*($I(\text{rdflg:lang})$) using the notation of RDF Semantics [6]. This property contravenes the purely syntactic constraint that literals cannot be subjects, but that presents no intrinsic difficulties¹ (other than that a few of the examples in this paper are written with a non-standard notation).

The second part of the definition treats all languages actually used within an XML literal equally. For some applications it may be more appropriate to, when possible, identify the principle language of an XML literal. For example, consider

```
<span xml:lang="en">
  This is <span xml:lang="la">ad hoc</span>.</span>
```

The text is essentially English, even though it contains a Latin phrase. This can be identified by noting that the outermost language tag for all the text is "en". Thus we define a further property `rdflg:mainLang` which relates a XML literal to its outermost language tag (if unique). This similarly constitutes a semantic extension to RDF.

6.2 Classes

An omission from the RDF Vocabulary [10] is that there is no class of plain literals, we rectify that with `rdflg:PlainLiteral`.

A further consideration is that the typical use case involves presenting text to an end-user. An arbitrary piece of XML does not include any presentational guidelines, hence it is useful to sometimes restrict consideration only to XML using the XHTML,

¹ This is not breaking new ground, for example: `"foo" owl:differentFrom "bar"` .

Ruby, SVG, MathML namespaces. Thus we also define a class `rdflg:XHTMLLiteral`, which is a subclass of the datatype `rdfl:XMLLiteral`.

Neither of these classes can be defined using the built-in formal machinery of RDF or OWL semantics, and so, as for the properties, they are suggested as a semantic extension.

6.3 Internationalized Interpretations

To follow the language of RDF Semantics [6], we define an internationalized interpretation I , as an RDFS Interpretation (as defined in [6]), such that:

- for each plain literal with language tag lg in the vocabulary V of I , the typed literal $lg^{xsd:language}$ is in V .
- for each XMLLiteral x in V and every language tag lg occurring in x as the value of an `xml:lang` attribute, $lg^{xsd:language}$ is in V
- the property extensions of `rdflg:lang` and `rdflg:main-lang` are as above
- the class extension of `rdflg:PlainLiteral` and `rdflg:XHTMLLiteral` are as above

7 Defaults in RFC 3066bis

RFC 3066bis has specific advice suggesting defaults for script codes and geographical codes, viz:

Use as precise a tag as possible, but no more specific than is justified. For example, 'de' might suffice for tagging an email written in German, while 'de-CH-1996' is probably unnecessarily precise for such a task. Avoid using subtags that add no distinguishing information about the content. For example, the script subtag in 'en-Latn-US' is generally unnecessary, since nearly all English texts are written in the Latin script.

Unfortunately, defaults are known to be problematic in the Semantic Web, for example OWL contains no provision for them, despite them being an objective of the OWL requirements [2].

There are two possible implications of the language in RFC 3066bis. One is that there are defaults implied by various combinations of subtags, such as the "default" of Latin script for a language tag "en-US".

The other possible implication is that an omitted subtag is an *implied range*, suggesting that a user will accept any value in that position. Thus the tag "en-US" really implies a language tag of "en-*--US-*".

We will address these implications in this paper by assuming the former for tags marking up data and further, that the rules for these defaults are shared public knowledge. Ideally these rules should be maintained by the IETF and kept at IANA. We return to implied ranges, for use when querying data, in section 9.2.

² This suffers from being non-extensible, different application environments may be able to support only XHTML and SVG, or XHTML and MathML etc.

The rules only address script codes and geographical codes, and each rule is a pair. The first item in the pair is a language tag missing either or both of a script subtag or a geographical subtag, the second item include the default values (if any) for the missing items. Some sample rules are as follows:

```
en en-latn
fr fr-latn-FR
zh-TW zh-hant-TW
```

The first rule says that English text defaults to the Latin script, and applies uniformly to any geographical (or private variant) of English, without some other explicit script code. The second rule says that French defaults to being in Latin script and from the geography France. These two defaults apply independently so that lacking a geographical code `fr-arab` defaults to `fr-arab-FR` (the French of France written in Arabic) rather than `fr-arab-015` (North African French written in Arabic)³.

With these rules we note that it becomes impossible to mark up some texts in a very general way. For example, while it is possible to use a language code of `zh` for Chinese in a variety of scripts, it is not possible to use `en` for English in a variety of scripts. This is not an additional constraint on top of RFC 3066bis, but merely an articulation of a limitation that is already implicit within it.

Content authors should use a shorter form of any language tag, if available (following RFC 3066bis). We then apply the default rules by introducing two new properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang` that are defined by axioms derived from the default rules, such as (in the OWL abstract syntax⁴ [7]):

```
EquivalentClass(
  restriction( rdfl-dflt:lang value("en-latn") )
  unionOf( restriction( rdflg:lang value("en-latn") )
            restriction( rdflg:lang value("en") ) ) )

EquivalentClass(
  restriction( rdfl-dflt:lang value("en-latn-us") )
  unionOf( restriction( rdflg:lang value("en-latn-us")
                      restriction( rdflg:lang value("en-us") ) ) )
)
```

The first axiom says that for any resource the property `rdfl-dflt:lang` has the value `en-latn` if and only if the property `rdflg:lang` has the value `en` or `en-latn`.

Note that the second axiom is implicit in the first rule, when combined with the ISO 3166 country code `US`. Since there are many such codes there are very many of these axioms generated. In fact, when we consider private extensions such as `en-US-x-newyorkcity`, there are infinitely many axioms. We return to this issue in section

³ If this was thought inappropriate a more specific rule for `fr-arab` could be included in the table of rules.

⁴ In all the examples we omit the datatype `^^xsd:language` inside the value construct. E.g. `value("en-latn")` abbreviates `value("en-latn" ^^xsd:language)`.

11. The axioms above, like all the axioms with sample language tags in this paper, should be seen as prototypical, invoking an infinite pattern.

For any language tag that is not (explicitly or implicitly) in the defaults table, we equate the `rdflg:lang` and `rdflg:mainLang` properties, e.g.:

```
EquivalentClass(
  restriction( rdfl-dflt:lang value("en-arabic") )
  restriction( rdflg:lang value("en-arabic") ) )
```

We generate an equivalent infinite set of axioms defining `rdfl-dflt:mainLang` in terms of `rdflg:mainLang`.

In combination these axioms apply the defaults table to an RDF or OWL knowledge base, and from hereon we use the properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang` in preference to `rdflg:lang` and `rdflg:mainLang`.

8 The Core DataRanges

In the previous section, we introduced two properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang`, bound into an extended RDF model theory. OWL's ontological modeling capability is class focused, so we move from this simple property view into a class view. Since all these classes are classes of literals, they are dataranges. We use the namespace prefix "`core-lang:`" for these dataranges, with corresponding definitions⁵ such as:

```
DataRange( core-lang:en-latn-us
  intersectionOf( rdflg:PlainLiteral
    restriction( rdfl-dflt:lang, value( "en-latn-us" ) ) ) )
```

i.e. the datarange `core-lang:en-latn-us` is those plain literals which have a value for `rdfl-dflt:lang` of "`en-latn-us`". We have an infinite number of these definitions, one for each possible language tag (including private extension tags). Note that the class name is the language tag normalized to lower case. These classes are pairwise disjoint so that:

```
"hello world"@en-US rdf:type core-lang:en-latn-us .
```

But none of the following are true:

```
"hello world"@en-US rdf:type core-lang:en-latn .
"hello world"@en rdf:type core-lang:en-latn-us .
"hello world" rdf:type core-lang:en-latn .
```

The operation of the defaulting rules for the script code is apparent in these facts. Further, we see that `core-lang:en-us` is empty, since any literal explicitly tagged as `en-US` is subject to the default rule, and treated as `en-latn-US`.

In addition we define `core-lang:None` as the datarange of plain literals without a language tag.

⁵ We extend the abstract syntax with Datarange axioms modeled on the class axiom of OWL DL. These are mapped to triples similarly to the class axiom.

```

DataRange( core-lang:None
  intersectionOf( rdflg:PlainLiteral
    restriction( rdflg:lang, cardinality=0 )))

```

8.1 XML Literals

XML Literals provide additionally complexity in that they may have more than one language tag. We model this complexity with three dataranges for each language tag. Using the prefix `xmllit-all-lang:` we create dataranges for XML Literals all of whose non-white character data content is tagged with the appropriate language tag; using the prefix `xmllit-some-lang:` we create dataranges for XML Literals some of whose non-white character data content is appropriately tagged; using the prefix `xmllit-main-lang:` we create dataranges for XML Literals all of whose non-white character data content is contained within ancestor XML elements with the appropriate `xml:lang` tag even if overridden on a closer ancestor element.

```

# The class of XMLLiterals wholly in language "it"
DataRange(xmllit-all-lang:it
  intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:lang, cardinality = 1 )
    restriction( rdfl-dflt:lang, value("it") )))
# The class of XMLLiterals partially in language "it"
DataRange(xmllit-some-lang:it
  intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:lang, value("it") )))
# The class of XMLLiterals partially in language "it"
DataRange(xmllit-main-lang:it
  intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:mainLang, value("it") )))

```

9 Approximate Matching

The core dataranges of the previous section do not provide any additional utility for our use cases. While they map the language tag into some base classes, the key issue in the use cases was the additional linguistic knowledge needed to make the best use of the language tags. In this section, we should how the ontology implicit in RFC 3066bis can be made explicit with further OWL datarange definitions.

9.1 Presentable Literals

For most use cases, the application wishes to display some natural language text.

In practice this text will be in the RDF graph either as a plain literal or an XML Literal which is XHTML. Other XML Literals are, in general, not useful, since they lack presentational information. Thus for each language tag we define a further datarange, using the namespace `presentable-lang:` defined in terms of the earlier definitions:

```
# Literals with tag "zh-hant", (traditional Chinese)
# suitable for display or other presentation.
DataRange(presentable-lang:zh-hant
  unionOf( core-lang:zh-hant
    intersectionOf( rdflg:XHTMLLiteral
      restriction( rdflg:mainLang, value("zh-hant")
    )))
)))))
```

9.2 Language Ranges

A language range is a way of matching a specific language tag and all extensions of it. Thus for every tag we can create a further datarange (with namespace prefix lang-range:) corresponding to the concept of language range, and the presentable-lang: datarange of all extension tags are subclasses:

```
# Language ranges corresponding of "zh-hant-CN",
# (traditional Chinese in mainland China)
SubClass(presentable-lang:zh-hant-cn lang-range:zh-hant-cn )
SubClass(presentable-lang:zh-hant-cn lang-range:zh-hant )
SubClass(presentable-lang:zh-hant-cn lang-range:zh )
```

In [4], an extended notion of language range including wildcards * is introduced. For example, zh-*-cn, indicates Chinese from mainland China in whatever script. Since the position of the * in this extended language range can be deduced from the subtag lengths, it is optional, and so we write it as zh-cn. This gives an additional axiom:

```
SubClass(presentable-lang:zh-hant-cn lang-range:zh-cn )
```

The language tag zh-CN is usually understood as being in simplified Chinese (script code hans), however that is using the default rules. For the classes we are defining on top of rdf1-dflt:lang, it is necessary to use the fully expanded form zh-hans-cn of the language tag (e.g. lang-range:zh-hans-cn) if that is what is desired. lang-range:zh-cn is understood as with a genuinely unknown script code, i.e. an implied range (cf. section 7).

It would be more accurate to define a language range, and extended language ranges ending in *, as an infinite union formed with all extension tags. We return to this in section 011.

A further extension to the concept of language range is to allow language ranges that contain only a script code, and omit the primary language tag. For example lang-range:latn is for all literals in Latin script; lang-range:hant- for all literals in traditional Chinese script. A sample axiom defining such ranges is:

```
SubClass(presentable-lang:zh-hant-cn lang-range:hant- )
```

9.3 Language Tag Fallback

In [4], a process of language tag fallback is suggested to exploit “a [putative] semantic relationship between two tags that share common prefixes”. This relationship is weaker than that exhibited with language ranges. This process excludes private use

extensions, which are explicitly ignored. We support this in the ontology with the use of three annotation properties on language ranges. `lang-range:fallback1` gives an alternative datarange that ignores extension tags, `lang-range:fallback2` also ignores the geographical code (if any), `lang-range:fallback3` also ignores the script code. Since we have expanded the default script information, it is generally unhelpful to use `lang-range:fallback3` except in specific cases where scripts have some degree of mutual intelligibility (e.g. simplified Chinese and traditional Chinese). An example axiom:

```
DataRange( lang-range:zh-s-min-s-nan-hant-cn
  annotation( lang-range:fallback1 lang-range:zh-hant-cn )
  annotation( lang-range:fallback2 lang-range:zh-hant )
  annotation( lang-range:fallback3 lang-range:zh ) )
```

9.4 Grandfathered Tags

When languages that have registered tags or subtags in the IANA registry are added to the ISO 639 registry, the old IANA entry is updated to show that it has been deprecated in favour of the ISO 639 entry. An example is Klingon, where the IANA tag `i-klingon` has been grandfathered in favour of `tli`. This can be expressed using `owl:DeprecatedClass` by expressing the identify between the corresponding language ranges:

```
DataRange( lang-range:i-klingon Deprecated lang-range:tli )
```

10 The Use Cases Revisited

10.1 Appropriate Display of Labels

Brian (or a tool he is using) can construct a sequence of dataranges from those given in this paper expressing his linguistic preferences. He needs to understand about language tag fallback, and when it is appropriate to use it. He needs to use fully specified language tags, expanded from their default form (e.g. `en-latn-GB` rather than `en-GB`), to form the language range expressions of interest.

10.2 Finding all Klingon Text in a Knowledge Base

`lang-range:tli` includes plain and XML Literals with both the `tli` and `i-klingon` language tags. Using this an OWL reasoner can be used to find all instances of these literals in a knowledge base.

10.3 Multilingual Knowledge Base Construction

Each group can use the language ranges defined above to construct datarange expressions corresponding to the linguistic requirements of their subproject. These expressions can be combined with the queries over the knowledge base, as further

OWL expressions, which can then be passed to an OWL reasoner which can simply identify the work that needs to be done, taking into account all knowledge already implicit in the axioms.

11 Finiteness of Necessary Knowledge

This paper has presented an infinite ontology representing both the (large and finite) generativity of RFC 3066bis through the registered ISO and IANA registered codes and tags, and the infinite potential extensibility of RFC 3066bis through private extension tags. This raises the question of how a semantic web application can load all this knowledge, or does it require special purpose code to implement?

However, we saw in section 06 that we only added two properties and two classes to the core of the theoretical model. Since the actual vocabulary in any knowledge base is finite, only a finite subset of the infinite axioms presented here are relevant to any specific piece of reasoning. This subset can be determined from the language tags used in the vocabulary of the knowledge base. Moreover, the infinite unions mentioned at the end of section 09.2 can be replaced with finite unions determined by the vocabulary actually used. The formal machinery of section 6.3 is expressed only in term of such a finite vocabulary. Thus we structure the ontology described in this paper in such a way that a Semantic Web knowledge base should import a subontology generated by the language tags that occurs in the knowledge base. Since the KB is finite, this is a finite number of tags. The amount of knowledge for each tag is fairly small, and moreover, in a typical knowledge base the number of language tags is small, hence this is only a small overhead.

An appropriate implementation might be with a servlet that accepts a set of language tags in an HTTP POST action, or as a query on a URI, and returns the relevant subontology generated from the prototypes in this paper together with the tags in the request.

12 Additional Expert Knowledge

The ontology presented could be augmented with additional expert knowledge. In particular, there are relationships amongst the script codes, for example “Hrkt” (Hiragana + Katakana) is a plausible fall back for “Hira” (Hiragana) but not for “Brai” (Braille). There are also relationships amongst the geographical codes, particularly with the new addition in RFC 3066bis of the numerical UN geographical codes, for example the geography US is part of the geography 021 (northern America) but not of 419 (Latin America and the Caribbean). This results in plausible corresponding subclass relationships between `lang-range:es-latn-US` (US Spanish) and `lang-range:es-latn-021`, (north American Spanish) but not with `lang-range:es-latn-419` (Latin American Spanish). Fleshing this idea out is left as future work.

13 Suggestions for the Recommendations

We believe that appropriate support for multilingual applications is vital to the Semantic Web. The work presented here provides some significant forward steps. In this section we suggest changes that could be incorporated in future revisions of the three standards we have considered.

13.1 RDF

The model theoretic changes to support the two new properties and two new classes defined in section 06 should be added to the core RDF Recommendations.

13.2 OWL DL

With the changes to RDF, OWL Full would already support all the functionality we have described, using the class expressions we have shown. However, OWL DL excludes the possibility of infinite DataRanges, does not permit named DataRanges, nor does it permit class expressions in the definition of DataRanges. All of these could be added without compromising the design integrity of OWL DL datatyping, which depends on the separation of a datatype oracle as in [11] from the tableau reasoner. The core dataranges of section 8 can be treated as primitive datatypes, and the additional expressivity of forming (finite) unions and intersections of datatypes does not compromise the functioning of the datatype oracle. I.e. the datatype oracle can be extended to include the ability to handle the dataranges we have used.

13.3 RFC 3066bis

The generative capabilities of RFC 3066bis fit fairly naturally into OWL, and are distinctly easier to use than the somewhat *ad hoc* list of tags maintained by IANA under RFC 3066. The systematic inclusion of script codes into RFC 3066bis enables new functionality that recognizes that script is sometimes more important than language when trying to (partially) understand some natural language.

However, the continuation of the defaulting rules from RFC 1766 through to RFC 3066bis creates difficulties for true interoperability. These could be addressed by making the default rules more explicit as described in section 7.

The language range concept we have used is more powerful than the simple mechanism used in RFC 3066bis (for example, taking into account grandfathered codes). It may be appropriate for further revisions of that standard to incorporate some of the ideas.

14 Conclusions

We have shown that relatively small changes to RDF and OWL make it significantly easier to build and use multilingual knowledge bases. The generative capacity of RFC

3066bis can be modeled and exploited in OWL; while this results in an infinite ontology, this is usable in practice, because for any knowledge base a finite subset suffices.

References

- [1] Klyne, G., Carroll, J. J. (eds.): RDF Concepts and Abstract Syntax. W3C Rec. 2004
- [2] Heflin, J. (ed) OWL Use Cases and Requirements W3C Rec. 2004.
- [3] Phillips, A., Davis, M.: Tags for Identifying Languages. draft-ietf-ltru-registry-00, 2005 (also known as RFC 3066bis).
- [4] Phillips, A., Davis, M.: Matching Language Identifiers. draft-ietf-ltru-matching-00, 2005.
- [5] Carroll, J.J. "An Introduction to the Semantic Web: Considerations for building multilingual Semantic Web sites and applications", Multilingual Computing #68, Volume 15 Issue 7 pp 19-24, 2004.
- [6] Hayes, P. (ed.): RDF Semantics. W3C Rec 2004
- [7] Patel-Schneider, P.F., Hayes, P., Horrocks, I. (eds.): OWL Semantics and Abstract Syntax. W3C Rec. 2004.
- [8] Carroll, J.J., de Roo, J. (eds.): OWL Test Cases, W3C Rec. 2004.
- [9] Sawicki, M., Suignard, M., Ishikawa, M., Dürst, M., Texin, T., (eds) Ruby Annotation, W3C Rec. 2001.
- [10] Brickley, D. Guha, R.V. (eds) RDF Vocabulary, W3C Rec 2004.
- [11] Pan, J. and Horrocks, I. Extending Datatype Support in Web Ontology Reasoning, CoopIS/DOA/ODBASE 2002 pp 1067-1081 2002

RDFSculpt: Managing RDF Schemas Under Set-Like Semantics

Zoi Kaoudi^{1,2}, Theodore Dalamagas², and Timos Sellis²

¹ Dept. of Electronic and Computer Engineering,
Technical University of Crete, Greece
`zoi@intelligence.tuc.gr`

² School of Electr. and Comp. Engineering,
National Technical University of Athens, Greece
`{zkaoudi, dalamag, timos}@dmlab.ece.ntua.gr`

Abstract. The Semantic Web is an extension of the current Web in which information is given well-defined meaning to support effective data discovery and integration. The RDF framework is a key issue for the Semantic Web. It can be used in resource discovery to provide better search engine capabilities, in cataloging for describing the content of thematic hierarchies in thematic catalogs and digital libraries, in knowledge sharing and exchange of Web agents, etc. Up to now, RDF schemas have been treated rather as sets of individual elements (i.e. model primitives like classes, properties, etc.). Under that view, queries like “find the part of a portal catalog which is not present in another catalog” can be answered only in a procedural way, specifying which nodes to select and how to get them. For this reason, we argue that answering such queries requires treating schemas as a whole rather than as sets of individual elements. We introduce a set of operators with set-like semantics to manage RDF schemas. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. We also present RDFSculpt, a prototype system that implements our framework.

1 Introduction

The Semantic Web [1] is an extension of the current Web in which information is given well-defined meaning to support effective data discovery and integration. The Semantic Web will provide the necessary infrastructure for Web pages, database systems, services, scripts, sensors, etc., to consume and produce data on the Web. For the Semantic Web to function, the information on the Web should become more machine-understandable. For this reason, new languages and models have been proposed to semantically enrich data on the Web. The RDF framework¹ is a foundation for processing metadata, that is data for the meaning of data. In an RDF document, one can make statements about particular Web resources, that is Web pages, page authors, scripts, etc. The RDF

¹ <http://www.w3c.org/RDF>

schema [14] provides mechanisms for describing groups of related resources and the relationships between these resources, acting as a semantic extension of RDF. The RDF schema description language is based on *classes* and *properties*, and is similar to the type system of object-oriented programming languages.

The RDF framework is a key issue for the Semantic Web. It can be used in resource discovery to provide better search engine capabilities, in cataloging for describing the content of thematic hierarchies in thematic catalogs and digital libraries, in knowledge sharing and exchange of Web agents, etc. In [6] benchmarks are presented to provide structural and statistical analysis for volumes of RDF data collected from the Web.

Up to now, RDF schemas have been treated rather as sets of individual elements (i.e. model primitives like classes, properties, etc.). However, in the Web environment, where searching in a knowledge domain requires information processing in many sources related to that domain, new query requirements arise for manipulating RDF schemas as a whole, like for example (Q_1) **find the part of Movie Catalog 1 which is not present in Movie Catalog 2** (see Figure 1(a) with examples of Movie Catalogs). Such a query has a ‘difference’ flavor

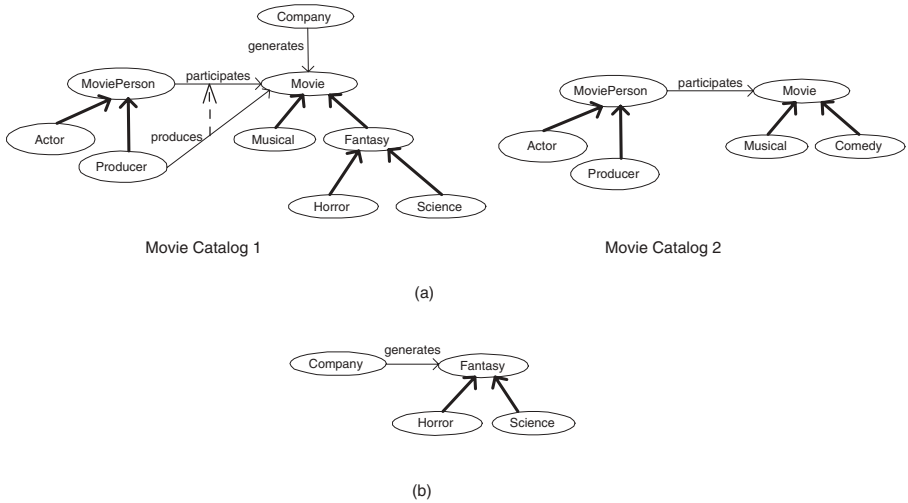


Fig. 1. (a) Parts of Movie Catalog 1 and 2. (b) The part of Movie Catalog 1 which is not present in Movie Catalog 2

and its answer should include schema information present in Movie Catalog 1 but not in Movie Catalog 2. Figure 1(b) shows the result of Q_1 . The resulting catalog has **Company**, **Fantasy**, **Horror** and **Science**, a categorization found only in Movie Catalog 1. Other example queries follow:

- (Q_2) **find the integrated catalog provided by Movie Catalog 1 and Movie Catalog 2** (with a ‘union’ flavor),

- (Q_3) find the common part of Movie Catalog 1 and Movie Catalog 2 (with an ‘intersection’ flavor),
- (Q_4) find the integrated catalog using the common part of Movie Catalog 1 and Movie Catalog 2, and another third Movie Catalog (a sequence of ‘intersection’ and ‘union’ subqueries).

Viewing the RDF schemas as a set of individual elements requires the usage of RDF query languages like RQL [5] in a procedural way. The user should specify which RDF nodes to select and how to get them to answer queries like Q_1, Q_2, Q_3, Q_4 . For this reason, we argue that answering such queries requires treating schemas as full-fledge objects rather than as sets of individual elements, and introducing a set of operators applied on RDF schemas as a whole.

Queries like the above produce *integrated* RDF schemas. Integration of schemas, in general, is considered as the task which produces a global schema to cover all involved schemas and is a widely studied research topic [10]. Our work, on the other hand, supports the integration of RDF schemas based on union, intersection and difference semantics provided by a set of operators. The integrated schema is the output of such operators applied on the involved schemas.

We classify such kind of query requirements as part of the *generic model management* framework presented in [2, 8]. According to this framework, models are manipulated as abstractions rather than sets of individual elements, using *model-at-a-time* and *mapping-at-a-time* operators.

Contribution. This paper introduces operators to manipulate RDF schemas. The operators are applied on RDF schema graphs and produce new, integrated RDF schema graphs. The key feature of our framework is that such integration is based on set-like semantics. We define three binary operators (union, intersection, difference) that can be applied on RDF schema graphs as a whole, and produce new ones. We also define a unary operator that can be applied on one RDF schema graph and return a part (subset) of it. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. We have implemented the operators in RDFSculpt, a prototype system for managing RDF schemas.

Related Work. Recently, there have been suggested quite a few RDF query languages. RQL [5] is a typed functional language (in the form of OQL) to uniformly query both RDF data descriptions and schemas. In [7], RVL is presented as an extension of RQL that supports views on RDF. RDQL [12] is an SQL-like RDF query language, manipulating RDF data as triple patterns. Sesame [3] is an RDF management system. SeRQL (the native language of Sesame) is used to query both RDF data descriptions and schemas. Views on RDF data are provided using CONSTRUCT queries. Triple [13] is a layered and modular rule language based on Horn-logic which is syntactically extended to support features for querying and transforming RDF models. Most of the above languages support algebraic operations on RDF statements (i.e. data) and not on RDF schemas. A survey of RDF query languages is presented in [4]. The operators suggested in this paper

can be included in any query language to answer queries that require operations on RDF schemas as full-fledged objects rather than as sets of individual elements.

Operators for ontology composition have been studied in [15, 9]. These operators are based on articulation rules, that is rules to establish correspondence between concepts in different ontologies. Our work differs since we emphasize on the semantics of RDF schema graphs. Also, we compose RDF schemas which are related to a global RDF schema through the subset relation that we define, and not by using articulation rules.

Outline. The rest of this paper is structured as follows. In Section 2, we discuss modelling issues for RDF schemas and we introduce RDF schema subsets and projections. Section 3 defines three RDF schema operators: union, intersection and difference. In Section 4, we describe the RDFSculpt prototype system that implements the suggested operators. Finally, Section 5 concludes this paper and discusses further work.

2 Modelling Issues

RDF schemas provide a type system for RDF. The primitives of RDF schemas are classes and properties. Classes describe general concepts and entities. Properties describe the characteristics of classes. They also represent the relationships that exist between classes. Classes and properties are primitives similar to those of the type system of object-oriented programming languages. The difference is that properties in RDF schemas are considered as first-class citizens and are defined independently from classes.

Classes are described using the RDF schema resources `rdf:Class` and `rdfs:subClassOf`, while properties are described using the RDF class `rdf:Property` and the property `rdfs:subPropertyOf`. The `rdfs:domain` property is used to indicate that a particular property applies to a designated class. The `rdfs:range` property is used to indicate that the values of a particular property are instances of a designated class. RDF schemas can be modelled as directed labelled graphs. For example, consider the graph shown in Figure 2. The oval labelled nodes represent classes. The rectangular labelled nodes denote literals, like string, integer, etc. The plain labelled edges represent properties. The thick edges define an *isA* hierarchy (class/subclass) of classes, while the thick, dashed edges de-

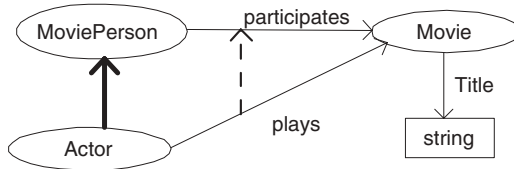


Fig. 2. An example of an RDF schema

fine an *isA* hierarchy (property/subproperty) of properties. For example, the class *Actor* is a subclass of *MoviePerson*, while the property *plays* is a subproperty of *participates*. The class *MoviePerson* is the domain of the property *participates*, while the class *Movie* is the range of *participates*. Formally, an RDF schema is defined as follows:

Definition 1. An RDF schema (RDFS) is a 5-tuple (C, L, P, SC, SP) representing a graph, where:

1. C is a set of labelled nodes. Each node in C represents an RDF class.
2. L is a set of nodes labelled with data types defined in XML schema [11], e.g. integer, string etc. Each node in L represents a literal.
3. P is a set of directed labelled edges (c_1, c_2, p) from node c_1 to node c_2 with label p , where $c_1 \in C$ and $c_2 \in C \cup L$. Each edge in P represents an RDF property p with domain c_1 and range c_2 .
4. SC is a set of directed edges (c_1, c_2) from node c_1 to node c_2 , where $c_1, c_2 \in C$. Each edge in SC represents an *isA* relationship between classes c_1 and c_2 (i.e. c_1 is a subclass of c_2).
5. SP is a set of directed edges $((c_1, c_2, p_1), (c_3, c_4, p_2))$ from edge (c_1, c_2, p_1) to edge (c_3, c_4, p_2) , where $(c_1, c_2, p_1), (c_3, c_4, p_2) \in P$. Each edge in SP represents an *isA* relationship between property (c_1, c_2, p_1) and property (c_3, c_4, p_2) (i.e. that is (c_1, c_2, p_1) is a subproperty of (c_3, c_4, p_2)).

Let $\preceq_C$ be a relation on C : $c_1 \preceq_C c_2$ holds if c_1 is a subclass of c_2 . With $\preceq_C^+$ we denote the transitive closure of $\preceq_C$. We consider c_1 to be an *ancestor* of c_2 (or c_2 to be a *descendant* of c_1) if $c_2 \preceq_C^+ c_1$. Similarly, let $\preceq_P$ be a relation on P : $(c_1, c_2, p_1) \preceq_P (c_3, c_4, p_2)$ holds if (c_1, c_2, p_1) is a subproperty of (c_3, c_4, p_2) . With $\preceq_P^+$ we denote the transitive closure of $\preceq_P$. We consider (c_1, c_2, p_1) to be an *ancestor* of (c_3, c_4, p_2) (or (c_3, c_4, p_2) to be a *descendant* of (c_1, c_2, p_1)) if $(c_3, c_4, p_2) \preceq_P^+ (c_1, c_2, p_1)$.

2.1 RDF Schema Subsets

We next introduce the concept of the *subset* relation for RDF schemas. Intuitively, an RDF schema R_1 is a subset of an RDF schema R_2 when R_1 contains some of the elements (i.e. classes, properties, etc.) of R_2 , and it does not violate the *isA* hierarchy of classes and properties maintained in R_2 .

Definition 2. Let $R_i = (C_i, L_i, P_i, SC_i, SP_i)$ and $R_j = (C_j, L_j, P_j, SC_j, SP_j)$ be two RDF schemas. R_i is a subset of R_j , denoted by $R_i \subseteq R_j$, if:

1. $C_i \subseteq C_j$.
2. $L_i \subseteq L_j$.
3. for each edge $(c_1, c_2, p_1) \in P_i$ there is an edge $(c_3, c_4, p_2) \in P_j$ with $(c_1 \equiv c_3 \text{ or } c_1 \preceq_{C_j}^+ c_3)$ and $(c_2 \equiv c_4 \text{ or } c_2 \preceq_{C_j}^+ c_4)$ and $p_1 = p_2$.
4. for each pair of nodes $c_1, c_2 \in C_i$,
if $c_1 \preceq_{C_i} c_2$ then $c_1 \preceq_{C_j}^+ c_2$ and
if $c_1 \preceq_{C_j}^+ c_2$ then $c_1 \preceq_{C_i}^+ c_2$.

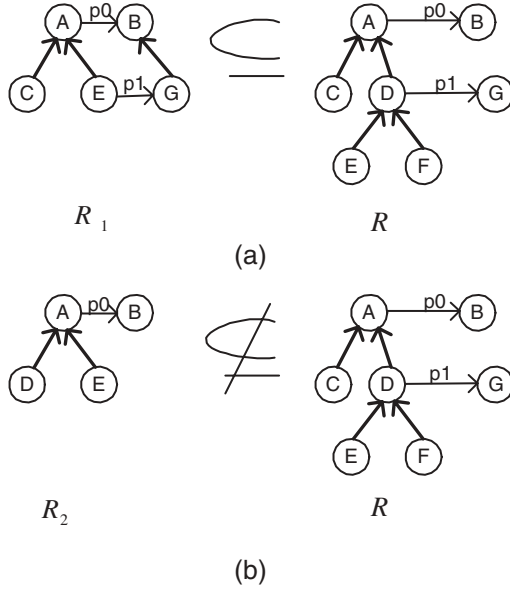


Fig. 3. Examples of RDF schema subsets

5. for each pair of edges $(c_1, c_2, p_1), (c_3, c_4, p_2) \in P_i$,
 if $(c_1, c_2, p_1) \preceq_{P_i}^+ (c_3, c_4, p_2)$ then $(c_1, c_2, p_1) \preceq_{P_j}^+ (c_3, c_4, p_2)$ and
 if $(c_1, c_2, p_1) \preceq_{P_j}^+ (c_3, c_4, p_2)$ then $(c_1, c_2, p_1) \preceq_{P_i}^+ (c_3, c_4, p_2)$.

Figure 3(a) shows the RDF schema R_1 which is a subset of R , since it satisfies all conditions of the definition. For example, having $C_1 = \{A, B, C, E, G\}$ and $C = \{A, B, C, D, E, F, G\}$, $C_1 \subseteq C$. Also, for each pair of nodes in C_1 the fourth condition of the above definition holds (e.g. $A \preceq_{C_1} E$ and $A \preceq_C^+ E$ hold, and $A \preceq_C^+ E$ and $A \preceq_{C_1} E$ hold as well for nodes A, E in C_1). On the other hand, the RDF schema R_2 is not a subset of R in Figure 3(b). The fourth condition of the definition is violated; although $D \preceq_C^+ E$ holds, $D \preceq_{C_2}^+ E$ does not hold.

In this work we manipulate RDF schemas which are subsets of a given RDF schema, called *global RDF schema*.

Definition 3. Let $S = \{R_1, R_2, \dots, R_n\}$ be a set of RDF schemas. A global RDF schema for S is an RDF schema R such that $R_i \subseteq R, 1 \leq i \leq n$.

2.2 Projecting RDF Schemas

This section defines the operator of projection on RDF schemas. Projecting RDF schemas is based on a given set of RDF classes and involves the extraction of a part of an RDF schema that includes at least those classes. Before we present the projection operator in detail, we give some definitions which are useful to the

discussion that will follow. All subsequent definitions refer to an RDF schema $R = (C, L, P, SC, SP)$.

Definition 4. *The extended domain of a property $(c, s, p) \in P$, denoted by $\mathcal{D}^+((c, s, p))$, is the set of classes $\{c, c_1, \dots, c_n\}$, where $\{c_1, \dots, c_n\}$ are all descendants of c .*

Using the *extended domain* of a property we refer to all classes which can be applied to a property as a set. For example, in the RDF schema of Figure 2 we have $\mathcal{D}^+(\text{MoviePerson}, \text{Movie}, \text{participates}) = \{\text{Movieperson}, \text{Actor}\}$. Similarly, we define the *extended range* of a property to refer to all the classes from which a property can take values as a set.

Definition 5. *The extended range of a property $(e, c, p) \in P$, denoted by $\mathcal{R}^+((e, c, p))$, is the set of classes $\{c, c_1, \dots, c_n\}$, where $\{c_1, \dots, c_n\}$ are all descendants of c .*

Below we define the nearest common ancestor of a set of classes. Intuitively, it is the class which is the lower-level ancestor of all classes in the set. For example, $\text{nca}\{G, H, F\} = B$ in R_G of Figure 5.

Definition 6. *The nearest common ancestor of a set of classes $C_s \subseteq C$, where C_s consists of more than two classes, denoted by $\text{nca}(C_s)$, is the class z such that for all $x \in C_s$ $x \preceq_C^+ z$ holds and there is no class $y \in C$ such that $x \preceq_C^+ y \preceq_C^+ z$. The nearest common ancestor of one class is the class itself.*

We now define the projection operator for RDF schemas. Intuitively, a projection on an RDF schema R , given a set of classes C_s , results in a subset of R that has all classes from C_s , their involved properties and some other classes, the role of which will be clarified shortly.

Consider the RDF schema R in Figure 4. Projecting R with $C_s = \{C, D, B\}$ results in an RDF schema which includes classes C, D, B and the involved property (D, B, p_2) . Class A (and its involved property (A, B, p_1)) is also part of the result although $A \notin C_s$. In general, classes like A (which are actually nearest common ancestors for classes that are included in C_s) are used to resolve the issue of having more than one classes as domain (or range) for a property. Another example of projection is presented in Figure 5. The formal definition for the projection operator follows.

Definition 7. *Let $R = (C, L, P, SC, SP)$ be an RDF schema and $C_s \subseteq C$ a set of classes. The projection Π on R , given C_s , denoted as $\Pi_{C_s}(R)$, is the RDF schema $R' = (C', L', P', SC', SP')$, where:*

1. $(c'_1, c'_2, p') \in P'$ if $\exists (c_1, c_2, p) \in P$, with $C_s \cap \mathcal{D}^+((c_1, c_2, p)) \neq \emptyset$ and $C_s \cap \mathcal{R}^+((c_1, c_2, p)) \neq \emptyset$, where
 - (a) c'_1 is the $\text{nca}(C_s \cap \mathcal{D}^+((c_1, c_2, p)))$,
 - (b) c'_2 is the $\text{nca}(C_s \cap \mathcal{R}^+((c_1, c_2, p)))$, and
 - (c) $p' = p$.

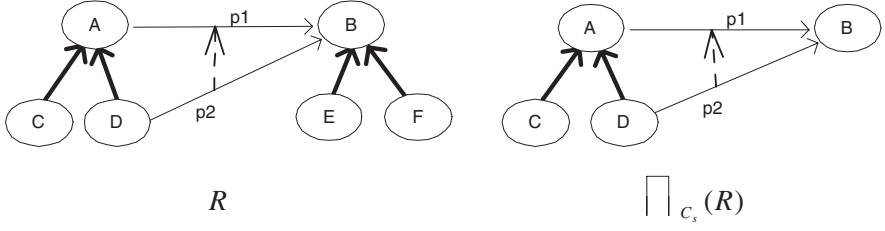


Fig. 4. Projecting R with $C_s = \{C, D, B\}$

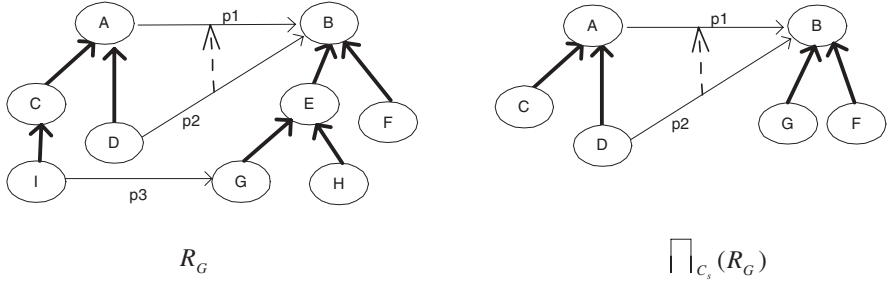


Fig. 5. Projecting R_G with $C_s = \{C, D, G, F\}$

2. $(c'_1, c'_2) \in SC'$ if $c'_1 \preceq_C^+ c'_2$.
3. $((c'_1, c'_2, p'_1), (c'_3, c'_4, p'_2)) \in SP'$ if $(c'_1, c'_2, p'_1) \preceq_P^+ (c'_3, c'_4, p'_2)$.
4. $C' = C_s \cup C_a$, where $C_a = (\cup_i c'_i) \cup (\cup_i s'_i)$, for all $(c'_i, s'_i, p'_i) \in P'$.
5. $L' = \{l \in L \mid \exists c \in C_s \text{ and } (c, l, p) \in P\}$.

3 Set-Like RDF Schema Operators

We define three binary operators applied on RDF schema graphs. The operators can be included in any RDF query language and support manipulation of RDF schemas as full-fledged objects, under union, intersection and difference semantics. In all presented examples, the RDF schemas are subsets of R_G illustrated in Figure 5.

3.1 Union

The *union* operator merges two RDF schemas and results in a new RDF schema that contains all elements from both schemas, without violating the *isA* hierarchies for the involved classes and properties. The union operator for two RDF schemas R_1 and R_2 is defined as a projection on the global schema R_G , given the (set) union of class sets of R_1 and R_2 , respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

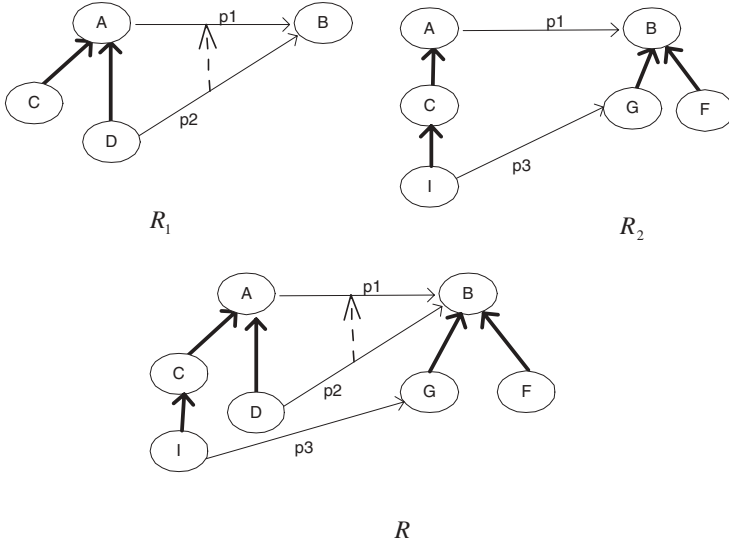


Fig. 6. An example of union operation: $R = R_1 \cup R_2$

Definition 8. Let $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ and $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ be two RDF schemas with $R_1, R_2 \subseteq R_G$ and $C = C_1 \cup C_2$. The union of R_1 and R_2 , denoted by $R_1 \cup R_2$, is the RDF schema $R = \Pi_C(R_G)$.

Figure 6 shows an example of union operation.

3.2 Intersection

The *intersection* operator results in a new RDF schema that contains only common elements from both schemas, keeping the *isa* hierarchies for the involved classes and properties. The intersection operator for two RDF schemas R_1 and R_2 is defined as a projection on the global schema R_G , given the (set) intersection of class sets of R_1 and R_2 , respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

Definition 9. Let $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ and $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ be two RDF schemas with $R_1, R_2 \subseteq R_G$ and $C = C_1 \cup C_2$. The intersection of R_1 and R_2 , denoted by $R_1 \cap R_2$, is the RDF schema $R = \Pi_C(R_G)$.

Figure 7 shows an example of intersection operation.

3.3 Difference

The *difference* operator results in a new RDF schema that contains elements of one schema which are not present in another one, keeping the *isa* hierarchies for the involved classes and properties. The difference operator for two RDF schemas

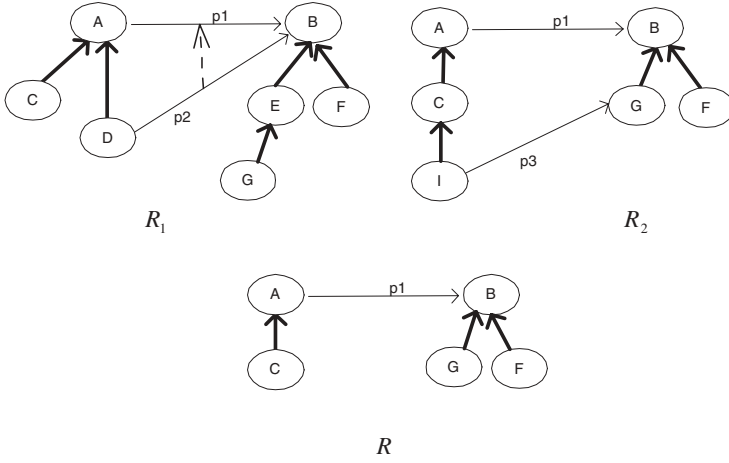


Fig. 7. An example of intersection operation: $R = R_1 \cap R_2$

R_1 and R_2 is defined as a projection on the global schema R_G , given the (set) difference of class sets of R_1 and R_2 , respectively. The final RDF schema R is a subset of R_G ($R \subseteq R_G$).

Definition 10. Let $R_1 = (C_1, L_1, P_1, SC_1, SP_1)$ and $R_2 = (C_2, L_2, P_2, SC_2, SC_2)$ be two RDF schemas with $R_1, R_2 \subseteq R_G$ and $C = C_1 - C_2$. The difference of the two RDF schemas, denoted by $R_1 - R_2$, is the RDF schema $R = \Pi_C(R_G)$.

Figure 8 shows an example of difference operation.

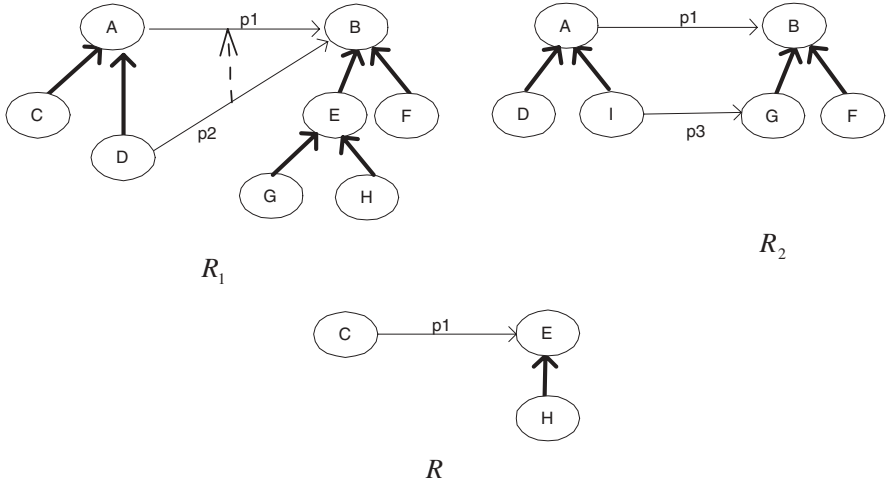


Fig. 8. An example of difference operation: $R = R_1 - R_2$

4 The RDFSculpt Prototype

The RDFSculpt is a prototype system for RDF schema management and implements the operators suggested in this paper. As shown in Figure 9, the system

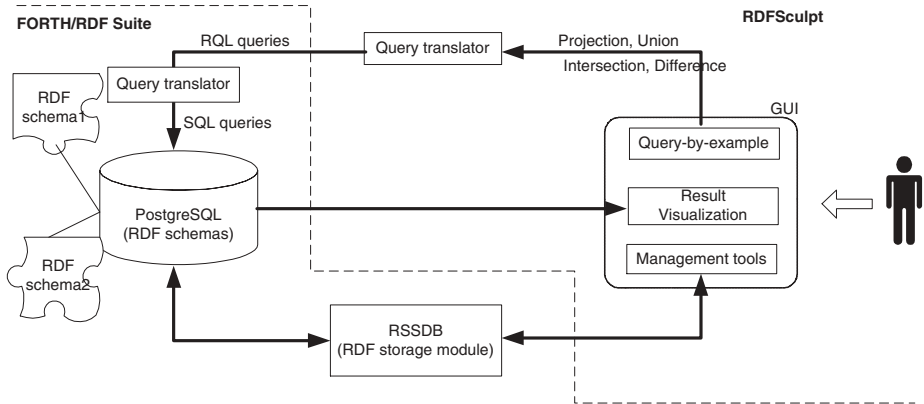


Fig. 9. The architecture of RDFSculpt

is built on top of the ICS-FORTH RDFSuite². To this extend, it exploits all RDF management and query APIs offered by RDFSuite. Users can issue queries on RDF schemas and produce new, integrated ones, using projection, union, intersection and difference operators. The RDFSculpt assists the user in queries formulation, offering her query-by-example capabilities. Results are visualized using RDFSViz³. Figure 10 shows some screen shots of the system.

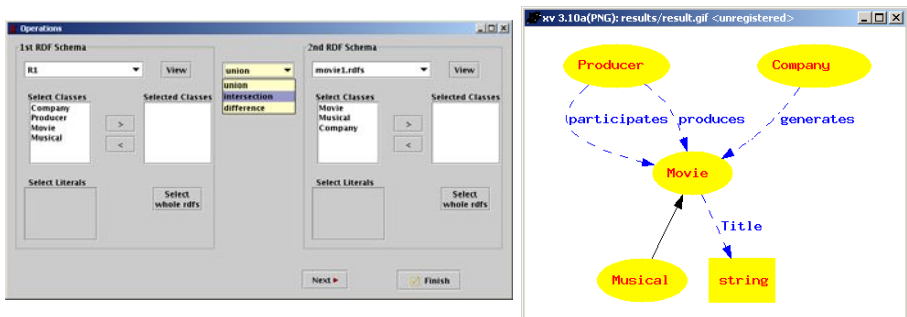


Fig. 10. Screen shots of the RDFSculpt prototype system

² <http://www.ics.forth.gr/is1/RDF/index.html>

³ <http://www.dfki.uni-kl.de/frodo/RDFSViz/>

4.1 Query Processing in RDFSculpt

We next describe in detail how RDFSculpt executes a projection operator on an RDF schema.

Algorithm projection(C_s)

```

/*  $D$ ,  $\mathcal{D}^+$ ,  $R$ ,  $\mathcal{R}^+$  denote property domain, extended domain,
range and extended range, respectively */
1 for each class  $c$  in  $C_s$  do
2    $P$  = properties that have class  $c$  in their  $\mathcal{D}^+$ ;
3   for each property  $p$  in  $P$  do
4      $R$  = classes that are in the  $\mathcal{R}^+$  of  $p$ ;
5      $R = R \cap C$ ;
6   endfor
7   for each  $p$  in  $P$  do
8     /* Working in the domain of  $p$  */
9     if there is only one class in  $\mathcal{D}^+$  of  $p$ , then have this class as  $D$  of  $p$ 
10    else
11      let  $R'$  be the classes in  $\mathcal{D}^+$  of  $p$ ;
12      if all classes in  $R'$  are in the same path of the isA hierarchy
13        then have the higher-level one as the domain of  $p$ 
14      else
15        find the nearest common ancestor of  $R'$ , have it as  $D$  of  $p$ ,
16        and add it in  $C_s$ ;
17      endif
18    endif
19    /* Working in the range of  $p$  */
20    if there is only one class in  $\mathcal{R}^+$  of  $p$ , then have this class as  $R$  of  $p$ 
21    else
22      let  $R'$  be the classes in  $\mathcal{R}^+$  of  $p$ ;
23      if all classes in  $R'$  are in the same path of the isA hierarchy
24        then have the higher-level one as the range of  $p$ 
25      else
26        find the nearest common ancestor of  $R'$ , have it as  $R$  of  $p$ ,
27        and add it in  $C_s$ ;
28      endif
29    endif
30  endfor
31 endfor

```

For example, consider the projection shown in Figure 4, where $C_s = \{B, C, D\}$. For class $c = D$ (line 1) we get $P = \{p_1, p_2\}$ (line 2). For property p_1 (line 3) we get $R = \{B, E, F\}$ (line 4), and after the intersection with C_s we get $R = \{B\}$ (line 5). Working in the domain of p_1 , we have $\mathcal{D}^+ = \{C, D\}$. Therefore, there is more than one classes in $\mathcal{D}^+$ (line 10) and those classes ($\{C, D\}$) are not in the same path (line 14). Consequently, we find the nearest common ancestor of C and D (line 15), which is class A , and add it in C_s (line 16).

Projection is implemented by posing a set of appropriate RQL queries to the RDF storage module. Below we show some examples of RQL queries used to implement projection (given the set of classes $C_s = \{B, C, D\}$) for some steps of the previous algorithm. Details about the RQL language can be found in [5]. For convenience, we note here that the prefix $\$$ denotes a class variable, the prefix $@$ a property variable and the expression $\{; B\}$ denotes a filtering condition of schema classes, taking into account the `rdfs:subClassOf` links. For instance, the path expression $\{; D\}@P$ denotes that the domain of $@P$ is denoted to be class D or any of its superclasses. Furthermore, $nca(B, C)$ is a function of RQL which finds the nearest common ancestor of two nodes.

- In line 2, in order to find the properties of a specific domain, let it be B , we use the RQL query:
`select @P from {; B}@P{ $C }`
- In line 4, in order to find the classes in $\mathcal{R}^+$ of a specific property, let it be p , we use the RQL query:
`select $C from p{ $C }`
- In line 12, in order to check if classes in $\mathcal{D}^+$ are in the same path, we make use of the RQL queries:
`domain(p)`
`superClassOf(B)`
- In line 15, in order to find the nearest common ancestor of R' , we make use of the RQL query:
`nca(B, C)`
- In line 23, in order to check if classes in $\mathcal{R}^+$ are in the same path, we make use of the RQL queries:
`range(p)`
`superClassOf(B)`
- In line 26, in order to find the nearest common ancestor of R' , we make use of the RQL query:
`nca(B, C)`

The union, intersection and difference operators are implemented as projections, with C_s being the (set) union, (set) intersection and (set) difference of class sets in the involved RDF schemas, respectively.

We should note here that the existence of blank nodes in an RDF schema does not affect our approach. A blank node does not have a URI. However, it should have a unique identifier to distinguish itself from other blank nodes in the RDF schema. In that case, blank nodes can be treated as classes and included in the set of classes C_s (see Definition 7).

5 Conclusion

This paper introduced a set of operators to manage RDF schemas. They are applied on RDF schema graphs and produce new, integrated RDF schema graphs.

Such integration is based on set-like semantics. Specifically, we formalized the notion of RDF schema subsets, and we defined a unary operator (projection) to extract parts (subsets) of RDF schemas. Based on projection, we defined three binary operators: union, intersection and difference. The operators can be included in any RDF query language to support manipulation of RDF schemas as full-fledged objects. Finally, we described RDFSculpt, a prototype system for managing RDF schemas that implements our framework.

We are currently working towards the following directions. We are first studying the algebraic properties of the presented operators to show formally that they obey all known laws of set theory. The other research direction involves extending our framework to manage RDF schemas under the assumption that the global RDF schema is not given, but it should be constructed from the available RDF schemas. Furthermore, we are planning to layer our operations to deal with other RDF vocabularies and not only RDF schema graphs.

References

1. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.
2. Philip A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, USA, 2003.
3. Jeen Broedstra, Arjohn Kampman, and Frank van Harmelen. Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Chia, Sardinia, Italy, 2002.
4. Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A Comparison of RDF Query Languages. In *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, Hiroshima, Japan, 2004.
5. Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th International World Wide Web Conference (WWW'02)*, Honolulu, Hawaii, USA, 2002.
6. Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, and Dimitris Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02)*, Chia, Sardinia, Italy, 2002.
7. Aimilia Magkanaraki, Val Tannen, Vassilis Christophides, and Dimitris Plexousakis. Viewing the Semantic Web through RVL Lenses. *Journal of Web Semantics*, 1:4, 2004.
8. Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *Proceedings of the Special Interest Group on Management of Data (SIGMOD) Conference (ACM SIGMOD 2003)*, 2003.
9. Prasenjit Mitra and Gio Wiederhold. An Algebra for Semantic Interoperability of Information Sources. In *Proceedings of the IEEE Symposium on BioInformatics and Bioengineering (BIBE'01)*, Bethesda, MD, Nov. 2001.
10. Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

11. W3C Recommendation. XML Schema Part 2: Datatypes Second Edition, 2004. <http://www.w3.org/TR/xmlschema-2/>.
12. Andy Seaborne. RDQL - A Query Language for RDF. W3C member submission, January 2004. <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>.
13. Michael Sintek and Stefan Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the Deductive Databases and Knowledge Management Workshop (DDLW 2001)*, Tokyo, Japan, 2001.
14. W3C. Resource Description Framework (RDF) Schema Specification 1.0, 2001. <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>.
15. Gio Wiederhold. An Algebra for Ontology Composition. In *Proceedings of the Monterey Workshop on Formal Methods*, U.S. Naval Postgraduate School, Monterey CA, 1994.

REDD: An Algorithm for Redundancy Detection in RDF Models

Floriana Esposito, Luigi Iannone, Ignazio Palmisano,
Domenico Redavid, and Giovanni Semeraro

Dipartimento di Informatica,
Università degli Studi di Bari,
Campus, Via Orabona 4, 70125 Bari, Italy
{esposito, iannone, palmisano, d.redavid, semeraro}@di.uniba.it

Abstract. The base of Semantic Web specifications is Resource Description Framework (RDF) as a standard for expressing metadata. RDF has a simple object model, allowing for easy design of knowledge bases. This implies that the size of knowledge bases can dramatically increase; therefore, it is necessary to take into account both scalability and space consumption when storing such bases. Some theoretical results related to blank node semantics can be exploited in order to design techniques that optimize, among others, space requirements in storing RDF descriptions. We present an algorithm, called REDD, that exploits these theoretical results and optimizes the space used by a RDF description.

1 Motivation

The realization of the Semantic Web (SW) vision [1] needs ontologies for generating or interpreting (semantic) metadata for resources. It is fundamental to have ontology creation and integration steps in order to share structural knowledge between ontology designers and users. Ontologies are to be expressed in RDF according to SW specifications, using languages such as RDFS¹ and OWL.² It is important to note that both RDFS and OWL ontologies can be expressed as RDF graphs, so that ontologies can be treated exactly as other RDF models. In RDF design, the least power principle was applied: data structures are to be kept as simple as possible. This imposes to have very simple basic components, that are URIs³, blank nodes and statements (or triples). These design decisions have the drawback that RDF descriptions tend to grow fast as the complexity of the knowledge they represent increases. This observation encourages SW research to investigate toward the most effective storage solutions for RDF knowledge bases, in order to minimize required space. Intuitively, the lesser the number of triples a software (say, a query engine) has to examine, the faster it will process them.

¹ <http://www.w3c.org/TR/rdf-schema>

² <http://www.w3c.org/2004/OWL>

³ <http://www.w3.org/Addressing/>

This issue has already been deeply investigated, as reported in the section 2.2; recently, some theoretical results were issued by both W3C in [6] and by Gutierrez et al. in [4]. Actually, these results apply also to RDFS, but in this paper we will refer only to blank node semantics. Relying on these results, we developed an algorithm to detect redundancies introduced by blank nodes in a RDF Description. Such redundancies can be removed by mapping blank nodes into concrete URIs or into different blank nodes, without changing or diminishing the RDF graph semantics. In other words, some descriptions can be expressed with lesser triples with no semantic loss.

Moreover, redundancy detection can turn out to be useful in higher level tasks, such as ontology design and alignment. Let us suppose to have designed some classes (say in OWL) and let one of them be a cardinality restriction. If somewhere in the ontology it has a name (an URI), as depicted in Figure 1 (i.e.: `ns:Test`), and somewhere else we created the same restriction without using a name (so using an anonymous restriction class), we would have defined this class twice unnecessarily, so intuitively we introduced redundancy. This kind of repetitions can be detected thanks to blank node semantics and removed, thus simplifying the design of the ontology. The same situation occurs, obviously, if both the restrictions are represented by blank nodes.

Another situation in which the algorithm can be useful is in ontology importing, i.e. the use of the *owl:imports* directive. In this case, let *A* and *B* be two ontologies, and *A owl:imports B*; referring to the restriction example, if there is an anonymous restriction in *B*, and *A* needs the same restriction, the designer of *A* needs to define its own restriction (since a blank node cannot be identified from outside the model in which it is defined). From the OWL point of view, however, *A* contains every statement in *B*, so the complete model (i.e. the model containing *A* plus the import closure) is redundant. The problem can become serious if there are multiple *owl:imports*. Suppose an ontology is imported more than once, e.g. *A owl:imports B*, *C* and *B, C owl:imports D*; in this latter case, *D* is

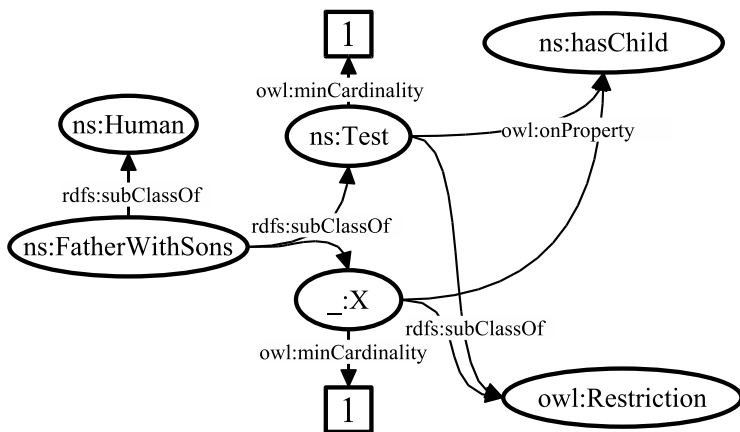


Fig. 1. Example of redundant Restrictions

imported twice, and this means that, unless the code used to resolve *owl:imports* handles the case of multiple imports, every blank node in D is duplicated in the resulting ontology. We will see an example of this situation in Section 5.

In order to accomplish this, we will show a *correct* algorithm (in the sense that it produces descriptions equivalent to the starting ones) without claiming for its *completeness* (it is not guaranteed to find the minimal equivalent description) called REDD (REDundancy Detection). This algorithm has been integrated in our RDF management system, named RDFCORE [2].

The remainder of this paper is organized as follows: Section 2 presents some necessary notions on RDF semantics, together with a brief survey of related work on RDF storage. In Section 3, the REDD algorithm is illustrated in detail; Section 4 describes RDFCORE, the system in which we implemented the REDD algorithm. Some experimental results are presented in Section 5.

2 Preliminaries

2.1 Basic Notions

We collect here some definitions and theorems that will be useful in the rest of the paper. Most of them have been taken from [6] and [4] and recalled here to make this paper as self-contained as possible. However, we assume the reader familiar with RDF Concepts and Syntax:⁴

Definition 1 (RDF-Graph). *A RDF-Graph is a set of RDF statements. Its nodes are URIs, literals or blank nodes (identifiable nodes with no intrinsic names⁵) representing subjects and objects of the statements. Its edges are labeled by URIs and represent the predicates of the triples.*

A small example can be found in Figure 2.

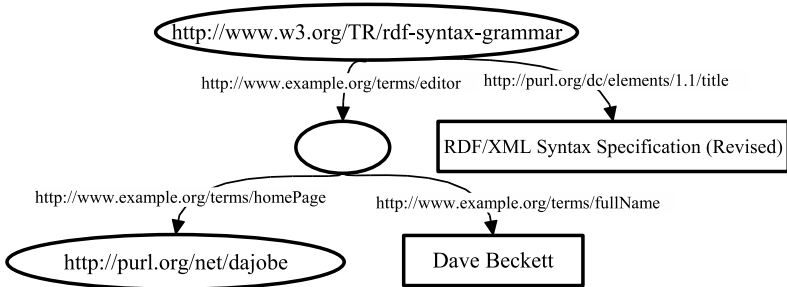


Fig. 2. A small example from <http://www.w3.org/TR/rdf-syntax-grammar/>

⁴ <http://www.w3.org/TR/rdf-concepts/>

⁵ <http://www.w3.org/TR/rdf-concepts/#section-URI-Vocabulary>

Definition 2 (Mapping). *Let N be a set of URIs, blank node names and literals. A mapping is a function $\mu : N \rightarrow N$ that changes a node name into another one.*

Definition 3 (Instance). *Let μ be a mapping from a set of blank nodes to some set of literals, blank nodes and URIs and G a graph, then any graph obtained from G by replacing some or all of the blank nodes N in G by $\mu(N)$ is an instance of G .*

Definition 4 (Lean Graph). *A RDF graph is lean if it has no instance which is a proper subgraph of the graph.*

The following results are proved in [6]:

Lemma 1 (Subgraph Lemma). *A graph entails all its subgraphs.*

Lemma 2 (Instance Lemma). *A graph is entailed by any of its instances.*

This means that every non-lean graph is equivalent to its *unique* lean subgraph [4]. Relying on these notions, in Section 3 we will present an algorithm that reduces non-lean graphs under certain conditions.

2.2 Related Work

Effective storage of RDF has always been bound to another key issue: Querying models. This was because no recommendation, at the time of writing, has been completed by W3C for RDF description querying (SPARQL⁶ is at the Working Draft stage of its evolution); thus, different solutions were developed, each one with its own query language and related optimizations. Some members of RDF Data Access Group issued a report⁷ in which six query engines were examined aiming to compare different expressive power of the underlying query languages. Actually, many different triple storage strategies are available. Among the systems implementing them, we remark the toolkit from HP Semantic Web Lab, called Jena [8, 9]. At the time of writing, Jena supports RDQL as query language, with support for SPARQL in a separate project, ARQ.⁸

Other interesting approaches to RDF data model optimization relies on properties of the RDF graph. One of them is described in [5], where the authors present an approach based on an intermediate layer between application data structures and the abstract triple syntax that uses hypergraphs. In this approach, theoretical results on graph theory can be used to minimize graphs and to cast application requests to well known graph problems, thus allowing to optimize different usage scenarios for RDF graphs.

⁶ <http://www.w3.org/2001/sw/DataAccess/rq23/>

⁷ <http://www.aifb.uni-karlsruhe.de/WBS/pha/rdp/rdf-query/rdfquery.pdf>

⁸ <http://cvs.sourceforge.net/viewcvs.py/jena/ARQ/>

3 Redundancy Detection

3.1 REDD Algorithm

Our redundancy detection algorithm is based on the notion of lean subgraph of a RDF graph. The lean subgraph is a subset of the RDF graph, and, as a consequence, is a subset of the set of statements of the original graph, having the property of being the smallest subgraph that is instance of the original graph. A pseudo code version of it can be found in Figure 3. The output of this algorithm has as a requirement the characteristic of expressing the same content of the original RDF graph (though in a more compact way). The output can be obtained from the original graph leaving untouched the ground part of the graph

ConnGraph{Set blanks, Model submodel, Map bToVarNames}

<pre> Set <u>FindRedundancies</u>(Model m){ Set redundancies Set connGraphs = CreateConnGraphs(m) FOR EACH graph in connGraphs{ Query q = CreateQuery(graph) Set redundancy = ExecuteQuery(m,q) ADD redundancy to redundancies } RETURN redundancies } Query <u>CreateQuery</u>(ConnGraph g){ Query q FOR EACH s in g.statements{ IF (s.subj is blank) AND (s.subj is not in g.bToVarNames){ create a variable name vn PUT(g.bToVarNames, s.subj, vn) ADD vn to q }ELSE{ vn = s.subj } IF (s.obj is blank) AND (s.obj is not in g.bToVarNames){ create a variable name o PUT(g.bToVarNames, s.obj, o) ADD o to q }ELSE{ o = s.obj } ADD (s, s.pred, o) condition to q } RETURN q } </pre>	<pre> Set <u>CreateConnGraphs</u>(Model m){ Set cg Map blanksToCg FOR EACH s in m{ IF s.subj is blank{ IF exists g in blanksToCg mapped by s.subj{ add s to g }ELSE{ create g for s.subj add s to g put g in cg PUT(blanksToCg, s.subj, g) } IF s.obj is blank{ add o to g.blanks PUT(blanksToCg, o, g) } } } RETURN cg } Set <u>ExecuteQuery</u>(Model m, Query q){ Bindings b = QUERYON(m, q) Set redundancy FOR EACH binding in b{ PUT binding.values in redundancy } RETURN redundancy } </pre>

Fig. 3. Pseudo-code description of the REDD algorithm

(i.e. every node that is not blank and any edge connecting non-blank nodes), and mapping from blank nodes to nodes already existing in the graph (blank nodes or URIs). The result is bound to be a subset of the original graph, apart from the identifiers of blank nodes.

Our algorithm searches for redundant blank nodes by looking at the graph and trying to find blank nodes that do not contain any additional information w.r.t. other nodes in the graph. Therefore, a blank node b is redundant if there is a node n that is involved in a set of statements that would be equal to the set of statements involving b if we replaced the occurrences of b with occurrences of n .

This is a special case of a more general view: taking as reference a subgraph built up of statements with blank nodes as subject and object, it is possible to search for a different subgraph of the model, isomorphic to the given subgraph (i.e. with the same properties between the nodes). On these two graphs, the algorithm can be applied considering the set of edges minus the edges already considered in the graph.

Our approach consists in finding a mapping from the original blank nodes of the graph to URI in the graph or to different blank nodes already in the graph (i.e. we do not introduce any new blank node). As an example, let us consider a simple graph containing two statements, say:

```
_:X ns:aGenericProperty ns:b
ns:a ns:aGenericProperty ns:b
```

we can determine that the graph is not lean by considering the mapping

$$_ : X \rightarrow ns : a$$

The result is a graph with a single statement

```
ns:a ns:aGenericProperty ns:b
```

which is lean by definition (being a graph with no blank nodes).

More formally, called:

- *ORIGIN* the original graph
- *RESULT* the new graph we are going to build
- X the anonymous node we want to map

we define:

Definition 5 (SUBMODEL). *All the statements in ORIGIN in which X is the subject.*

Definition 6 (SUPERMODEL). *All the statements in ORIGIN in which X is the object.*

We then can check every possible mapping from X to an URI or to a blank node identifier already occurring in *ORIGIN* for applicability to obtain an instance of *ORIGIN* which is both an instance and a proper subgraph (an approximation

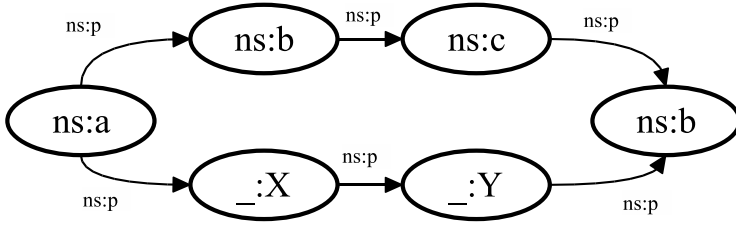


Fig. 4. Chained redundant blank nodes

of the lean subgraph) simply by checking that *SUBMODEL* of X is contained in *SUBMODEL* of the candidate node and *SUPERMODEL* of X is contained in *SUPERMODEL* of the candidate node. In fact, it can be easily proved that such a mapping does not produce any statement not contained in *ORIGIN*; *RESULT* then is a graph containing the same ground statements and a subset of the statements containing blank nodes. The missing statements are those containing the X node we just mapped. From the logical point of view, the information expressed by the graph is unchanged, since the mapping is equivalent to changing from:

$$\exists X.p(X, b) \text{ and } \exists a.p(a, b)$$

to

$$\exists a.p(a, b)$$

which are equivalent, not being stated that X is different from a . This mapping can be built for every redundant blank node in *ORIGIN*, but in some situations it is not guaranteed to find all redundancies. In fact, as in Figure 4, it is possible to have a chain of redundant blank nodes which cannot be spotted with a one-level visit of the RDF graph. In fact, in Figure 4, the two blank nodes represent the same structure as the two nodes labeled b and c . To find this redundancy, it is necessary to switch from a single node view to a multi node view.

For ease of reference, let us use an N-TRIPLE-like notation for the example in Figure 4:

```

ns:a ns:p _:X
ns:a ns:p ns:b
ns:b ns:p ns:c
ns:c ns:p ns:d
_:X ns:p _:Y
_:Y ns:p ns:d

```

considering the subgraph

```

_:X ns:p _:Y

```

named *BLANKS* for future reference, its structure can be described with a query like (using RDQL as query language, for the example)

```

SELECT ?a, ?b WHERE (?a, ns:p, ?b)

```

This query offers two results when executed against the model in the example: the first result is the mapping $?a \rightarrow \_ : X$ and $?b \rightarrow \_ : Y$, while the second is $?a \rightarrow ns : b$ and $?b \rightarrow ns : c$. Actually, the results are two graphs; checking every incoming edge and outgoing edge of the two graphs, we can determine if the graphs are equivalent, in analogy with the previous particular case in which the graph degenerates to a single blank node. This can be done adding conditions to the original query:

```
SELECT ?a, ?b WHERE (?a, ns:p, ?b)(ns:a ns:p ?a)(?b ns:p ns:d)
```

This can be done in a general way starting from the subgraph built considering every triple involving the *BLANKS* subgraph, and then generating the query with a condition for every triple and a variable for each blank node, keeping track of the blank node $\rightarrow$ variable name association. The resulting query, executed on the model, will surely produce one resulting graph (the subgraph used to generate it); any other result is a graph that respects the constraints we imposed on the single node case: on any node, the set of incoming edges includes the set of incoming edges of the corresponding node in the *BLANKS* graph (the corresponding node is the node that grounds the same variable), excluding the edges already included in the graph (i.e. where both subject and object are variables in the query).

For ease of future reference, we give the following definition:

Definition 7 (CONNECTED SUBGRAPH). *A connected subgraph of a graph G is a subgraph of G containing at least one blank node (as subject or object); if more than a blank node is contained in the graph, then the blank nodes make up a chain (i.e. it is possible to navigate from a blank node to another following the predicates). The connected subgraph is made up of all the triples in G that involve at least one of these blank nodes.*

As an example, in Figure 4 there is one chain of blank nodes; the corresponding connected graph is:

```
ns:a ns:p _:X
_:X ns:p _:Y
_:Y ns:p ns:d
```

This algorithm has been implemented in two steps: first, the special case in which we consider only single blank nodes (i.e. no chain redundancies are detected) has been implemented both as a Java class (built on top of the Jena API) and directly in the storage layer of RDFCORE with stored procedures in the Oracle DB. The second step, i.e. the implementation of both single nodes and chains detection, has been completed as a Java class (again using the Jena API), and at the time of writing we are implementing it in the storage layer.

3.2 REDD Computational Complexity

In this subsection we will shortly carry out an *a priori* evaluation of computational cost required by REDD algorithm. We will keep as reference the pseudo

code version of REDD in Figure 3. Obviously, the actual implementations working both in memory and natively on the storage layers (see Section 4) underwent some optimizations, not shown in the pseudo code for sake of brevity; hence calculations in this section represent only an upper theoretical limit for the computational cost of REDD. In section 5, the reader can find some empirical evaluations.

We start defining some metrics on RDF descriptions on which, as shown below, REDD complexity depends.

Definition 8 (RDF Description metrics). *Be G a RDF description and n a generic node in G then*

- N_T^G stands for the number of RDF triples within G
- $N_{T_B}^G$ stands for the number of RDF triples within G containing at least a blank node
- $N_{T_{NB}}^G$ stands for the number of RDF triples within G with no blank nodes (it is equal to $N_T^G - N_{T_B}^G$)
- $\#_{CG}^G$ stands for the number of connected subgraphs with blank nodes within G

Referring to Figure 3, complexity of *FindRedundancies* C_{FR} is:

$$C_{FR} = C_{CCG} + \#_{CG}^G * C_{CQ} + \#_{CG}^G * C_{EQ} \quad (1)$$

where

- C_{CCG} is the complexity of the *CreateConnGraphs* operation, which is $O(N_T^G)$ (linear in the size of the model); more in detail, the main cycle of *CreateConnGraphs* depends on N_T^G , while each operation executed in the cycle does not depend on N_T^G (depending on the implementation, map and set operations can vary their complexity; assuming a hash implementation for both of them, every operation can be considered $O(1)$). On the other hand, the number of mappings in the *blanksToCG* map depends on the degree of connectedness in the graph: at worse, there will be no more than N_T^G mappings (because the number of mappings cannot exceed the number of triples in the graph), in the case that every statement has at least a blank node as subject or object;
- C_{CQ} is the complexity of the *CreateQuery* operation, which depends on the number of triples in the connected subgraph it is operated onto; since the connected subgraphs are disjoint set (if two connected subgraphs overlap, i.e. they have some common blank node and in consequence some common triple, they are actually one connected subgraph by definition, and they are built in this way), the whole group $\#_{CG}^G * C_{CQ}$ has complexity $O(N_{T_B}^G)$;
- C_{EQ} is the complexity of the *ExecuteQuery* operation, which depends on the *QUERYON* operator and on the number of results the query finds in the model. In the worst case (very unlikely to happen), the number of results can be equal to the number of resources in the model, which is at worst

$3 * N_T^G$ (again, very unlikely to happen - in particular, since in the query the predicate URI is never a variable, if the second situation is the case then the number of results will be exactly one, and no redundancies will be possible). Hence, this portion is $O(N_T^G)$. The *QUERYON* operator complexity depends on the query facility, that in the actual implementation relies on Jena RDQL support. An upper limit for the complexity of this operation can be calculated considering each condition in the query and verifying them one by one against the model. There is a condition for each statement in the connected graph, and each one of them requires (in an implementation with no optimizations) at most N_T^G checks on the model; under these assumptions (which are surely an underestimation of the real performances), the whole group $\#_{CG}^G * C_{EQ}$ has complexity $O(\#_{CG}^G * N_T^G + N_T^{G^2})$

As a result, C_{FR} has complexity $O(N_T^{G^2})$; in particular, the quadratic complexity depends on the query phase of the algorithm, since the other two main sections are $O(N_T^G)$, and it comes from a deliberate overestimation of the query performances. As an example, in an hypothetical implementation a simple indexing on statement predicates P reduces the complexity of the search from $O(N_T^G)$ to $O(\#_P)$. In real models, it can reasonably be assumed that $\#_P \ll N_T^G$.

In the next sections we will briefly present the RDFCORE system, where REDD has been implemented, and, in section 5, we will show some empirical results for the algorithm.

4 The RDFCore Component

The RDFCORE component, presented in [2, 7], is based on two classes, *DescriptionManager* and *TripleManager*, and an interface, *RDFEngineInterface*.

RDFEngineInterface is an interface that enables the managers to abstract from the physical persistence details. Thanks to this design, based on the well known *Strategy* pattern [3], the system can use one or more persistence implementations (with different performance or scalability tradeoffs) whenever needed, on the basis of the needs of the external applications, without the programmer having to bother about different APIs. Currently, there are four implementations of *RDFEngineInterface*, two based on the well-known Jena Semantic Web Toolkit, one with MySQL RDBMS⁹ and another with SQL Server¹⁰ as persistent storage; a third implementation is based on RDF/XML files. The last implementation, called *RDFEngineREDD*, is the one in which we embedded the REDD algorithm natively in the storage level. It uses Oracle¹¹ as RDBMS. The database has been chosen because of the availability of stored procedures and

⁹ <http://dev.mysql.com/doc/mysql/en/index.html>

¹⁰ <http://www.microsoft.com/sql/>

¹¹ Oracle 9.2.0.1.0 (Oracle 9i Release 2) <http://otn.oracle.com/documentation/oracle9i.html>

the ability to execute Java code directly on the database, avoiding the overhead of data transfer that would have arisen using different solutions.

Currently RDFCORE is a central component within the core infrastructure of the software architecture that will result out of the 6th Framework Project VIKEF (Virtual Information and Knowledge Framework Priority 2.3.1.7. Semantic Based Knowledge Systems Contract no.: 507173).

5 Experimental Results

To evaluate the scalability of our implementation of the REDD algorithm in the *RDFEngineREDD* implementation of *RDFEngineInterface*, we built a set of Models to check some situations inspired by real models; the results are in Table 3. The models come from different sources: the first two, *lean* and *nolean*, are from [6], where they are presented as basic examples of lean and non-lean graphs. *nolean2B* is a slight variation of *nolean*, with two redundant blank nodes. *cycleTest* is used to check the behavior of the algorithm when dealing with complex cyclic situations in graphs, while *blankChain* contains a chain of redundant blank nodes like in the Figure 4. *restriction* contains a redundant restriction class definition (as in Figure 1) together with a redundant union class definition (in OWL); the last Model, *daml*, contains a sketch of a DAML ontology, with some class definitions including both Restriction, Union and Intersection types. For each model, we recorded the number of statements, the number of blank nodes present in the graph, the elapsed time to insert the models in our persistence (in milliseconds), the elapsed time to execute REDD (in milliseconds) and the number of removable blanks in the graph. Since the size of these models is way too small to evaluate scalability on model size and complexity, we kept these test cases as correctness checks while developing the algorithm, and then created a parametric method to generate bigger models with known structure, in order to scale the size and complexity without having to check the correctness of the results (which can be a time consuming task for models with more than some tens of nodes). The parameters we used are: the number of blank nodes in a graph, the number of incoming/outgoing edges for each node, and the number of redundancies for each blank node (i.e. a blank node can be found redundant with one or more nodes in the graph). The test models were built scaling on the three parameters independently (showed in Table 1); in the last section, both the number of blank nodes and the number of redundancies per node is augmented.

These tests were performed on the database implementation, that, as said earlier in the paper, still does not handle the blank node chains.

In order to give a preliminary evaluation of the complete algorithm, we used some models built from a real ontology, as said in Section 1. The ontology is the BM Ontology¹², that aims to describe the domain of business process description. In this ontology, we tried to artificially increase the number of blank nodes and of blank nodes chain.

¹² <http://www.bpiresearch.com>

Table 1. Fake models scaling on ingoing / outgoing edges, blank node number and redundancy number

Model id	Triple #	Blank node #	Storing time (ms)	REDD (ms)	Redundancies #	Removable blanks #	Ingoing/outgoing edges
0	120	1	1469	62	5	1	10
1	240	1	2469	94	5	1	20
2	360	1	3438	141	5	1	30
3	480	1	4515	188	5	1	40
4	600	1	5266	234	5	1	50
5	720	1	6328	297	5	1	60
6	840	1	7109	360	5	1	70
7	960	1	8172	437	5	1	80
8	1080	1	9203	594	5	1	90
9	1200	1	11016	625	5	1	100
10	200	5	1953	78	1	5	10
11	400	10	3766	125	1	10	10
12	600	15	5406	250	1	15	10
13	800	20	7203	219	1	20	10
14	1000	25	10000	281	1	25	10
15	1200	30	10860	375	1	30	10
16	1400	35	12828	407	1	35	10
17	1600	40	14844	469	1	40	10
18	1800	45	15969	563	1	45	10
19	2000	50	18047	750	1	50	10
20	120	1	2235	453	5	5	10
21	220	1	2235	93	10	10	10
22	320	1	3188	156	15	15	10
23	420	1	3828	188	20	20	10
24	520	1	4485	234	25	25	10
25	620	1	5047	266	30	30	10
26	720	1	5813	297	35	35	10
27	820	1	6907	546	40	40	10
28	920	1	7360	406	45	45	10
29	1020	1	8188	437	50	50	10
30	600	5	4906	234	5	5	10
31	2200	10	18328	922	10	10	10
32	4800	15	39141	2187	15	15	10
33	8400	20	69578	4203	20	20	10
34	13000	25	118031	6078	25	25	10
35	18600	30	171563	10031	30	30	10

Our use of the BM Ontology was as follows: we loaded the ontology in a Jena OntModel, with no reasoning in order to use only the original triples, and then wrote out the complete model, obtaining a RDF model containing the BM Ontology and the import closure. Then, we reloaded this new model (that we will call BMO1) in an OntModel. The resolution of *owl:imports* directive in this admittedly pathological model produces a new inferred model in which every blank node and blank node chain is duplicated, and this produces redundancies that REDD can discover (model BMO2). We repeated the procedure and obtained the models BMO3 and BMO4. The results are shown in Table 2.

As can be seen in Table 1, the insertion of new descriptions in RDFCORE roughly scales linearly with the size of the descriptions. The performance overhead due to index updating, however, increases when the number of triples in a description increase, so the total complexity is more than linear. The heavy indexing, on the other side, enables us to obtain very good results when running

Table 2. Models with blank node chains

Model id	Triple #	Blank node #	Redundant triples #	REDD (ms)	Chain #
BMO1	12267	1416	16	3294	804
BMO2	16487	2832	8440	5258	1608
BMO3	20707	4248	12660	18146	2412
BMO4	24927	5664	16880	102648	3216

Table 3. Some real-world models tests

Model id	Triple #	Blank node #	Storing time (ms)	REDD (ms)	Removable blanks #
lean	2	1	140	32	0
nolean	2	1	62	31	1
nolean2B	3	2	46	47	2
blankChain	7	2	94	31	0
cycleTest	15	2	204	31	1
restriction	35	17	500	93	7
daml	38	33	718	282	16

the REDD algorithm on the data. About the real size reduction of the model after the removal of the blank nodes (which means the removal of every triple referring to these nodes), it is not possible to draw general conclusions since the number of triples strongly depends on the graph; the only reasonable lower limit is two triples per blank node, since it is quite unusual to have a dangling blank node or a graph rooted in a blank node, and in these cases it is unlikely that the nodes are redundant (e.g. `ns:a ns:aProperty _:X` means that *ns:a* has a filler for the role *ns:aProperty*, but nothing else is known about this filler; adding another statement, `ns:a ns:aProperty _:Y`, would assert the same thing; unless stating that `_:X` is different from `_:Y`, REDD signals the nodes as redundant).

About the implementation running in memory, the test data shows an interesting behavior: while BMO1 only contains 16 redundant triples, BMO2 contains 8440 redundant triples. What happens here is that the BMO1 model contains 4 redundant restrictions (that are anonymous cardinality restrictions similar to the one represented in Figure 1), each built up of 4 triples; the other 1412 blank nodes are arranged in 800 chains (with different chain length), usually RDF lists, that are not redundant with any structure in BMO1. In BMO2, however, the duplication of these structures produces 1608 blank node chains, and each one of them is redundant with at least one structure. Same explanation for the further increase in BMO3 and BMO4.

From this information, it is possible to infer what would be the results of querying a reduced model: in fact, BMO1 is only 12 triples bigger than the smallest model that REDD can produce. Since every blank node chain produces a RDQL query to be executed on the model, from the data it is possible to deduce that the time required for a query on BMO4 is bigger than the time required for a query in BMO1 (from about 4 ms in BMO1 to more than 26 ms in BMO4). This huge difference between query performance can be partially attributed to lack of optimization in the current algorithm implementation (e.g., two redundant blank

node chains produce two queries, but the queries are equal; this is recognized only in some cases by our implementation), but it is an empirical confirmation of our initial intuitive claim that queries on a smaller model are faster than queries on a larger model.

Our aim in the ongoing work (i.e. pushing down into the persistence layer the chain redundancy detection) is to match the performances of the first version of the algorithm. Moreover, we plan extensions to the algorithm applications, e.g. recognition of Alt and Bag structures in order to be able to detect duplications. Another extension (based on OWL semantics) is the recognition of the use of Lists in the declaration of union and intersection classes; while differently ordered lists are different at the RDF level, they express the same meaning at the OWL level, and this should be detected as redundancy. Also, it is necessary to establish ordering criteria when choosing the blank nodes to be removed from the graph: in fact, detecting a redundancy corresponds to finding of set inclusion relationships between *SUPERGRAPHS* and *SUBGRAPHS*; the choice can be made freely only when *SUPERGRAPHS* and *SUBGRAPHS* are equal, while in other cases an ordering criterion must be used.

6 Conclusions

In this paper we started from the consideration that SW is based on a particular language for metadata description, RDF, whose semantics has been recently thoroughly investigated by W3C and other researchers. This initial effort produced some valuable results in terms of theoretical foundations for entailment in RDF. We examined, in particular, results concerning blank node semantics and their effects on the problem of compacting RDF graphs. We presented a correct algorithm for spotting out redundant blank nodes in RDF graphs and we provided a pseudo code implementation. We discussed its complexity proving it is tractable (polynomial). Afterward we presented its actual prototypical implementation within our RDF management system (RDFCORE). From empirical evaluation we found out that these initial results are encouraging (being it a prototype). Furthermore, redundancies can be also referred to a vocabulary. In fact this work did not take into account RDFS (and its derivatives) semantics that can be deeply exploited for compacting descriptions.

Acknowledgments

This research was partially funded by the European Commission under the 6th Framework Programme IST Integrated Project VIKEF - Virtual Information and Knowledge Environment Framework (Contract no. 507173, Priority 2.3.1.7 Semantic-based Knowledge Systems; more information at <http://www.vikef.net>), and under the DELOS 2 Network of Excellence on Digital Libraries started on January 1, 2004 Priority IST-2002-2.3.1.12 Technology-enhanced Learning and Access to Cultural Heritage - Contract no.: G038-507618 (<http://www.delos.info/>).

References

1. Berners-Lee, T.: Semantic Web Road map (1998) <http://www.w3.org/DesignIssues/Semantic.html>.
2. Esposito, F., Iannone, L., Palmisano, I., Semeraro, G.: RDF Core: a Component for Effective Management of RDF Models. In Cruz, I.F., Kashyap, V., Decker, S., Eckstein, R., eds.: Proceedings of SWDB'03, The First International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003. (2003)
3. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. 1st edn. Addison-Wesley (1995)
4. Gutiérrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of Semantic Web Databases. In: Proceedings of ACM Symposium on Principles of Database Systems (PODS) Paris, France, June 2004. (2004)
5. Hayes, J., Gutiérrez, C.: Bipartite graphs as intermediate model for rdf. In: International Semantic Web Conference. (2004) 47–61
6. Hayes, P.: RDF semantics (2004) W3C Recommendation 10 February 2004 <http://www.w3.org/TR/rdf-mt/>.
7. Iannone, L., Palmisano, I., Redavid, D.: Optimizing RDF storage removing redundancies: an algorithm. In Ali, M., Esposito, F., eds.: Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Bari, Italy, June 22-25 2005. Lecture Notes in Artificial Intelligence, Springer (2005) (to appear).
8. McBride, B.: JENA: A Semantic Web toolkit. *IEEE Internet Computing* **6** (2002) 55–59
9. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF storage and retrieval in jena2. In Cruz, I.F., Kashyap, V., Decker, S., Eckstein, R., eds.: Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003. (2003) 131–150

OWL-Eu: Adding Customised Datatypes into OWL

Jeff Z. Pan and Ian Horrocks

School of Computer Science, University of Manchester, UK
{pan, horrocks}@cs.man.ac.uk

Abstract. Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome. Accordingly, the Semantic Web Best Practices and Development Working Group sets up a task force to address this issue. This paper makes the following two contributions: (i) it provides a brief summary of OWL-related datatype formalisms, and (ii) it provides a decidable extension of OWL DL, called OWL-Eu, that supports customised datatypes.

1 Introduction

The OWL Web Ontology Language [3] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support [16, 17] is one of the most useful features OWL is expected to provide, and has brought extensive discussions in the RDF-Logic mailing list [18] and Semantic Web Best Practices mailing list [20]. Although OWL adds considerable expressive power to the Semantic Web, the OWL datatype formalism (or simply *OWL datatyping*) is much too weak for many applications; in particular, OWL datatyping does not provide a general framework for customised datatypes,¹ such as XML Schema derived datatypes.

It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome [19], as it is often necessary to enable users to define their own datatypes and datatype predicates for their ontologies and applications. For instance, when using a computer sales ontology, a user may need to describe a PC with memory size greater than or equal to 512Mb, unit price less than 700 pounds and delivery date earlier than 15/03/2004. In this context, ‘greater than or equal to 512’, ‘less than 700’ and ‘earlier than 15/12/2004’ can be seen as customised datatypes, with the base datatypes being integer, integer and date, respectively.

After reviewing the design of OWL, and the needs of various applications and (potential) users, the following requirements for an extension to OWL DL have been identified:

1. It should provide customised datatypes; therefore, it should be based on a datatype formalism which is compatible with OWL datatyping, provides facilities to con-

¹ A widely discussed example would be the ‘BigWheel’ example discussed in, e.g., <http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0061.html>.

struct customised datatypes and, most importantly, guarantees the computerability of the kinds of customised datatypes it supports.

2. It should overcome other important limitations of OWL datatyping, such as the absence of negated datatypes and the un-intuitive semantics for unsupported datatypes (which will be further explained in Section 4).
3. It should satisfy the *small extension requirement*, which is two folded: on the one hand, the extension should be a substantial and necessary extension that overcomes the above mentioned limitations of OWL datatyping; on the other hand, following W3C's 'one *small* step at a time' strategy, it should only be as large as is necessary in order to satisfy the requirements.
4. It should be a decidable extension of OWL DL.

This paper makes two main contributions. Firstly, it provides an overview of relevant (to OWL) datatype formalisms, namely those of XML, RDF and OWL itself. Secondly, and most importantly, it presents an extension of OWL DL,² called OWL-Eu (OWL with unary datatype Expressions), which satisfies the above requirements.

The rest of the paper is organised as follows. Section 2 briefly introduces the OWL Web Ontology Language. Section 3 describes OWL-related datatype formalisms. Section 4 summarises the limitations of OWL datatyping. Section 5 presents the OWL-Eu language, showing how it satisfies the above four requirements. Section 6 describes some related works, and Section 7 concludes the paper and suggests some future works.

2 An Overview of OWL

OWL is a standard (W3C recommendation) for expressing ontologies in the Semantic Web. The OWL language facilitates greater machine understandability of Web resources than that supported by RDFS by providing additional constructors for building class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. The OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. *OWL Lite* and *OWL DL* are, like DAML+OIL, basically very expressive Description Logics (DLs); they are almost³ equivalent to the *SHIF*($\mathbf{D}^+$) and *SHOIN*($\mathbf{D}^+$) DLs. *OWL Full* provides the same set of constructors as OWL DL, but allows them to be used in an unconstrained way (in the style of RDF). It is easy to show that OWL Full is undecidable, because it does not impose restrictions on the use of transitive properties [10]; therefore, when we mention OWL in this paper, we usually mean OWL DL.

Let $\mathbf{C}$, $\mathbf{R}_I$, $\mathbf{R}_D$ and $\mathbf{I}$ be the sets of URIs that can be used to denote classes, *individual-valued* properties, *data-valued* properties and individuals respectively. An OWL DL *interpretation* is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta_D^{\mathcal{I}}, \cdot^{\mathcal{I}}, \cdot^{\mathcal{P}})$ where the individual domain $\Delta^{\mathcal{I}}$ is a nonempty set of individuals, the datatype domain $\Delta_D^{\mathcal{I}}$ is a nonempty set of data values, $\cdot^{\mathcal{I}}$ is an individual interpretation function that maps

² cf. Section 2 for the differences of three sub-languages of OWL.

³ They also provide annotation properties, which Description Logics don't.

Table 1. OWL *individual-valued* property descriptions

Abstract Syntax	DL Syntax	Semantics
ObjectProperty(R)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
ObjectProperty(S inverseOf(R))	R^{-}	$(R^{-})^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Table 2. OWL class descriptions

Abstract Syntax	DL Syntax	Semantics
Class(A)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Class(owl:Thing)	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Class(owl:Nothing)	$\perp$	$\perp^{\mathcal{I}} = \emptyset$
intersectionOf($C_1, C_2, \dots$)	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
unionOf($C_1, C_2, \dots$)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
complementOf(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
oneOf($\mathbf{o}_1, \mathbf{o}_2, \dots$)	$\{\mathbf{o}_1\} \sqcup \{\mathbf{o}_2\}$	$(\{\mathbf{o}_1\} \sqcup \{\mathbf{o}_2\})^{\mathcal{I}} = \{\mathbf{o}_1^{\mathcal{I}}, \mathbf{o}_2^{\mathcal{I}}\}$
restriction(R someValuesFrom(C))	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restriction(R allValuesFrom(C))	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
restriction(R hasValue($\mathbf{o}$))	$\exists R.\{\mathbf{o}\}$	$(\exists R.\{\mathbf{o}\})^{\mathcal{I}} = \{x \mid \langle x, \mathbf{o}^{\mathcal{I}} \rangle \in R^{\mathcal{I}}\}$
restriction(R minCardinality(m))	$\geq mR$	$(\geq mR)^{\mathcal{I}} = \{x \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \geq m\}$
restriction(R maxCardinality(m))	$\leq mR$	$(\leq mR)^{\mathcal{I}} = \{x \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \leq m\}$
restriction(T someValuesFrom(u))	$\exists T.u$	$(\exists T.u)^{\mathcal{I}} = \{x \mid \exists t. \langle x, t \rangle \in T^{\mathcal{I}} \wedge t \in u^{\mathbf{D}}\}$
restriction(T allValuesFrom(u))	$\forall T.u$	$(\forall T.u)^{\mathcal{I}} = \{x \mid \forall t. \langle x, t \rangle \in T^{\mathcal{I}} \rightarrow t \in u^{\mathbf{D}}\}$
restriction(T hasValue(w))	$\exists T.\{w\}$	$(\exists T.\{w\})^{\mathcal{I}} = \{x \mid \langle x, w^{\mathbf{D}} \rangle \in T^{\mathcal{I}}\}$
restriction(T minCardinality(m))	$\geq mT$	$(\geq mT)^{\mathcal{I}} = \{x \mid \#\{t \mid \langle x, t \rangle \in T^{\mathcal{I}}\} \geq m\}$
restriction(T maxCardinality(m))	$\leq mT$	$(\leq mT)^{\mathcal{I}} = \{x \mid \#\{t \mid \langle x, t \rangle \in T^{\mathcal{I}}\} \leq m\}$

- each individual name $\mathbf{a} \in \mathbf{I}$ to an element $\mathbf{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- each concept name $\text{CN} \in \mathbf{C}$ to a subset $\text{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each *individual-valued* property name $RN \in \mathbf{R}_{\mathbf{I}}$ to a binary relation $RN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and
- each *data-valued* property name $TN \in \mathbf{R}_{\mathbf{D}}$ to a binary relation $TN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}$,

and $\cdot^{\mathbf{D}}$ is a datatype interpretation function. More details of $\Delta_{\mathbf{D}}$ and $\cdot^{\mathbf{D}}$ will be presented in Section 3.3.

The individual interpretation function can be extended to give semantics to class and *individual-valued* property descriptions shown in Tables 1 and 2, where $A \in \mathbf{C}$ is a concept URIref, $C, C_1, \dots, C_n$ are concept descriptions, $R \in \mathbf{R}_{\mathbf{I}}$ is an *individual-valued* property URIref, $R_1, \dots, R_n$ are *individual-valued* property descriptions and $\mathbf{o}, \mathbf{o}_1, \mathbf{o}_2 \in \mathbf{I}$ are individual URIrefs, u is a data range (cf. Definition 8), $T \in \mathbf{R}_{\mathbf{D}}$ is a *data-valued* property and $\#$ denotes cardinality.

An OWL DL ontology can be seen as a DL knowledge base [11], which consists of a set of *axioms*, including class axioms, property axioms and individual axioms.⁴ Table 3 presents the abstract syntax, DL syntax and semantics of OWL axioms.

⁴ Individual axioms are also called *facts*.

Table 3. OWL axioms

Abstract Syntax	DL Syntax	Semantics
Class(A partial $C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$	$A^I \subseteq C_1^I \cap \dots \cap C_n^I$
Class(A complete $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$	$A^I = C_1^I \cap \dots \cap C_n^I$
EnumeratedClass(A $\circ_1 \dots \circ_n$)	$A \equiv \{\circ_1\} \sqcup \dots \sqcup \{\circ_n\}$	$A^I = \{\circ_1^I, \dots, \circ_n^I\}$
SubClassOf(C_1, C_2)	$C_1 \sqsubseteq C_2$	$C_1^I \subseteq C_2^I$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$	$C_1^I = \dots = C_n^I$
DisjointClasses($C_1 \dots C_n$)	$C_i \sqsubseteq \neg C_j,$ ($1 \leq i < j \leq n$)	$C_1^I \cap C_n^I = \emptyset,$ ($1 \leq i < j \leq n$)
SubPropertyOf(R_1, R_2)	$R_1 \sqsubseteq R_2$	$R_1^I \subseteq R_2^I$
EquivalentProperties($R_1 \dots R_n$)	$R_1 \equiv \dots \equiv R_n$	$R_1^I = \dots = R_n^I$
ObjectProperty(R super(R_1) ... super(R_n))	$R \sqsubseteq R_i$	$R^I \subseteq R_i^I$
domain(C_1) ... domain(C_k)	$\geq 1 R \sqsubseteq C_i$	$R^I \subseteq C_i^I \times \Delta^I$
range(C_1) ... range(C_h)	$\top \sqsubseteq \forall R.C_i$	$R^I \subseteq \Delta^I \times C_i^I$
[Symmetric]	$R \equiv R^-$	$R^I = (R^-)^I$
[Functional]	Func(R)	$\{\langle x, y \rangle \mid \#\{y.\langle x, y \rangle \in R^I\} \leq 1\}$
[InverseFunctional]	Func(R^-)	$\{\langle x, y \rangle \mid \#\{y.\langle x, y \rangle \in (R^-)^I\} \leq 1\}$
[Transitive]	Trans(R)	$R^I = (R^I)^+$
AnnotationProperty(R)		
Individual($\circ$ type(C_1) ... type(C_n))	$\circ : C_i, 1 \leq i \leq n$	$\circ^I \in C_i^I, 1 \leq i \leq n$
value($R_1, \circ_1$) ... value($R_n, \circ_n$)	$\langle \circ, \circ_i \rangle : R_i, 1 \leq i \leq n$	$\langle \circ^I, \circ_i^I \rangle \in R_i^I, 1 \leq i \leq n$
SameIndividual($\circ_1 \dots \circ_n$)	$\circ_1 = \dots = \circ_n$	$\circ_1^I = \dots = \circ_n^I$
DifferentIndividuals($\circ_1 \dots \circ_n$)	$\circ_i \neq \circ_j, 1 \leq i < j \leq n$	$\circ_i^I \neq \circ_j^I, 1 \leq i < j \leq n$

3 Datatype Formalisms

In this section we will provide a brief overview of the XML, RDF and OWL datatype formalisms.

3.1 XML Schema Datatypes

W3C XML Schema Part 2 [4] defines facilities for defining simple types to be used in XML Schema as well as other XML specifications.

Definition 1. An XML Schema simple type d is characterised by a value space, $V(d)$, which is a non-empty set, a lexical space, $L(d)$, which is a non-empty set of Unicode [6] strings, and a set of facets, $F(d)$, each of which characterizes a value space along independent axes or dimensions. $\diamond$

XML Schema simple types are divided into disjoint built-in simple types and derived simple types. Derived datatypes can be defined by derivation from primitive or existing derived datatypes by the following three means:

- Derivation by *restriction*, i.e., by using facets on an existing type, so as to limit the number of possible values of the derived type.
- Derivation by *union*, i.e., to allow values from a list of simple types.
- Derivation by *list*, i.e., to define the list type of an existing simple type.

Example 1. The following is the definition of a derived simple type (of the base datatype xsd:integer) which restricts values to integers greater than or equal to 0 and less than 150, using the facets minInclusive and maxExclusive.

```

<simpleType name = "humanAge">
  <restriction base = "xsd:integer">
    <minInclusive value = "0"/>
    <maxExclusive value = "150"/>
  </restriction>
</simpleType>

```

◇

3.2 Datatypes in RDF

According to [8], RDF allows the use of datatypes defined by any external type systems, e.g., the XML Schema type system, which conform to the following specification.

Definition 2. A datatype d is characterised by a lexical space, $L(d)$, which is a non-empty set of Unicode strings; a value space, $V(d)$, which is a non-empty set, and a total mapping $L2V(d)$ from the lexical space to the value space. ◇

This specification allows the use of non-list XML Schema built-in simple types as datatypes in RDF, although some built-in XML Schema datatypes are problematic because they do not fit the RDF datatype model.⁵ Furthermore, comparisons between Definition 1 and 2 show that RDF does not take XML Schema facets into account, which are essential to define derived simple types.

In RDF, data values are represented by literals.

Definition 3. All literals have a lexical form being a Unicode string. Typed literals are of the form “ s ” ^{u} , where s is a Unicode string, called the lexical form of the typed literal, and u is a datatype URI reference. Plain literals have a lexical form and optionally a language tag as defined by [1], normalised to lowercase. ◇

Example 2. Boolean is a datatype with value space $\{true, false\}$, lexical space $\{\text{“true”}, \text{“false”}, \text{“1”}, \text{“0”}\}$ and lexical-to-value mapping $\{\text{“true”}_i \rightarrow true, \text{“false”}_i \rightarrow false, \text{“1”}_i \rightarrow true, \text{“0”}_i \rightarrow false\}$. “true”^{xsd:boolean} is a typed literal, while “true” is a plain literal. ◇

The associations between datatype URI references (e.g., xsd:boolean) and datatypes (e.g., boolean) can be provided by datatype maps defined as follows.

Definition 4. A datatype map M_d is a partial mapping from datatype URI references to datatypes. ◇

Note that XML Schema derived simple types are *not* RDF datatypes because XML Schema provides no mechanism for using URI references to refer to derived simple types.

The semantics of RDF datatypes are defined in terms of M_d -interpretations, which extend RDF-interpretations and RDFS-interpretations (cf. RDF Semantics [8]) with extra conditions for datatypes.

⁵ Readers are referred to [8] for more details.

Definition 5. Given a datatype map $\mathbf{M}_d$, an RDFS $\mathbf{M}_d$ -interpretation I of a vocabulary $\mathbf{V}$ (a set of URIs and plain literals) is any RDFS-interpretation of $\mathbf{V} \cup \{u \mid \exists d. \langle u, d \rangle \in \mathbf{M}_d\}$ which introduces

- a non-empty set $\mathbf{IR}$ of resources, called the domain (or universe) of I ,
- a set $\mathbf{IP}$ (the RDF-interpretation requires $\mathbf{IP}$ to be a sub-set of $\mathbf{IR}$) called the set of properties in I ,
- a set $\mathbf{IC}$ (the RDFS-interpretation requires $\mathbf{IC}$ to be a sub-set of $\mathbf{IR}$) called the set of classes in I , and
- a distinguished subset $\mathbf{LV}$ of $\mathbf{IR}$, called the set of literal values, which contains all the plain literals in $\mathbf{V}$,
- a mapping IS from URIs in $\mathbf{V}$ to $\mathbf{IR}$,
- a mapping $IEXT$, called the extension function, from $\mathbf{IP}$ to the powerset of $\mathbf{IR} \times \mathbf{IR}$,
- a mapping $ICEXT$, called the class extension function, from $\mathbf{IC}$ to the set of subsets of $\mathbf{IR}$,
- a mapping IL from typed literals in $\mathbf{V}$ into $\mathbf{IR}$,

and satisfies the following extra conditions:

1. $\mathbf{LV} = ICEXT(IS(rdfs:Literal))$,
2. for each plain literal pl , $IL(pl) = pl$,
3. for each pair $\langle u, d \rangle \in \mathbf{M}_d$,
 - (a) $ICEXT(d) = V(d) \subseteq \mathbf{LV}$,
 - (b) there exist $d \in \mathbf{IR}$ s.t. $IS(u) = d$,
 - (c) $IS(u) \in ICEXT(IS(rdfs:Datatype))$,
 - (d) for “ s ” $\wedge$ “ u' ” $\in \mathbf{V}$, $IS(u') = d$, if $s \in L(d)$, then $IL(\text{“}s\text{”}\wedge u') = L2S(d)(s)$, otherwise, $IL(\text{“}s\text{”}\wedge u') \in \mathbf{IR} \setminus \mathbf{LV}$,
4. if $d \in ICEXT(IS(rdfs:Datatype))$, then $\langle d, IS(rdfs:Literal) \rangle \in IEXT(rdfs:subClassOf)$. $\diamond$

According to Definition 5, $\mathbf{LV}$ is a subset of $\mathbf{IR}$, i.e., literal values are resources. Condition 1 ensures that the class extension of $rdfs:Literal$ is $\mathbf{LV}$. Condition 2 ensures that the plain literals are interpreted as themselves. Condition 3a asserts that RDF(S) datatypes are classes (because datatypes are interpreted using the class extension function $ICEXT$), condition 3b ensures that there is a resource d for datatype d in $\mathbf{M}_d$, and condition 3c ensures that the class $rdfs:Datatype$ contains the datatypes used in any satisfying $\mathbf{M}_d$ -interpretation. Condition 3d explains why the range of IL is $\mathbf{IR}$ rather than $\mathbf{LV}$ (because, for “ s ” $\wedge$ “ u ”, if $s \notin L(IS(u))$, then $IL(\text{“}s\text{”}\wedge u) \notin \mathbf{LV}$); note that this is different from OWL datatypes (cf. Definition 9). Condition 4 requires that RDF(S) datatypes are *sub-classes* of $rdfs:Literal$.

3.3 Datatypes in OWL

OWL datatypeing adopts the RDF specification of datatypes and data values. It extends RDF datatypeing by (i) allowing different OWL reasoners to provide different supported datatypes, and (ii) introducing the use of so called enumerated datatypes.

Definition 6. Given a datatype map $\mathbf{M}_d$, a datatype URI reference u is called a supported datatype URI reference w.r.t. $\mathbf{M}_d$ if there exists a datatype d s.t. $\mathbf{M}_d(u) = d$ (in this case, d is called a supported datatype w.r.t. $\mathbf{M}_d$); otherwise, u is called an unsupported datatype URI reference w.r.t. $\mathbf{M}_d$. $\diamond$

Definition 7. Let $y_1, \dots, y_n$ be typed literals. An enumerated datatype is of the form $\text{oneOf}(y_1 \dots y_n)$. $\diamond$

Definition 8. An OWL data range has one of the forms: (i) a datatype URI reference, (ii) an enumerated datatype, or (iii) rdf:Literal . $\diamond$

The semantics of OWL DL datatypes are defined in terms of OWL datatype interpretations.

Definition 9. An OWL datatype interpretation w.r.t. to a datatype map $\mathbf{M}_d$ is a pair $(\Delta_D, \cdot^D)$, where the datatype domain $\Delta_D = \mathbf{PL} \cup \bigcup_{\text{for each supported datatype URIref } u \text{ w.r.t. } \mathbf{M}_p} V(\mathbf{M}_p(u))$ ($\mathbf{PL}$ is the value space for plain literals, i.e., the union of the set of Unicode strings and the set of pairs of Unicode strings and language tags) and $\cdot^D$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $\text{rdfs:Literal}^D = \Delta_D$;
2. for each plain literal l , $l^D = l \in \mathbf{PL}$;
3. for each supported datatype URIref u (let $d = \mathbf{M}_d(u)$):
 - (a) $u^D = V(d) \subseteq \Delta_D$,
 - (b) if $s \in L(d)$, then $(\text{"s"} \hat{^u})^D = L2V(d)(s)$,
 - (c) if $s \notin L(d)$, then $(\text{"s"} \hat{^u})^D$ is not defined;
4. for each unsupported datatype URIref u , $u^D \subseteq \Delta_D$, and $(\text{"s"} \hat{^u})^D \in u^D$.
5. each enumerated datatype $\text{oneOf}(y_1 \dots y_n)$ is interpreted as $y_1^D \cup \dots \cup y_n^D$. $\diamond$

The above definition shows that OWL datatyping is similar to RDF datatyping, except that (i) RDF datatypes are classes, while OWL DL datatypes are not classes,⁶ and (ii) in RDF ill-defined typed literals are interpreted as resources in $\mathbf{IR} \setminus \mathbf{LV}$, while in OWL DL the interpretation of ill-defined typed literals are undefined.

4 Limitations of OWL Datatyping

OWL datatyping has the following serious limitations, which discourage potential users from adopting OWL DL in their SW and ontology applications [15, 19].

1. OWL does not support customised datatypes (except enumerated datatypes). Firstly, XML Schema derived simple types are not OWL DL datatypes, because of the problem of datatype URI references for XML Schema derived simple types. Secondly, OWL does not provide a mechanism to tell which (customised) datatypes can be used together so that the language is still decidable.

⁶ In fact, classes and datatypes in OWL DL use different interpretation functions; cf. Section 2.

2. OWL does not support negated datatypes. For example, ‘all integers but 0’, which is the relativised negation of the enumerated datatype $\text{oneOf}(\text{"0"}^{\wedge}\text{xsd:integer})$, is not expressible in OWL. Moreover, negated datatypes are *necessary* in the negated normal form (NNF)⁷ of datatype-related class descriptions in, e.g., DL tableaux algorithms.
3. An OWL DL datatype domain seriously restricts the interpretations of typed literals with unsupported datatype URIrefs. For example, given the datatype map $\mathbf{M}_{d_1} = \{\text{xsd:integer} \mapsto \text{integer}, \text{xsd:string} \mapsto \text{string}\}$, “1.278e-3”[^] xsd:float has to be interpreted as either an integer, a string or a string with a language tag, which is counter-intuitive.

5 OWL-Eu

This section presents OWL-Eu and elaborates how OWL-Eu satisfies the four requirements (listed in Section 1) in the following four sub-sections.

5.1 Supporting Customised Datatypes

OWL-Eu supports customised datatypes through unary datatype expressions based on unary datatype groups. Intuitively, an unary datatype group extends the OWL datatyping with a hierarchy of supported datatypes.⁸

Definition 10. A unary datatype group $\mathcal{G}$ is a tuple $(\mathbf{M}_d, \mathbf{B}, \text{dom})$, where $\mathbf{M}_d$ is the datatype map of $\mathcal{G}$, $\mathbf{B}$ is the set of primitive base datatype URI references in $\mathcal{G}$ and dom is the declared domain function. We call $\mathbf{S}$ the set of supported datatype URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_d(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. We assume that there exists a unary datatype URI reference $\text{owlx:DatatypeBottom} \notin \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$, if $u \in \mathbf{B}$, $\text{dom}(u) = u$; otherwise, $\text{dom}(u) = v$, where $v \in \mathbf{B}$. $\diamond$

Definition 10 ensures that all the primitive base datatype URIrefs of $\mathcal{G}$ are supported ($\mathbf{B} \subseteq \mathbf{S}$) and that each supported datatype URIref relates to a primitive base datatype URIref through the declared domain function dom .

Example 3. $\mathcal{G}_1 = (\mathbf{M}_{d_1}, \mathbf{B}_1, \text{dom}_1)$ is a unary datatype group, where

- $\mathbf{M}_{d_1} = \{\text{xsd:integer} \mapsto \neg \text{integer}, \text{xsd:string} \mapsto \neg \text{string}, \text{xsd:nonNegativeInteger} \mapsto \geq, \text{xsd:x:integerLessThanN} \mapsto \leq_N\}$,
- $\mathbf{B}_1 = \{\text{xsd:string}, \text{xsd:integer}\}$, and

⁷ A concept is in negation normal form iff negation is applied only to atomic concept names, nominals or datatypes.

⁸ Note that in [15] datatype groups allow arbitrary datatype predicates, while here we consider only datatypes, which can be regarded as *unary* datatype predicates.

- $\text{dom}_1 = \{\text{xsd:integer} \mid \neg \text{xsd:integer}, \text{xsd:string} \mid \neg \text{xsd:string}, \text{xsd:nonNegativeInteger} \mid \neg \text{xsd:integer}, \text{xsd:x:integerLessThanN} \mid \neg \text{xsd:integer}\}.$

According to $\mathbf{M}_{d_1}$, we have $\mathbf{S}_1 = \{\text{xsd:integer}, \text{xsd:string}, \text{xsd:nonNegativeInteger}, \text{xsd:x:integerLessThanN}\}$, hence $\mathbf{B}_1 \subseteq \mathbf{S}_1$. Note that the value space of $<_N$ is

$$V(<_N) = \{i \in V(\text{integer}) \mid i < L2S(\text{integer})(N)\},$$

and by $<_N$ we mean there exists a supported datatype $<_N$ for each integer $L2S(\text{integer})(N)$. $\diamond$

Based on a unary datatype group, OWL-Eu provides a formalism (called datatype expressions) for constructing customised datatypes using supported datatypes.

Definition 11. *Let $\mathcal{G}$ be a unary datatype group. The set of $\mathcal{G}$ -unary datatype expressions in abstract syntax (corresponding DL syntax can be found in Table 4 on page 162), abbreviated $\mathbf{Dexp}(\mathcal{G})$, is inductively defined as follows:*

1. atomic expressions: if u is a datatype *URIref*, then $u \in \mathbf{Dexp}(\mathcal{G})$;
2. relativised negated expressions: if u is a datatype *URIref*, then $\text{not}(u) \in \mathbf{Dexp}(\mathcal{G})$;
3. enumerated datatypes: if $l_1, \dots, l_n$ are literals, then $\text{oneOf}(l_1, \dots, l_n) \in \mathbf{Dexp}(\mathcal{G})$;
4. conjunctive expressions: if $\{E_1, \dots, E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\text{and}(E_1, \dots, E_n) \in \mathbf{Dexp}(\mathcal{G})$;
5. disjunctive expressions: if $\{E_1, \dots, E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\text{or}(E_1, \dots, E_n) \in \mathbf{Dexp}(\mathcal{G})$. $\diamond$

Example 4. $\mathcal{G}$ -unary datatype expressions can be used to represent XML Schema non-list simple types. Given the unary datatype group $\mathcal{G}_1$ presented in Example 3 (page 160),

- built-in XML Schema simple types *integer*, *string*, *nonNegativeInteger* are supported datatypes in $\mathcal{G}_1$;
- the XML Schema derived simple type (using only one facet)

```
<simpleType name = "lessThan5">
  <restriction base = "xsd:integer">
    <maxExclusive value = "5" />
  </restriction>
</simpleType>
```

i.e. $<_5$, is a supported datatype in $\mathcal{G}_1$;

- the XML Schema derived simple type (using more than one facet) “humanAge” presented in Example 1 (page 156) can be represented by the following conjunctive expression

$\text{and}(\text{xsd:nonNegativeInteger}, \text{xsd:x:integerLessThan150});$

Table 4. Syntax and semantics of datatype expressions (OWL-Eu data ranges)

Abstract Syntax	DL Syntax	Semantics
a datatype URIref u	u	$u^{\mathbf{D}}$
oneOf($l_1, \dots, l_n$)	$\{l_1, \dots, l_n\}$	$\{l_1^{\mathbf{D}}\} \cup \dots \cup \{l_n^{\mathbf{D}}\}$
not(u)	$\overline{u}$	$(\mathbf{M}_d(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}$ $\Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$ otherwise
and($E_1, \dots, E_n$)	$E_1 \wedge \dots \wedge E_n$	$E_1^{\mathbf{D}} \cap \dots \cap E_n^{\mathbf{D}}$
or(P, Q)	$E_1 \vee \dots \vee E_n$	$E_1^{\mathbf{D}} \cup \dots \cup E_n^{\mathbf{D}}$

– the following XML Schema derived union simple type

```

<simpleType name = "cameraPrice">
  <union>
    <simpleType>
      <restriction base = "xsd:nonNegativeInteger">
        <maxExclusive value = "100000" />
      </restriction>
    </simpleType>
    <simpleType>
      <restriction base = "xsd:string">
        <enumeration value = "low" />
        <enumeration value = "medium" />
        <enumeration value = "expensive" />
      </restriction>
    </simpleType>
  </union>
</simpleType>

```

can be represented by the following disjunctive expression

```

or(
  and(xsd:nonNegativeInteger, xsdx:integerLessThan100000)
  oneOf("low"^^xsd:string, "medium"^^xsd:string, "expensive"^^xsd:string)
).

```

◇

Definition 12. A datatype interpretation $\mathcal{I}_{\mathbf{D}}$ of a unary datatype group $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \text{dom})$ is a pair $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$, where $\Delta_{\mathbf{D}}$ (the datatype domain) is a non-empty set and $\cdot^{\mathbf{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $(\text{rdfs:Literal})^{\mathbf{D}} = \Delta_{\mathbf{D}}$ and $(\text{owlx:DatatypeBottom})^{\mathbf{D}} = \emptyset$;
2. for each plain literal l , $l^{\mathbf{D}} = l \in \mathbf{PL}$ and $\mathbf{PL} \subseteq \Delta_{\mathbf{D}}$;⁹
3. for any two primitive base datatype URIrefs $u_1, u_2 \in \mathbf{B}$: $u_1^{\mathbf{D}} \cap u_2^{\mathbf{D}} = \emptyset$;
4. for each supported datatype URIref $u \in \mathbf{S}$ (let $d = \mathbf{M}_d(u)$):
 - (a) $u^{\mathbf{D}} = V(d) \subseteq \Delta_{\mathbf{D}}$, $L(u) \subseteq L(\text{dom}(u))$ and $L2S(u) \subseteq L2S(\text{dom}(u))$;
 - (b) if $s \in L(d)$, then $(s^{\wedge}u)^{\mathbf{D}} = L2V(d)(s)$; otherwise, $(s^{\wedge}u)^{\mathbf{D}}$ is not defined;
5. $\forall u \notin \mathbf{S}$, $u^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, and $(v^{\wedge}u)^{\mathbf{D}} = u^{\mathbf{D}}$.

⁹ $\mathbf{PL}$ is the value space for plain literals; cf. Definition 9 on page 159.

Moreover, we extend $\cdot^{\mathbf{D}}$ to $\mathcal{G}$ unary datatype expression as shown in Table 4 (page 162). Let E be a $\mathcal{G}$ unary datatype expression, the negation of E is of the form $\neg E$, which is interpreted as $\Delta_{\mathbf{D}} \setminus E^{\mathbf{D}}$. $\diamond$

In Definition 12, Condition 3 ensures that the value spaces of all primitive base datatypes are disjoint with each other. Condition 4a ensures that each supported datatype is a derived datatype of its primitive base datatype. Please note the difference between a relativised negated expression and the negation of a unary datatype expression: the former one is a kind of unary datatype expression, while the latter one is the form of negation of all kinds of unary datatype expressions.

Now we introduce the kind of basic reasoning mechanisms required for a unary datatype group.

Definition 13. Let $\mathbf{V}$ be a set of variables, $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \text{dom})$ a unary datatype group. A datatype conjunctions of $\mathcal{G}$ of the form

$$\mathcal{C} = \bigwedge_{j=1}^k u_j(v_j) \wedge \bigwedge_{i=1}^l \neq_i(v_1^{(i)}, v_2^{(i)}), \quad (1)$$

where the v_j are variables from $\mathbf{V}$, $v_1^{(i)}, v_2^{(i)}$ are variables appear in $\bigwedge_{j=1}^k u_j(v_j)$, u_j are datatype URI references from $\mathbf{S}$ and $\neq_i$ are the inequality predicates for primitive base datatypes $\mathbf{M}_d(\text{dom}(u_i))$ where u_i appear in $\bigwedge_{j=1}^k u_j(v_j)$.

A predicate conjunction $\mathcal{C}$ is called satisfiable iff there exist an interpretation $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$ of $\mathcal{G}$ and a function δ mapping the variables in $\mathcal{C}$ to data values in $\Delta_{\mathbf{D}}$ s.t. $\delta(v_j) \in u_j^{\mathbf{D}}$ (for all $1 \leq j \leq k$) and $\{\delta(v_1^{(i)}), \delta(v_2^{(i)})\} \subseteq u_i^{\mathbf{D}}$ and $\delta(v_1^{(i)}) \neq \delta(v_2^{(i)})$ (for all $1 \leq i \leq l$). Such a function δ is called a solution for $\mathcal{C}$ w.r.t. $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$. $\diamond$

We end this section by elaborating the conditions that computable unary datatype groups require.

Definition 14. A unary datatype group $\mathcal{G}$ is conforming iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exist $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathbf{D}} = \bar{u}^{\mathbf{D}}$, and
2. the satisfiability problems for finite datatype conjunctions of the form (1) is decidable. $\diamond$

5.2 Small Extension: From OWL DL to OWL-Eu

In this section, we present a small extension of OWL DL, i.e., OWL-Eu. The underpinning DL of OWL-Eu is $\mathcal{SHOIN}(\mathcal{G}_1)$, i.e., the $\mathcal{SHOIN}$ DL combined with a unary datatype group $\mathcal{G}$ (1 for unary). Specifically, OWL-Eu (only) extends OWL data range (cf. Definition 8) to OWL-Eu data ranges defined as follows.

Definition 15. An OWL-Eu data range is a $\mathcal{G}$ unary datatype expression. Abstract (as well as DL) syntax and model-theoretic semantics of OWL-Eu data ranges are presented in Table 4 (page 162). $\diamond$

The consequence of the extension is that customised datatypes, represented by OWL-Eu data ranges, can be used in datatype exists restrictions ($\exists T.u$) and datatype value restrictions ($\forall T.u$), where T is a datatype property and u is an OWL-Eu data range (cf. Table 2 on page 155). Hence, this extension of OWL DL is as large as is necessary to support customised datatypes.

Example 5. PCs with memory size greater than or equal to 512 Mb and with price cheaper than 700 pounds can be represented in the following OWL-Eu concept description in DL syntax (cf. Table 4 on page 162):

$$\text{PC} \sqcap \exists \text{memorySizeInMb}.\overline{\leq_{512}} \sqcap \exists \text{priceInPound}.\leq_{700},$$

where $\overline{\leq_{512}}$ is a relativised negated expression and $\leq_{700}$ is a supported datatype in $\mathcal{G}_1$. $\diamond$

5.3 Decidability of OWL-Eu

Theorem 5.19 of [15] indicates that we can combine any decidable DL that provides the conjunction ($\sqcap$) and bottom ($\perp$) constructors with a conforming unary datatype group and the combined DL is still decidable. Therefore, OWL-Eu is decidable.

Theorem 1. (*Theorem 6.2 of [15]*) *The knowledge base satisfiability problem of OWL-Eu is decidable if the combined unary datatype group is conforming.*

5.4 Overcoming the Limitations of OWL Datatyping

This section summarises how OWL-Eu overcomes the limitations of OWL datatyping presented in Section 4. Firstly, OWL-Eu is a decidable extension (Theorem 1) of OWL DL that supports customised datatypes with unary datatype expressions (cf. Example 4). Secondly, Definition 12 defines the negations of datatype expressions and OWL-Eu provides relativised negated datatype expression (Definition 11). Thirdly, according to Definition 12, the datatype domain in an interpretation of a datatype group is a superset of (instead of equivalent to) the value spaces of primitive base datatypes and plain literals; hence, typed literals with unsupported predicates are interpreted more intuitively.

6 Related Work

The concrete domain approach [2, 14] provides a rigorous treatment of datatype predicates, rather than datatypes.¹⁰ In the type system approach [12], datatypes are considered to be sufficiently structured by type systems; however, it does not specify how the derivation mechanism of a type system affects the set of datatypes $\mathbf{D}$. [5] suggests some solutions to the problem of referring to an XML Schema user defined simple type with a URI reference; however, it does not address the computability issue of combining the *SHOIN* DL with customised datatypes.

¹⁰ The reader is referred to Section 5.1.3 of [15] for detailed discussions on concrete domains.

7 Discussion

Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome. Accordingly, the Semantic Web Best Practices and Development Working Group sets up a task force to address this issue. As discussed above, a solution for the problem should cover much more than just a standard way of referring to an XML Schema user defined simple type with a URI reference.

In this paper, we propose OWL-Eu, an extension of OWL DL that supports customised datatypes. The underpinning of OWL-Eu is the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL, a combination of $\mathcal{SHOIN}$ and a unary datatype group. OWL-Eu is decidable if the combined unary datatype group is conforming; the conformability of a unary datatype group precisely specifies the conditions on the set of supported datatypes. OWL-Eu provides a general framework for integrating OWL DL with customised datatypes, such as XML Schema non-list simple types.

We have implemented a prototype extension of the FaCT [9] DL system to support TBox reasoning of the $\mathcal{SHIQ}(\mathcal{G}_1)$ DL, a sub-language of OWL-Eu. As for future work, we are planning to extend the DIG1.1 interface [7] to support OWL-Eu and to implement a Protégé [13] plug-in to support XML Schema non-list simple types, i.e. users should be able to define and/or import customised XML Schema non-list simple types based on a set of supported datatypes, and to exploit our prototype through the extended DIG interface.

References

- [1] H. Alvestrand. Rfc 3066 - tags for the identification of languages. Technical report, IETF, Jan 2001. <http://www.isi.edu/in-notes/rfc3066.txt>.
- [2] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.
- [3] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. <http://www.w3.org/TR/owl-ref/>, Feb 2004.
- [4] Paul V. Biron and Ashok Malhotra. Extensible Markup Language (XML) Schema Part 2: Datatypes – W3C Recommendation 02 May 2001. Technical report, World Wide Web Consortium, 2001. <http://www.w3.org/TR/xmlschema-2/>.
- [5] Jeremy J. Carroll and Jeff Z. Pan. XML Schema Datatypes in RDF and OWL. Technical report, W3C Semantic Web Best Practices and Development Group, Nov 2004. Editors' Draft, <http://www.w3.org/2001/sw/BestPractices/XSCH/xsch-sw/>.
- [6] Unicode Consortium. *The Unicode Standard*. Addison-Wesley, 2000. ISBN 0-201-61633-5. version 3.
- [7] DIG. SourceForge DIG Interface Project. <http://sourceforge.net/projects/dig/>, 2004.
- [8] Patrick Hayes. RDF Semantics. Technical report, W3C, Feb 2004. W3C recommendation, <http://www.w3.org/TR/rdf-mt/>.
- [9] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR'98*, pages 636–647, 1998.

- [10] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proc. of Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in LNAI, pages 161–180, 1999.
- [11] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [12] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the *SHOQ(D)* description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.
- [13] Holger Knublauch, Ray W. Ferguson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.
- [14] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.
- [15] Jeff Z. Pan. *Description Logics: Reasoning Support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK, 2004.
- [16] Jeff Z. Pan and Ian Horrocks. Extending Datatype Support in Web Ontology Reasoning. In *Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEMantics (ODBASE 2002)*, Oct 2002.
- [17] Jeff Z. Pan and Ian Horrocks. Web Ontology Reasoning with Datatype Groups. In *Proc. of the 2003 International Semantic Web Conference (ISWC2003)*, pages 47–63, 2003.
- [18] RDF-Logic Mailing List. <http://lists.w3.org/archives/public/www-rdf-logic/>. W3C Mailing List, starts from 2001.
- [19] A. Rector. Re: [UNITS, OEP] FAQ : Constraints on data values range. Discussion in [20], Apr. 2004. <http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0216.html>.
- [20] Semantic Web Best Practice and Development Working Group Mailing List. <http://lists.w3.org/archives/public/public-swbp-wg/>. W3C Mailing List, starts from 2004.

Towards a Fuzzy Description Logic for the Semantic Web (Preliminary Report)

U. Straccia

ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy
straccia@isti.cnr.it

Abstract. In this paper we present a fuzzy version of $\mathcal{SHOIN}(\mathcal{D})$, the corresponding Description Logic of the ontology description language OWL DL. We show that the representation and reasoning capabilities of fuzzy $\mathcal{SHOIN}(\mathcal{D})$ go clearly beyond classical $\mathcal{SHOIN}(\mathcal{D})$. We present its syntax and semantics. Interesting features are that concrete domains are fuzzy and entailment and subsumption relationships may hold to some degree in the unit interval $[0, 1]$.

1 Introduction

In the last decade a substantial amount of work has been carried out in the context of *Description Logics* (DLs) [2]. DLs are a logical reconstruction of the so-called frame-based knowledge representation languages, with the aim of providing a simple well-established Tarski-style declarative semantics to capture the meaning of the most popular features of structured representation of knowledge.

Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web* [4, 15]. Semantic Web has recently attracted much attention both from academia and industry, and is widely regarded as the next step in the evolution of the World Wide Web. It aims at enhancing content on the World Wide Web with meta-data, enabling *agents* (machines or human users) to *process*, *share* and *interpret* Web content.

Ontologies [10] play a key role in the Semantic Web and major effort has been put by the Semantic Web community into this issue. Informally, an ontology consists of a hierarchical description of important concepts in a particular domain, along with the description of the properties (of the instances) of each concept. DLs play a particular role in this context as they are essentially the theoretical counterpart of the *Web Ontology Language OWL DL*, the state of the art language to specify ontologies. Web content is then annotated by relying on the concepts defined in a specific domain ontology.

However, OWL DL becomes less suitable in all those domains in which the concepts to be represent have not a precise definition. If we take into account that we have to deal with Web content, then it is easily verified that this scenario is, unfortunately, likely the rule rather than an exception. For instance, just consider the case we would like to build an ontology about flowers. Then we may encounter the problem of representing

concepts like¹ “Candia is a creamy white rose with dark pink edges to the petals”, “Jacaranda is a hot pink rose”, “Calla is a very large, long white flower on thick stalks”. As it becomes apparent such concepts hardly can be encoded into OWL DL, as they involve so-called *fuzzy* or *vague concepts*, like “creamy”, “dark”, “hot”, “large” and “thick”, for which a clear and precise definition is not possible.²

The problem to deal with imprecise concepts has been addressed several decades ago by Zadeh [31], which gave birth in the meanwhile to the so-called *fuzzy set and fuzzy logic theory* and a huge number of real life applications exists. Unfortunately, despite the popularity of fuzzy set theory, relative little work has been carried out in extending DLs towards the representation of imprecise concepts, notwithstanding DLs can be considered as a quite natural candidate for such an extension [5, 6, 13, 23, 25, 26, 27, 29, 30] (see also [9], Chapter 6).

In this paper we consider a fuzzy extension of $\mathcal{SHOIN}(\mathcal{D})$, the corresponding DL of the ontology description language OWL DL, and present its syntax and semantics. The main feature of fuzzy $\mathcal{SHOIN}(\mathcal{D})$ is that it allows us to represent and reason about vague concepts. None of the approaches to fuzzy DLs deal with the expressive power of the fuzzy extension of $\mathcal{SHOIN}(\mathcal{D})$ we present here. Our purpose is also to integrate most of these contributions within an unique setting and, thus, hope to define a reference language for fuzzy $\mathcal{SHOIN}(\mathcal{D})$. A main feature of fuzzy $\mathcal{SHOIN}(\mathcal{D})$ is that the subsumption relation between classes and the entailment relation is no more a crisp yes/no problem, but it becomes now fuzzy, i.e. is established *to some degree*.

We will proceed as follows. In the following section we recall the description logic $\mathcal{SHOIN}(\mathcal{D})$. In Section 3 we extend $\mathcal{SHOIN}(\mathcal{D})$ to the fuzzy case and discuss some properties of it. Section 4 concludes and presents some topics for further research.

2 Preliminaries

The ontology language OWL DL is a syntactic variant of $\mathcal{SHOIN}(\mathcal{D})$ [15]. Although several XML and RDF syntaxes for OWL-DL exist, in this paper we use the traditional description logic notation. For explicating the relationship between OWL DL and DLs syntax, see e.g. [15, 16]. The purpose of this section is to make the paper self-contained. More importantly it helps in understanding the differences between classical $\mathcal{SHOIN}(\mathcal{D})$ and fuzzy $\mathcal{SHOIN}(\mathcal{D})$. The reader confident with the $\mathcal{SHOIN}(\mathcal{D})$ terminology may skip directly to Section 3.

Syntax. $\mathcal{SHOIN}(\mathcal{D})$ allows to reason with concrete data types, such as strings and integers using so-called *concrete domains* [1, 18, 20, 21]. A *concrete domain* $\mathcal{D}$ is a pair $\langle \Delta_{\mathcal{D}}, \Phi_{\mathcal{D}} \rangle$, where $\Delta_{\mathcal{D}}$ is an interpretation domain and $\Phi_{\mathcal{D}}$ is the set of concrete domain predicates d with a predefined arity n and an interpretation $d^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^n$. For instance, over the integers $\geq_{20}$ may be a unary predicate denoting the set of integers greater or

¹ Taken from a text book on flowers.

² Another issue relates to the representation of terms like “very”, which are called *fuzzy concepts modifiers*, as we will see later on.

equal to 20. For instance, $\text{Person} \sqcap \exists \text{age} . \geq_{20}$ denotes a person whose age is greater or equal to 20. So, let $\mathbb{C}$, $\mathbb{R}_a$, $\mathbb{R}_c$, $\mathbb{I}_a$ and $\mathbb{I}_c$ be non-empty finite and pair-wise disjoint sets of *concepts names*, *abstract roles names*, *concrete roles names*, *abstract individual names* and *concrete individual names*. An *abstract role* is an abstract role name or the inverse S^- of an abstract role name S (concrete role names do not have inverses). An *RBox* $\mathcal{R}$ consists of a finite set of transitivity axioms $\text{trans}(R)$, and role inclusion axioms of the form $R \sqsubseteq S$ and $T \sqsubseteq U$, where R and S are abstract roles, and T and U are concrete roles. The reflexive-transitive closure of the role inclusion relationship is denoted with $\sqsubseteq^*$. A role not having transitive sub-roles is called *simple role*. The set of $\text{SHOIN}(\mathbb{D})$ *concepts* is defined by the following syntactic rules, where A is an atomic concept, R is an abstract role, S is an abstract simple role, T_i are concrete roles, d is a concrete domain predicate, a_i and c_i are abstract and concrete individuals, respectively, and $n \in \mathbb{N}$:

$$C \longrightarrow \top \mid \perp \mid A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid \forall R.C \mid \exists R.C \mid \geq n S \mid \leq n S \mid \{a_1, \dots, a_n\} \mid \geq n T \mid \leq n T \mid \forall T_1, \dots, T_n.D \mid \exists T_1, \dots, T_n.D$$

$$D \longrightarrow d \mid \{c_1, \dots, c_n\}$$

For instance, we may write the concept $\text{Flower} \sqcap (\exists \text{hasPetalWidth} . (\geq_{20\text{mm}} \sqcap \leq_{40\text{mm}})) \sqcap \exists \text{hasColour} . \text{Red}$ to informally denote the set of flowers having petal's dimension within 20mm and 40mm, whose colour is red. Here $\geq_{20\text{mm}}$ (and $\leq_{40\text{mm}}$) is a concrete domain predicate. We use $(= 1 S)$ as an abbreviation for $(\geq 1 S) \sqcap (\leq 1 S)$. A *TBox* $\mathcal{T}$ consists of a finite set of concept inclusion axioms $C \sqsubseteq D$, where C and D are concepts. For ease, we use $C = D \in \mathcal{T}$ in place of $C \sqsubseteq D$, $D \sqsubseteq C \in \mathcal{T}$. An *ABox* $\mathcal{A}$ consists of a finite set of concept and role assertion axioms and individual (in)equality axioms $a:C$, $(a, b):R$, $(a, c):T$, $a \approx b$ and $a \not\approx b$, respectively. A $\text{SHOIN}(\mathbb{D})$ *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ consists of a TBox $\mathcal{T}$, a RBox $\mathcal{R}$, and an ABox $\mathcal{A}$.

Semantics. An *interpretation* $\mathcal{I}$ with respect to a concrete domain $\mathbb{D}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non empty set $\Delta^{\mathcal{I}}$ (called the *domain*), disjoint from $\Delta_{\mathbb{D}}$, and of an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each $C \in \mathbb{C}$ a subset of $\Delta^{\mathcal{I}}$, to each $R \in \mathbb{R}_a$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, to each $a \in \mathbb{I}_a$ an element in $\Delta^{\mathcal{I}}$, to each $c \in \mathbb{I}_c$ an element in $\Delta_{\mathbb{D}}$, to each $T \in \mathbb{R}_c$ a subset of $\Delta^{\mathcal{I}} \times \Delta_{\mathbb{D}}$ and to each n -ary concrete predicate d the interpretation $d^{\mathbb{D}} \subseteq \Delta_{\mathbb{D}}^n$. The mapping $\cdot^{\mathcal{I}}$ is extended to concepts and roles as usual: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $\perp^{\mathcal{I}} = \emptyset$,

$$\begin{aligned} (C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\ (C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (S^-)^{\mathcal{I}} &= \{\langle y, x \rangle : \langle x, y \rangle \in S^{\mathcal{I}}\} \\ (\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\} \\ (\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(x) \cap C^{\mathcal{I}} \neq \emptyset\} \\ (\geq n S)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} : |S^{\mathcal{I}}(x)| \geq n\} \\ (\leq n S)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} : |S^{\mathcal{I}}(x)| \leq n\} \\ \{a_1, \dots, a_n\}^{\mathcal{I}} &= \{a_1^{\mathcal{I}}, \dots, a_n^{\mathcal{I}}\} \end{aligned}$$

and similarly for the other constructs, where $R^{\mathcal{I}}(x) = \{y: \langle x, y \rangle \in R^{\mathcal{I}}\}$ and $|X|$ denotes the cardinality of the set X . In particular,

$$(\exists T_1, \dots, T_n. d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} : [T_1^{\mathcal{I}}(x) \times \dots \times T_n^{\mathcal{I}}(x)] \cap d^{\mathcal{I}} \neq \emptyset\}.$$

The *satisfiability* of an axiom E in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, denoted $I \models E$, is defined as follows: $I \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $I \models R \sqsubseteq S$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, $I \models T \sqsubseteq U$ iff $T^{\mathcal{I}} \subseteq U^{\mathcal{I}}$, $I \models \text{trans}(R)$ iff $R^{\mathcal{I}}$ is transitive, $I \models a:C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $I \models (a, b):R$ iff $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$, $I \models (a, c):T$ iff $\langle a^{\mathcal{I}}, c^{\mathcal{I}} \rangle \in T^{\mathcal{I}}$, $I \models a \approx b$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$, $I \models a \not\approx b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. An abstract simple role S is called *functional* if the interpretation of role S is always functional. A functional role S can always be obtained from an abstract role by means of the axiom $\top \sqsubseteq (\leq 1 S)$. Therefore, whenever we say that a role is functional, we assume that $\top \sqsubseteq (\leq 1 S)$ is in the ABox. For a set of axioms $\mathcal{E}$, we say that I *satisfies* $\mathcal{E}$ iff I satisfies each element in $\mathcal{E}$. If $I \models E$ (resp. $I \models \mathcal{E}$) we say that I is a *model* of E (resp. $\mathcal{E}$). I *satisfies* (is a *model* of) a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, denoted $I \models \mathcal{K}$, iff I is a model of each component $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{A}$, respectively. An axiom E is a *logical consequence* of a knowledge base $\mathcal{K}$, denoted $\mathcal{K} \models E$ iff every model of $\mathcal{K}$ satisfies E . According to [16], the entailment and subsumption problem can be reduced to knowledge base satisfiability problem, for which decision procedures and reasoning tools exists (e.g. RACER [11] and FACT [14]).

Example 1. Let us consider the following excerpt of a simple ontology (TBox $\mathcal{T}$) about cars, with empty RBox ($\mathcal{R} = \emptyset$):

$$\text{Car} \sqsubseteq (= 1 \text{ maker}) \sqcap (= 1 \text{ passanger}) \sqcap (= 1 \text{ speed})$$

$$\begin{array}{ll} (= 1 \text{ maker}) \sqsubseteq \text{Car} & \top \sqsubseteq \forall \text{maker. Maker} \\ (= 1 \text{ passanger}) \sqsubseteq \text{Car} & \top \sqsubseteq \forall \text{passanger. } \mathbb{N} \\ (= 1 \text{ speed}) \sqsubseteq \text{Car} & \top \sqsubseteq \forall \text{speed. Km/h} \end{array}$$

$$\begin{array}{l} \text{Roadster} \sqsubseteq \text{Cabriolet} \sqcap \exists \text{passenger. } \{2\} \\ \text{Cabriolet} \sqsubseteq \text{Car} \sqcap \exists \text{topType. SoftTop} \\ \text{SportsCar} = \text{Car} \sqcap \exists \text{speed. } \geq_{245 \text{ km/h}} \end{array}$$

In $\mathcal{T}$, the value for speed ranges over the concrete domain of kilometers per hour, Km/h, while the value for passengers ranges over the concrete domain of natural numbers, $\mathbb{N}$. The concrete predicate $\geq_{245 \text{ km/h}}$ is true if the value is greater or equal than to 245km/h. The ABox $\mathcal{A}$ contains the following assertions:

$$\begin{array}{l} \text{mgb:Roadster} \sqcap (\exists \text{maker. } \{\text{mg}\}) \sqcap (\exists \text{speed. } \{170 \text{ km/h}\}) \\ \text{enzo:Car} \sqcap (\exists \text{maker. } \{\text{ferrari}\}) \sqcap (\exists \text{speed. } >_{350 \text{ km/h}}) \\ \text{tt:Car} \sqcap (\exists \text{maker. } \{\text{audi}\}) \sqcap (\exists \text{speed. } \{243 \text{ km/h}\}) \end{array}$$

Consider the knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$. It is then easily verified that, e.g.

$$\begin{array}{ll} \mathcal{K} \models \text{Roadster} \sqsubseteq \text{Car} & \mathcal{K} \models \text{mg:Maker} \\ \mathcal{K} \models \text{enzo:SportsCar} & \mathcal{K} \models \text{tt:}\neg \text{SportsCar} . \end{array}$$

The above example illustrates an evident difficulty in defining the class of sport cars. Indeed, it is highly questionable why a car whose speed is 243km/h is *not* a sport car any more. The point is that essentially, the higher the speed the more likely a car is a sports car, which makes the concept of sports car rather a *fuzzy* concept, i.e. *vague* concept, rather than a crisp one. In the next section we will see how to represent such concepts more appropriately.

3 Fuzzy OWL DL

Fuzzy sets have been introduced by Zadeh [31] as a way to deal with vague concepts like low pressure, high speed and the like. Formally, a *fuzzy set* A with respect to a universe X is characterized by a *membership function* $\mu_A : X \rightarrow [0, 1]$, assigning an A -membership degree, $\mu_A(x)$, to each element x in X . $\mu_A(x)$ gives us an estimation of the belonging of x to A . Typically, if $\mu_A(x) = 1$ then x definitely belongs to A , while $\mu_A(x) = 0.8$ means that x is “close” to be an element of A .

When we switch to fuzzy logics, the notion of degree of membership $\mu_A(x)$ of an element $x \in X$ w.r.t. the fuzzy set A over X is regarded as the *degree of truth* in $[0, 1]$ of the statement “ x is A ”. Accordingly, in our fuzzy DL, (i) a concept C , rather than being interpreted as a classical set, will be interpreted as a fuzzy set and, thus, concepts become *imprecise*; and, consequently, (ii) the statement “ a is C ”, i.e. $a:C$, will have a truth-value in $[0, 1]$ given by the degree of membership of being the individual a a member of the fuzzy set C .

In the following, we present first some preliminaries on fuzzy set theory (for a more complete and comprehensive presentation see e.g. [7]) and then define fuzzy *SHOIN*(D).

3.1 Preliminaries on Fuzzy Set Theory

Let X be a countable crisp set and let A be a fuzzy subset of X , with membership function $\mu_A(x)$, or simply $A(x) \in [0, 1], x \in X$. The *support* of A , $\text{supp}(A)$, is the crisp set $\text{supp}(A) = \{x \in X : A(x) \neq 0\}$. The *scalar cardinality* of A , $|A|$, is defined as $|A| = \sum_{x \in X} A(x)$. The *fuzzy powerset* of X , $\mathcal{F}(X)$, is the set of all the fuzzy sets over X . Let $A, B \in \mathcal{F}(X)$. We say that A and B are *equal* iff $A(x) = B(x), \forall x \in X$. A is a *subset* of B iff $A(x) \leq B(x), \forall x \in X$. We will see later on a different notion of subset, in which A is a subset of B to some degree in $[0, 1]$. We next give the basic definitions on fuzzy set operations (complement, intersection and union).

The *complement* of A , $\neg A$, is given by membership function $(\neg A)(x) = n(A(x))$, for any $x \in X$. The function $n: [0, 1] \rightarrow [0, 1]$, called *negation*, has to satisfy the following conditions and extends boolean negation:

- $n(0) = 1$ and $n(1) = 0$;
- $\forall a, b \in [0, 1], a \leq b$ implies $n(b) \leq n(a)$;
- $\forall a \in [0, 1], n(n(a)) = a$.

Several negation functions have been given in the literature, e.g. Lukasiewicz negation $n_L(a) = 1 - a$ (syntax, $\neg_L$) and Gödel negation $n_G(0) = 1$ and $n(a) = 0$ if $a > 0$ (syntax, $\neg_G$).

The *intersection* of two fuzzy sets A and B is given $(A \wedge B)(x) = t(A(x), B(x))$, where t is a *triangular norm*, or simply *t-norm*. A t-norm is a function $t: [0, 1] \times [0, 1] \rightarrow [0, 1]$ that has to satisfy the following conditions:

- $\forall a \in [0, 1], t(a, 1) = a$;
- $\forall a, b, c \in [0, 1], b \leq c$ implies $t(a, b) \leq t(a, c)$;
- $\forall a, b \in [0, 1], t(a, b) = t(b, a)$;
- $\forall a, b, c \in [0, 1], t(a, t(b, c)) = t(t(a, b), c)$.

Examples of t-norms are: $t_L(a, b) = \max(a + b - 1, 0)$ (Lukasiewicz t-norm, syntax $\wedge_L$), $t_G(a, b) = \min(a, b)$ (Gödel t-norm, syntax $\wedge_G$), and $t_P(a, b) = a \cdot b$ (product t-norm, syntax $\wedge_P$). Note that $\forall a \in [0, 1], t(a, 0) = 0$.

The *union* of two fuzzy sets A and B is given $(A \vee B)(x) = s(A(x), B(x))$, where s is a *triangular co-norm*, or simply *s-norm*. A s-norm is a function $s: [0, 1] \times [0, 1] \rightarrow [0, 1]$ that has to satisfy the following conditions:

- $\forall a \in [0, 1], s(a, 0) = a$;
- $\forall a, b, c \in [0, 1], b \leq c$ implies $s(a, b) \leq s(a, c)$;
- $\forall a, b \in [0, 1], s(a, b) = s(b, a)$;
- $\forall a, b, c \in [0, 1], s(a, s(b, c)) = s(s(a, b), c)$.

Examples of s-norms are: $s_L(a, b) = \min(a + b, 1)$ (Lukasiewicz s-norm, syntax $\vee_L$), $s_G(a, b) = \max(a, b)$ (Gödel s-norm, syntax $\vee_G$), and $s_P(a, b) = a + b - a \cdot b$ (product s-norm, syntax $\vee_P$). Note that if we consider Lukasiewicz negation, then Lukasiewicz, Gödel and product s-norm are related to their respective t-norm according to the De Morgan law: $\forall a, b \in [0, 1], s(a, b) = n(t(n(a), n(b)))$.

Another important operator is *implication*, denoted $\rightarrow$, that gives a truth-value to the formula $A \rightarrow B$, when the truth of A and B are known. A fuzzy implication is a function $i: [0, 1] \times [0, 1] \rightarrow [0, 1]$ that has to satisfy the following conditions:

- $\forall a, b, c \in [0, 1], a \leq b$ implies $i(a, c) \geq i(b, c)$;
- $\forall a, b, c \in [0, 1], b \leq c$ implies $i(a, b) \leq i(a, c)$;
- $\forall a \in [0, 1], i(0, b) = 1$;
- $\forall a \in [0, 1], i(a, 1) = 1$;
- $i(1, 0) = 0$.

In classical logic, $a \rightarrow b$ is a shorthand for $\neg a \vee b$. A generalization to fuzzy logic is, thus, $\forall a, b \in [0, 1], i(a, b) = s(n(a), b)$. For instance, $\forall a, b \in [0, 1], i_{KD}(a, b) = \max(1 - a, b)$ is the so-called Kleene-Dienes implication (syntax, $\rightarrow_{KD}$). Another approach to fuzzy implication is based on the so-called *residuum*. His formulation starts from the fact that in classical logic $\neg a \vee b$ can be re-written as $\max\{c \in \{0, 1\} : a \wedge c \leq b\}$. Therefore, another generalization of implication to fuzzy logic is

$$\forall a, b \in [0, 1], i(a, b) = \sup\{c \in [0, 1] : t(a, c) \leq b\}.$$

For residuum based implication, $i(a, b) = 1$ if $a \leq b$. If $a > b$ then, according to the chosen t-norm, we have that e.g. $i_L(a, b) = 1 - a + b$ for Lukasiewicz implication (syntax, $\rightarrow_L$), $i_G(a, b) = b$ for Gödel implication (syntax, $\rightarrow_G$) and $i_P(a, b) = a/b$ for product implication (syntax, $\rightarrow_P$). Note that, for Lukasiewicz implication, s-norm and negation, we have $i_L(a, b) = s_L(n_L(a), b)$. The same holds using Kleene-Dienes implication, Lukasiewicz negation and Gödel s-norm. On the other hand $i_P(a, b) \neq s_P(n_G(a), b)$ (for instance, for $0 < a \leq b < 1$, $i_P(a, b) = 1$, while $s_P(n_G(a), b) = b < 1$).

Another interesting question is when $\forall a, b \in [0, 1], i(a, b) = n(t(a, n(b)))$ holds, which in formulae is formulated as $a \rightarrow b \equiv \neg(a \wedge \neg b)$. It turns out that e.g., in Zadeh's logic [31] (i.e. using $\rightarrow_{KD}, \wedge_G, \neg_L$) this relation holds. It holds as well in the so-called Lukasiewicz logic (i.e. using $\rightarrow_L, \wedge_L, \neg_L$), while it does neither hold for Gödel logic (i.e. using $\rightarrow_G, \wedge_G, \neg_G$) nor for the product logic (i.e. using $\rightarrow_P, \wedge_P, \neg_G$). For them, just consider the case $1 > a > b > 0$ to verify the inequality. We will see later on that whenever $i(a, b) \neq n(t(a, n(b)))$ then under the fuzzy semantics, $\forall R.C$ is not equivalent to $\neg \exists R.\neg C$.

Fuzzy implication can also be used to determine the degree of subset relationship between two fuzzy subsets A and B over X . Indeed, we define the *degree of subsumption* between A and B , denoted $A \rightarrow B$, as $\inf_{x \in X} i(A(x), B(x))$, where i is an implication function. Note that if $\forall x \in [0, 1], A(x) \leq B(x)$ holds then $A \rightarrow B$ evaluates to 1. Of course, it may be that $A \rightarrow B$ evaluates to a value $0 < v < 1$ as well.

We conclude the discussion on fuzzy implication by noting that we have the following inferences: assume $a \geq n$ and $i(a, b) \geq m$. Then

- under Kleene-Dienes implication we infer that if $n > 1 - m$ then $b \geq m$. Indeed, from $i(a, b) = \max(1 - a, b) \geq m$, either $1 - a \geq m$ or $b \geq m$. But $a \geq n$, so $1 - a \geq m$ implies $1 - m \geq a \geq n > 1 - m$, a contradiction. Therefore, $b \geq m$ must hold.
- under residuum based implication w.r.t. a t-norm t , we infer that $b \geq t(n, m)$. Indeed, from $i(a, b) = \sup\{c: t(a, c) \leq b\} \geq m$ and $a \geq n$ we have $t(n, m) \leq t(n, c) \leq t(a, c) \leq b$.

A (binary) *fuzzy relation* R over two countable crisp sets X and Y is a function $R: X \times Y \rightarrow [0, 1]$. The *inverse* of R is the function $R^{-1}: Y \times X \rightarrow [0, 1]$ with membership function $R^{-1}(y, x) = R(x, y)$, for every $x \in X$ and $y \in Y$. The *composition* of two fuzzy relations $R_1: X \times Y \rightarrow [0, 1]$ and $R_2: Y \times Z \rightarrow [0, 1]$ is defined as $(R_1 \circ R_2)(x, z) = \sup_{y \in Y} t(R_1(x, y), R_2(y, z))$, where t is a t-norm. A fuzzy relation R is said to be *transitive* iff $R(x, z) = (R \circ R)(x, z)$.

We conclude this part with fuzzy modifiers. Fuzzy modifiers applies to fuzzy sets to change their membership function. Well known examples are modifiers like *very*, *more_or_less*, *slightly*, etc. These allow us to define fuzzy sets like *very(High)* and *slightly(Mature)*. Formally, a *modifier*, m , is a function $m: [0, 1] \rightarrow [0, 1]$. For instance, we may define *very*(x) = x^2 , while define *slightly*(x) = $\sqrt{x}$.

In the following, we use $\wedge, \vee, \neg$ and $\rightarrow$ in infix notation, in place of a t-norm t , s-norm s , negation n and implication operator i .

3.2 Fuzzy *SHOIN*(D)

In this section we give syntax and semantics of fuzzy *SHOIN*(D), using the fuzzy operators defined in the previous section. We generalize the semantics given in [13, 26, 29].

Syntax. We have seen that *SHOIN*(D) allows to reason with concrete data types, such as strings and integers using so-called concrete domains. In our fuzzy approach, concrete domains may be based on fuzzy sets as well. A *concrete fuzzy domain* is a pair $\langle \Delta_D, \Phi_D \rangle$, where Δ_D is an interpretation domain and Φ_D is the set of concrete fuzzy domain predicates d with a predefined arity n and an interpretation $d^D: \Delta_D^n \rightarrow [0, 1]$, which is a n -ary fuzzy relation over Δ_D . For instance, as for *SHOIN*(D), the predicate $\leq_{18}$ may be an unary crisp predicate over the natural numbers denoting the set of integers smaller or equal to 18, i.e. $\leq_{18}: \text{Natural} \rightarrow [0, 1]$ and $\leq_{18}(x) = 1$ if $x \leq 18$, $\leq_{18}(x) = 0$ otherwise. So,

$$\text{Minor} = \text{Person} \sqcap \exists \text{age.} \leq_{18} \quad (1)$$

defines a person, whose age is less or equal 18, i.e. it defines a minor. On the other hand, $\text{Young}: \text{Natural} \rightarrow [0, 1]$ may be a fuzzy concrete predicate over the natural numbers denoting the degree of youngness of a person's age. The concrete fuzzy predicate Young may be defined as $\text{Young}(x) = \max(0, 1 - 0.00075x^2)$. So,

$$\text{YoungPerson} = \text{Person} \sqcap \exists \text{age. Young} \quad (2)$$

will denote a young person. Furthermore, by referring to Example 1, we may define the concept of sports car as the concept

$$\text{SportsCar} = \text{Car} \sqcap \exists \text{speed. very(High)}, \quad (3)$$

where *very* is a concept modifier and *High* is a fuzzy concrete predicate over the domain of speed expressed in kilometers per hour and may be defined as $\text{High}(x) = \min(1, 0.004x)$.

Similarly, we may represent “Calla is a very large, long white flower on thick stalks” as

$$\begin{aligned} \text{Calla} = & \text{Flower} \sqcap (\exists \text{hasSize. very(Large)}) \sqcap (\exists \text{hasPetalWidth. Long}) \sqcap \\ & \sqcap (\exists \text{hasColour. White}) \sqcap (\exists \text{hasStalks. Thick}), \end{aligned}$$

where *Large*, *Long* and *Thick* are fuzzy concrete predicates.

The interesting point is that according to our semantics, e.g. a minor is *likely* a young person. Indeed, a minor will be a young person with degree at least $(1 - 0.00075 \cdot 18^2) \approx 0.76$. Informally, this value corresponds of the computation of the degree of subsumption between the two defined concepts, i.e. the degree of $\forall x. \text{Minor}(x) \rightarrow \text{YoungPerson}(x)$, which is determined by $\inf_{x \in \text{Natural}} i(\leq_{18}(x), \text{Young}(x))$, where i is an implication function. The fact that, as expected, a minor is a young person (to some degree) is obtained without explicitly mentioning it. This inference cannot be achieved in classical *SHOIN*(D).

Similarly, by referring to Example 1, we will have that the car $\mathbf{tt}$ will be a sports car to a certain degree given by $(0.004 \cdot 243)^2 \approx 0.94$. Therefore, unlike Example 1, $\mathbf{tt}$ is now likely a sports car, *as it should be*.

Concerning concepts and roles, the syntax is as for $\mathcal{SHOIN}(\mathcal{D})$, except that we allow modifiers in concept expressions. That is, if $\mathbf{M}$ is a new alphabet for modifier symbols, $m \in \mathbf{M}$ is a modifier and C is a $\mathcal{SHOIN}(\mathcal{D})$ concept, then $m(C)$ is fuzzy $\mathcal{SHOIN}(\mathcal{D})$ concept as well. For instance, the definition of $\mathbf{SportsCar}$ above involves a modifier. Modifiers are allowed in fuzzy description logics such as [13, 29].

Concerning the axioms, similarly to [26], we introduce fuzzy axioms. For $n \in (0, 1]$,

- a fuzzy *RBox* $\mathcal{R}$ is a finite set of $\mathcal{SHOIN}(\mathcal{D})$ transitivity axioms $\mathbf{trans}(R)$ and *fuzzy role inclusion axioms* of the form $\langle \alpha \geq n \rangle$, $\langle \alpha \leq n \rangle$, $\langle \alpha > n \rangle$ and $\langle \alpha < n \rangle$, where α is a $\mathcal{SHOIN}(\mathcal{D})$ role inclusion axiom;
- a fuzzy *TBox* $\mathcal{T}$ consists of a finite set of *fuzzy concept inclusion axioms* of the form $\langle \alpha \geq n \rangle$, $\langle \alpha \leq n \rangle$, $\langle \alpha > n \rangle$ and $\langle \alpha < n \rangle$ where α is a $\mathcal{SHOIN}(\mathcal{D})$ concept inclusion axiom ($C \sqsubseteq D$);
- a fuzzy *ABox* $\mathcal{A}$ consists of a finite set of *fuzzy concept* and *fuzzy role assertion axioms* of the form $\langle \alpha \geq n \rangle$, $\langle \alpha \leq n \rangle$, $\langle \alpha > n \rangle$, or $\langle \alpha < n \rangle$, where α is a $\mathcal{SHOIN}(\mathcal{D})$ concept or role assertion. As for the crisp case, $\mathcal{A}$ may also contain a finite set of individual (in)equality axioms $a \approx b$ and $a \not\approx b$, respectively.

For instance, $\langle a:C \geq 0.1 \rangle$, $\langle (a,b):R \leq 0.3 \rangle$, $\langle R \sqsubseteq S \geq 0.4 \rangle$, or $\langle C \sqsubseteq D \leq 0.6 \rangle$ are fuzzy axioms. Informally, from a semantics point of view, a fuzzy axiom $\langle \alpha \leq n \rangle$ constrains the membership degree of α to be less or equal to n (similarly for $\geq, >, <$). For instance, $\langle \mathbf{jim:YoungPerson} \geq 0.2 \rangle$, i.e. $\langle \mathbf{jim:Person} \sqcap \exists \mathbf{age.Young} \geq 0.2 \rangle$, dictates that $\mathbf{jim}$ is a $\mathbf{YoungPerson}$ with degree at least 0.2. On the other hand, a fuzzy concept inclusion axiom of the form $\langle C \sqsubseteq D \geq n \rangle$ dictates that the subsumption degree between C and D is at least n . A $\mathcal{SHOIN}(\mathcal{D})$ *fuzzy knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ consists of a fuzzy TBox $\mathcal{T}$, a fuzzy RBox $\mathcal{R}$, and a fuzzy ABox $\mathcal{A}$.

Semantics. The semantics extends [26]. The main idea is that concepts and roles are interpreted as fuzzy subsets of an interpretation's domain. Therefore, $\mathcal{SHOIN}(\mathcal{D})$ axioms, rather being satisfied (true) or unsatisfied (false) in an interpretation, become a degree of truth in $[0, 1]$.

A *fuzzy interpretation* $\mathcal{I}$ with respect to a concrete domain $\mathcal{D}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non empty set $\Delta^{\mathcal{I}}$ (called the *domain*), disjoint from $\Delta_{\mathcal{D}}$, and of a *fuzzy interpretation function* $\cdot^{\mathcal{I}}$ that assigns

- to each abstract concept $C \in \mathbf{C}$ a function $C^{\mathcal{I}}: \Delta^{\mathcal{I}} \rightarrow [0, 1]$;
- to each abstract role $R \in \mathbf{R}_a$ a function $R^{\mathcal{I}}: \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow [0, 1]$;
- to each abstract individual $a \in \mathbf{I}_a$ an element in $\Delta^{\mathcal{I}}$;
- to each concrete individual $c \in \mathbf{I}_c$ an element in $\Delta_{\mathcal{D}}$;
- to each concrete role $T \in \mathbf{R}_c$ a function $R^{\mathcal{I}}: \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}} \rightarrow [0, 1]$;
- to each modifier $m \in \mathbf{M}$ a fixed function $m: [0, 1] \rightarrow [0, 1]$;
- to each n -ary concrete predicate d the fuzzy relation $d^{\mathcal{D}}: \Delta_{\mathcal{D}}^n \rightarrow [0, 1]$.

The mapping $\cdot^{\mathcal{I}}$ is extended to concepts and roles as specified in the following table (where $x, y \in \Delta^{\mathcal{I}}, v \in \Delta_{\mathcal{D}}$):

$$\begin{aligned}
\top^{\mathcal{I}}(x) &= 1 \\
\perp^{\mathcal{I}}(x) &= 0 \\
(C_1 \sqcap C_2)^{\mathcal{I}}(x) &= C_1^{\mathcal{I}}(x) \wedge C_2^{\mathcal{I}}(x) \\
(C_1 \sqcup C_2)^{\mathcal{I}}(x) &= C_1^{\mathcal{I}}(x) \vee C_2^{\mathcal{I}}(x) \\
(\neg C)^{\mathcal{I}}(x) &= \neg C^{\mathcal{I}}(x) \\
(m(C))^{\mathcal{I}}(x) &= \mathbf{m}(C^{\mathcal{I}}(x)) \\
(\forall R.C)^{\mathcal{I}}(x) &= \inf_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x, y) \rightarrow C^{\mathcal{I}}(y) \\
(\exists R.C)^{\mathcal{I}}(x) &= \sup_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x, y) \wedge C^{\mathcal{I}}(y) \\
(\geq n S)^{\mathcal{I}}(x) &= \sup_{y_1, \dots, y_n \in \Delta^{\mathcal{I}}} \bigwedge_{i=1}^n S^{\mathcal{I}}(x, y_i) \\
(\leq n S)^{\mathcal{I}}(x) &= \neg(\geq n+1 S)^{\mathcal{I}}(x) \\
\{a_1, \dots, a_n\}^{\mathcal{I}}(x) &= \bigvee_{i=1}^n a_i^{\mathcal{I}} = x \\
d(v) &= d^{\mathcal{D}}(v) \\
\{c_1, \dots, c_n\}^{\mathcal{I}}(v) &= \bigvee_{i=1}^n c_i^{\mathcal{I}} = v \\
(\forall T_1, \dots, T_n.D)^{\mathcal{I}}(x) &= \inf_{y_1, \dots, y_n \in \Delta_{\mathcal{D}}^{\mathcal{I}}} (\bigwedge_{i=1}^n T_i^{\mathcal{I}}(x, y_i)) \rightarrow D^{\mathcal{I}}(y_1, \dots, y_n) \\
(\exists T_1, \dots, T_n.D)^{\mathcal{I}}(x) &= \sup_{y_1, \dots, y_n \in \Delta_{\mathcal{D}}^{\mathcal{I}}} (\bigwedge_{i=1}^n T_i^{\mathcal{I}}(x, y_i)) \wedge D^{\mathcal{I}}(y_1, \dots, y_n) \\
(S^-)^{\mathcal{I}}(x, y) &= S^{\mathcal{I}}(y, x).
\end{aligned}$$

We comment briefly some points. The semantics of $\exists R.C$

$$(\exists R.C)^{\mathcal{I}}(d) = \sup_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(d, y) \wedge C^{\mathcal{I}}(y)$$

is the result of viewing $\exists R.C$ as the open first order formula $\exists y.F_R(x, y) \wedge F_C(y)$ (where F is the obvious translation of roles and concepts into First-Order Logic -FOL) and the existential quantifier $\exists$ is viewed as a disjunction over the elements of the domain. Similarly,

$$(\forall R.C)^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x, y) \rightarrow C^{\mathcal{I}}(y)$$

is related to the open first order formula $\forall y.F_R(x, y) \rightarrow F_C(y)$, where the universal quantifier $\forall$ is viewed as a conjunction over the elements of the domain. However, as we already pointed out in Section 3.1, unlike the classical case, in general we do not have that $(\forall R.C)^{\mathcal{I}} = (\neg \exists R. \neg C)^{\mathcal{I}}$. If the t-norm and negation are chosen such that $\forall a, b \in [0, 1], i(a, b) = n(t(a, n(b)))$ holds, i.e. in formulae $a \rightarrow b \equiv \neg(a \wedge \neg b)$, then $(\forall R.C)^{\mathcal{I}} = (\neg \exists R. \neg C)^{\mathcal{I}}$ holds.

Another point concerns the semantics of number restrictions. The semantics of the concept $(\geq n S)$

$$(\geq n S)^{\mathcal{I}}(x) = \sup_{y_1, \dots, y_n \in \Delta^{\mathcal{I}}} \bigwedge_{i=1}^n S^{\mathcal{I}}(x, y_i)$$

is the result of viewing $(\geq n S)$ as the open first order formula

$$\exists y_1, \dots, y_n. \bigwedge_{i=1}^n F_S(x, y_i) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \neq y_j.$$

That is, there are at least n distinct elements that satisfy to some degree $F_R(x, y_i)$. This guarantees us that $\exists S.\top \equiv (\geq 1 S)$. The semantics of $(\leq n S)$ is defined in such a way to guarantee the classical relationship $(\leq n S) \equiv \neg(\geq n + 1 S)$.

An alternative definition for the $(\geq n S)$ and the $(\leq n S)$ constructs may rely on the scalar cardinality of a fuzzy set. However, we prefer to stick on the formulation, which derives directly from its FOL translation.

Finally, the mapping $\cdot^{\mathcal{I}}$ is extended to non-fuzzy axioms as specified in the following table (where $a, b \in \mathcal{I}_a$):

$$\begin{aligned} (R \sqsubseteq S)^{\mathcal{I}} &= \inf_{x,y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x, y) \rightarrow S^{\mathcal{I}}(x, y) \\ (T \sqsubseteq U)^{\mathcal{I}} &= \inf_{x,y \in \Delta^{\mathcal{I}}} T^{\mathcal{I}}(x, y) \rightarrow U^{\mathcal{I}}(x, y) \\ (C \sqsubseteq D)^{\mathcal{I}} &= \inf_{x \in \Delta^{\mathcal{I}}} C^{\mathcal{I}}(x) \rightarrow D^{\mathcal{I}}(x) \\ (a:C)^{\mathcal{I}} &= C^{\mathcal{I}}(a^{\mathcal{I}}) \\ ((a, b):R)^{\mathcal{I}} &= R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}). \end{aligned}$$

Note here that e.g. the semantics of a concept inclusion axiom $C \sqsubseteq D$ is derived directly from its FOL translation, which is of the form $\forall x. F_C(x) \rightarrow F_D(x)$. This definition is novel and is clearly different from the approaches in which $C \sqsubseteq D$ is viewed as $\forall x. C(x) \leq D(x)$. This latter approach has the effect that the subsumption relationship is a classical $\{0, 1\}$ relationship, while the former has the advantage that subsumption is determined up to a certain degree in $[0, 1]$.

The notion of *satisfiability* of a fuzzy axiom E by a fuzzy interpretation $\mathcal{I}$, denoted $\mathcal{I} \models E$, is defined as follows: $\mathcal{I} \models \text{trans}(R)$, iff $\forall x, y \in \Delta^{\mathcal{I}}. R^{\mathcal{I}}(x, y) \geq \sup_{z \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x, z) \wedge R^{\mathcal{I}}(z, y)$. $\mathcal{I} \models \langle \alpha \geq n \rangle$, where α is a role inclusion or concept inclusion axiom, iff $\alpha^{\mathcal{I}} \geq n$. Similarly, for the other relations $\leq, <$ and $>$. $\mathcal{I} \models \langle \alpha \geq n \rangle$, where α is a concept or a role assertion axiom, iff $\alpha^{\mathcal{I}} \geq n$. Similarly, for the other relations $\leq, <, >$. Finally, $\mathcal{I} \models a \approx b$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$ and $\mathcal{I} \models a \not\approx b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

For a set of fuzzy axioms $\mathcal{E}$, we say that $\mathcal{I}$ *satisfies* $\mathcal{E}$ iff $\mathcal{I}$ satisfies each element in $\mathcal{E}$. If $\mathcal{I} \models E$ (resp. $\mathcal{I} \models \mathcal{E}$) we say that $\mathcal{I}$ is a *model* of E (resp. $\mathcal{E}$). $\mathcal{I}$ *satisfies* (is a *model* of) a fuzzy knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, denoted $\mathcal{I} \models \mathcal{K}$, iff $\mathcal{I}$ is a model of each component $\mathcal{T}, \mathcal{R}$ and $\mathcal{A}$, respectively. A fuzzy axiom E is a *logical consequence* of a knowledge base $\mathcal{K}$, denoted $\mathcal{K} \models E$ iff every model of $\mathcal{K}$ satisfies E .

Example 2. Let us consider Example 1, where all axioms of the TBox and ABox are asserted with degree 1, i.e. are of the form $\langle \alpha \geq 1 \rangle$. We replace the definition of SportsCar with Definition (3) and replace the assertion involving mgb with

$$\langle \text{mgb:Roadster} \sqcap (\exists \text{maker.}\{\text{mg}\}) \sqcap (\exists \text{speed.} \leq_{170\text{km/h}}) \geq 1 \rangle.$$

Then we have that

$$\begin{aligned} \mathcal{K} \models \langle \text{SportsCar} \sqsubseteq \text{Car} \geq 1 \rangle & & \mathcal{K} \models \langle \text{mgb:SportsCar} \leq 0.46 \rangle \\ \mathcal{K} \models \langle \text{enzo:SportsCar} \geq 1 \rangle & & \mathcal{K} \models \langle \text{tt:SportsCar} \geq 0.94 \rangle. \end{aligned}$$

Note how the maximal speed limit of the mgb car ($\leq_{170\text{km/h}}$) induces an upper limit, 0.46, of the membership degree. Neither this inference is possible in classical $\mathcal{SHOIN}(\mathcal{D})$, nor the one involving tt.

Example 3. Consider the knowledge base $\mathcal{K}$ with Definitions (1) and (2). Then we have that

$$\mathcal{K} \models \langle \text{Minor} \sqsubseteq \text{YoungPerson} \geq 0.76 \rangle ,$$

which is a relationship not captured with classical $\mathcal{SHOIN}(\mathcal{D})$.

Finally, given $\mathcal{K}$ and an axiom α , where α is neither a transitivity axiom, nor an individual (in) equality axiom, it is of interest to compute α 's best lower and upper degree value bounds. The *greatest lower bound* of α w.r.t. $\mathcal{K}$ (denoted $glb(\mathcal{K}, \alpha)$) is

$$glb(\mathcal{K}, \alpha) = \sup\{n: \mathcal{K} \models \langle \alpha \geq n \rangle\} ,$$

while the *least upper bound* of α with respect to $\mathcal{K}$ (denoted $lub(\mathcal{K}, \alpha)$) is

$$lub(\mathcal{K}, \alpha) = \inf\{n: \mathcal{K} \models \langle \alpha \leq n \rangle\} ,$$

where $\sup \emptyset = 0$ and $\inf \emptyset = 1$. Determining the *lub* and the *glb* is called the *Best Degree Bound* (BDB) problem. For instance, the entailments in Examples 2 and 3 are the best possible degree bounds. Furthermore, note that,

$$lub(\Sigma, a:C) = \neg glb(\Sigma, a:\neg C) , \quad (4)$$

i.e. the *lub* can be determined through the *glb* (and vice-versa). Similarly, $lub(\Sigma, (a, b):R) = \neg glb(\Sigma, a:\neg \exists R.\{b\})$ holds. Also, note that, $\Sigma \models \langle \alpha \geq n \rangle$ iff $glb(\Sigma, \alpha) \geq n$, and similarly $\Sigma \models \langle \alpha \leq n \rangle$ iff $lub(\Sigma, \alpha) \leq n$ hold.

Concerning the entailment problem, it is quite easily verified that, as for the crisp case, the entailment problem can be reduced to the unsatisfiability problem:

$$\begin{aligned} \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models \langle \alpha \geq n \rangle & \text{ iff } \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\langle \alpha < n \rangle\} \rangle \text{ is not satisfiable} \\ \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models \langle \alpha \leq n \rangle & \text{ iff } \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\langle \alpha > n \rangle\} \rangle \text{ is not satisfiable .} \end{aligned}$$

Unfortunately, from a computational point of view, no calculus exists yet checking satisfiability of fuzzy $\mathcal{SHOIN}(\mathcal{D})$ knowledge bases. [13, 29] report a calculus for the case of $\mathcal{ALC}$ [24] (with concept constructors $\top, \perp, \neg, \sqcap, \sqcup, \forall, \exists$) with modifiers and simple TBox, with $\min, \max$ and $\rightarrow_{KD}$ connectives. No indication for the BDB problem is given. [25, 26] reports a calculus for $\mathcal{ALC}$ and simple TBox, with $\min, \max$ and $\rightarrow_{KD}$ connectives and addresses the BDB problem and, [27] shows how the satisfiability problem and the BDB problem can be reduced to classical $\mathcal{ALC}$ and, thus, can be resolved by means of a tools like FACT and RACER. However, despite these negative results, recently [28] reports a calculus for $\mathcal{ALC}(\mathcal{D})$ whenever the connectives, the modifiers and the concrete fuzzy predicates are representable as a bounded Mixed Integer Program. For instance, Lukasiewicz logic satisfies these conditions as well as the membership functions for concrete fuzzy predicates we have presented in this paper. Additionally, modifiers should be a combination of linear functions. In that case the calculus consists of a set of constraint propagation rules and an invocation to an oracle for bounded Mixed Integer Programming. But, indeed, the computational aspect is definitely a point that has to be addressed in forthcoming works.

4 Conclusions and Outlook

We have presented a fuzzy extension of $\mathcal{SHOIN}(\mathcal{D})$ showing that its representation and reasoning capabilities go clearly beyond classical $\mathcal{SHOIN}(\mathcal{D})$. Interestingly, we allow modifiers, fuzzy concrete domain predicates and fuzzy axioms to appear in a $\mathcal{SHOIN}(\mathcal{D})$ knowledge base and the entailment and the subsumption relationship hold to a certain degree. To the best of our knowledge, no other work has yet extended the semantics to $\mathcal{SHOIN}(\mathcal{D})$ in such a way. The argument supporting the necessity of such an extension relies on the fact that vague concepts are abundant in human knowledge and, thus, appear *likely* in Web content.

The main direction for future work involves the computational aspect. Currently, we are addressing the fundamental issue to develop a calculus for reasoning within $\mathcal{ALC}(\mathcal{D})$, i.e. $\mathcal{ALC}$ with concrete domains and arbitrary t-norm, co-norm, negation and residuum as implication. We are investigating the possibility to use the methods developed in the context of *Many-Valued Logics* [12], which seem to particularly well-suited to our context. These procedures have then to be combined with a procedure to deal with fuzzy concrete domains, for which we plan to rely on [18].

Another direction is in extending fuzzy $\mathcal{SHOIN}(\mathcal{D})$ with *fuzzy quantifiers*, where the $\forall$ and $\exists$ quantifiers are replaced with fuzzy quantifiers like *most*, *some*, *usually* and the like (see [23] for a preliminary work in this direction). This allows to define concepts like

$$\begin{aligned}\text{TopCustomer} &= \text{Customer} \sqcap (\text{Usually})\text{buys.ExpensiveItem} \\ \text{ExpensiveItem} &= \text{Item} \sqcap \exists \text{price.High} .\end{aligned}$$

Here, the fuzzy quantifier *Usually* replaces the classical quantifier $\forall$ and *High* is a fuzzy concrete predicate.

Fuzzy quantifiers can be applied to inclusion axioms as well, allowing to express, for instance:

$$(\text{Most})\text{Bird} \sqsubseteq \text{FlyingObject} .$$

Here the fuzzy quantifier *Most* replaces the classical universal quantifier $\forall$ assumed in the inclusion axioms. The above axiom allows to state that most birds fly.

Ultimately, we believe that the fuzzy extension of $\mathcal{SHOIN}(\mathcal{D})$ is of great interest to the Semantic Web community, as it allows to express naturally a wide range of concepts of actual domains, for which a classical $\mathcal{SHOIN}(\mathcal{D})$ representation is unsatisfactory.

References

1. Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, 1991.
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. Fahiem Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. The MIT Press, 1990.

4. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *The Scientific American*, 284(5):34–43, 2001.
5. P. Bonatti and A. Tettamanzi. Some complexity results on fuzzy description logics. In A. Petrosino V. Di Gesù, F. Masulli, editor, *WILF 2003 Int. Workshop on Fuzzy Logic and Applications*, LNCS 2955, Berlin, 2004. Springer Verlag.
6. Rita Maria da Silva, Antonio Eduardo C. Pereira, and Marcio Andrade Netto. A system of knowledge representation based on formulae of predicate calculus whose variables are annotated by expressions of a fuzzy terminological logic. In *Proc. of the 5th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-94)*, LNCS 945. Springer-Verlag, 1994.
7. Didier Dubois and Henri Prade. *Fuzzy Sets and Systems*. Academic Press, New York, NJ, 1980.
8. Didier Dubois and Henri Prade. Approximate and commonsense reasoning: From theory to practice. In Zbigniew W. Ras and Michalewicz Maciek, editors, *Proc. of the 9th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-96)*, LNAI 1079, pages 19–33. Springer-Verlag, 1996.
9. Pan *et al.* Specification of coordination of rule and ontology languages. Technical report, Knowledgeweb Network of Excellence, EU-IST-2004-507482, 2004. Deliverable D2.5.1.
10. N. Guarino and R. Poli. Formal ontology in conceptual analysis and knowledge representation. *Int. Journal of Human and Computer Studies*, 43(5/6):625–640, 1995.
11. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of Int. Joint Conf. on Automated Reasoning (IJCAR-01)*, LNAI 2083, pages 701–705, 2001. Springer.
12. Reiner Hähnle and Gonzalo Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware and Soft Computing*, IV(2):69–97, 1997.
13. Steffen Hölldobler, Hans-Peter Störr, and Tran Dinh Khang. A fuzzy description logic with hedges and concept modifiers. In *Proc. of the 10th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, 2004.
14. Ian Horrocks. Using an expressive description logic: Fact or fiction? In *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, 1998.
15. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
16. Ian Horrocks, Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 2004.
17. R. Kruse, E. Schwecke, and J. Heinsohn. *Uncertainty and Vagueness in Knowledge Based Systems*. Springer-Verlag, Berlin, Germany, 1991.
18. C. Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics Volume 4*. King's College Publications, 2003.
19. C. Lutz, F. Wolter, and M. Zakharyashev. A tableau algorithm for reasoning about concepts and similarity. In *Proc. of the 12th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods TABLEUX 2003*, number 2796 in LNAI, Rome, Italy, 2003. Springer.
20. Carsten Lutz. Reasoning with concrete domains. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence*, pages 90–95. Morgan Kaufmann Publishers Inc., 1999.
21. Carsten Lutz. Nexp time-complete description logics with concrete domains. *ACM Trans. Comput. Logic*, 5(4):669–705, 2004.
22. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Los Altos, 1988.
23. D. Sánchez and G.B. Tettamanzi. Generalizing quantification in fuzzy description logics. In *Proc. of the 8th Fuzzy Days in Dortmund*, 2004.
24. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

25. Umberto Straccia. A fuzzy description logic. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, pages 594–599, Madison, USA, 1998.
26. Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.
27. Umberto Straccia. Transforming fuzzy description logics into classical description logics. In *Proc. of the 9th European Conf. on Logics in Artificial Intelligence (JELIA-04)*, LNCS 3229, pages 385–399, 2004. Springer Verlag.
28. Umberto Straccia. Fuzzy description logics with concrete domains. Technical Report 2005-TR-03, Istituto di Scienza e Tecnologie dell’Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 2005.
29. C. Tresp and R. Molitor. A description logic for vague knowledge. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, Brighton (England), August 1998.
30. John Yen. Generalizing term subsumption languages to fuzzy logic. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 472–477, 1991.
31. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Consistent Evolution of OWL Ontologies

Peter Haase¹ and Ljiljana Stojanovic²

¹ Institute AIFB, University of Karlsruhe, Germany

² FZI at the University of Karlsruhe, Germany

pha@aifb.uni-karlsruhe.de, stojanovic@fzi.de

Abstract. Support for ontology evolution is extremely important in ontology engineering and application of ontologies in dynamic environments. A core aspect in the evolution process is the to guarantee consistency of the ontology when changes occur. In this paper we discuss the consistent evolution of OWL ontologies. We present a model for the semantics of change for OWL ontologies, considering structural, logical, and user-defined consistency. We introduce resolution strategies to ensure that consistency is maintained as the ontology evolves.

1 Introduction

Most of the work conducted so far in the field of ontologies has focused on ontology construction issues, which assumes that domain knowledge encapsulated in an ontology does not change over time. However, in a more open and dynamic environment, the domain knowledge evolves continually [5]. These changes include accounting for the modification in the application domain, incorporating additional functionality according to changes in the users' needs, organizing information in a better way, etc.

Ontology evolution can be defined as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes. It is not a trivial process, due to the variety of sources and consequences of changes, it thus cannot be performed manually by the ontology engineer. Therefore, this process needs to be supported by the ontology management system. An important aspect in the evolution process is to guarantee the consistency of the ontology when changes occur, considering the semantics of the ontology change. A formalization of the semantic of change requires a definition of the ontology model together with its change operations, the consistency conditions and rules to enforce these conditions.

There exists a number of languages for ontologies, both proprietary and standards-based. They differ not only in their syntax, but more importantly in their semantics. The OWL ontology language is a standard for representing ontologies on the Web [8]. However, the semantics of change operations for OWL has not been considered so far. In this paper, we focus on the evolution of OWL ontologies. More precisely, we consider the OWL DL language, including sublanguages such as OWL Lite.

The approach presented in this paper builds partly on our previous work in ontology evolution [18], which we adapt towards handling OWL ontologies. The differences are mostly reflected in the ontology consistency definition. As we will show, it does not suffice to define a fixed set of consistency conditions, due to the characteristics of

the various sublanguages and the varying usage contexts. Instead, we define the consistency of OWL ontologies at three different levels: structural, logical, and user-defined consistency.

We further define methods for detecting and resolving inconsistencies in an OWL ontology after the application of a change. Finally, as for some changes there may be several different consistent states of the ontology, we define resolution strategies allowing the user to control the evolution. We exemplarily present resolution strategies for various consistency conditions.

This paper is organized as follows: The ontology evolution process is described in Section 2. In Section 3 we define the notions of ontology, ontology change operations, and the semantics of change. In Sections 4, 5, and 6 we discuss how to detect and resolve structural inconsistency, logical inconsistency and user-defined inconsistency, respectively. Before we conclude, we present an overview of related work.

2 Evolution Process

Ontology evolution can be defined as the timely adaptation of an ontology and consistent management of changes. The complexity of ontology evolution increases as ontologies grow in size, so a structured ontology evolution process is required. We follow the process described in [18]. The process starts with *capturing changes* either from explicit requirements or from the result of change discovery methods. Next, in the *change representation* phase, changes are represented formally and explicitly. *The semantics of change* phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into a consistent state. In the *change propagation* phase all dependent artifacts (ontology instances on the Web, dependent ontologies and application programs using the changed ontology) are updated. During the *change implementation* phase required and induced changes are applied to the ontology in a transactional manner. In the *change validation* phase the user evaluates the results and restarts the cycle if necessary.

In this paper we focus on the semantics of change phase. Its role is to enable the resolution of a given ontology change in a systematic manner by ensuring the consistency of the whole ontology. It is realized through two tasks:

- *Inconsistency Detection*: It is responsible for checking the consistency of an ontology with the respect to the ontology consistency definition. Its goal is to find “parts” in the ontology that do not meet consistency conditions;
- *Change Generation*: It is responsible for ensuring the consistency of the ontology by generating additional changes that resolve detected inconsistencies.

The semantics of change phase of the ontology evolution process is shown in Figure 1. Changes are applied to an ontology in a consistent state (c.f. Change Application in Figure 1), and after all the changes are performed, the ontology must remain consistent (c.f. Change Resolution in Figure 1). This is done by finding inconsistencies in the ontology and completing required changes with additional changes, which guarantee the consistency. Indeed, the updated ontology is not defined directly by applying a re-

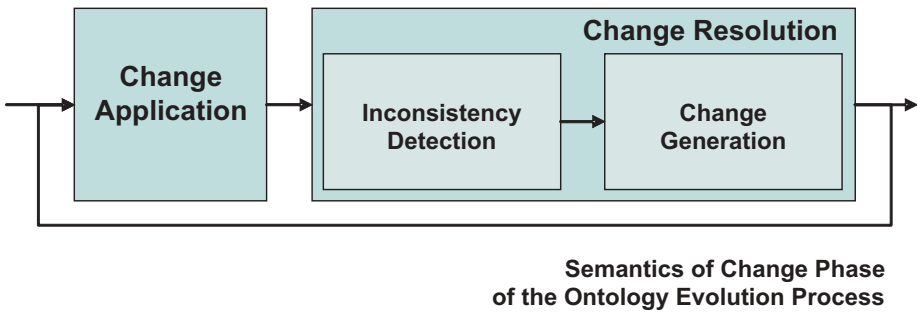


Fig. 1. Semantics of Change Phase

requested change. Instead, it is indirectly characterized as an ontology that satisfies the user's requirement for a change and it is at the same time a consistent ontology.

In this paper we specifically consider the semantics of change phase for OWL DL ontologies. Ontology consistency in general is defined as a set of conditions that must hold for every ontology [18]. Here, we have to distinguish various notions of consistency:

- *Structural Consistency*: First, we have to consider the structural consistency, which ensures that the ontology obeys the constraints of the ontology language with respect to how the constructs of the ontology language are used.
- *Logical Consistency*: Then, we need to consider the formal semantics of the ontology: Viewing the ontology as a logical theory, we consider an ontology as logically consistent if it is satisfiable, meaning that it does not contain contradicting information.
- *User-defined Consistency*: Finally, there may be definitions of consistency that are not captured by the underlying ontology language itself, but rather given by some application or usage context. The conditions are explicitly defined by the user and they must be met in order for the ontology to be considered consistent.

We note that most of the existing evolution systems (including the schema evolution systems as well) consider only the structural consistency. The role of an ontology evolution system is not only to find inconsistencies in an ontology and to alert an ontology engineer about them. Helping ontology engineers notice the inconsistencies only partially addresses the issue. Ideally, an ontology evolution system should be able to support ontology engineers in resolving problems at least by making suggestions how to do that.

Moreover, an inconsistency may be resolved in many ways. In order to help to user to control and customize this process, we have introduced the so-called resolution strategies. Resolution strategies are developed as a method of “finding” a consistent ontology that meets the needs of the ontology engineer. An resolution strategy is the policy for evolution with respect to the his/her requirements. It unambiguously defines the way in which a change will be resolved, i.e. which additional changes will be generated.

In the rest of this paper we formally define different types of consistency and elaborate on how corresponding inconsistencies can be detected and resolved.

3 Ontology Model and Ontology Change Operations

The goal of ontology evolution is to guarantee the correct semantics of ontology changes, i.e. ensuring that they produce an ontology conforming to a set of consistency conditions. The set of ontology change operations – and thus the consistency conditions – depends heavily on the underlying ontology model. Most existing work on ontology evolution builds on frame-like or object models, centered around classes, properties, etc. However, as in this work we focus on the evolution of OWL DL ontologies, we follow the axiom-centered ontology model, heavily influenced by Description Logics. In this section, we will first review the ontology model, define change operations for this model, and describe the semantics of change.

3.1 Ontology Model

OWL DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [7]. Hence, although several XML and RDF syntaxes exist, for convenience we will adhere to the more compact, traditional $\mathcal{SHOIN}(\mathbf{D})$ syntax. For the correspondence between this notation and various OWL DL syntaxes see [7].

We use a datatype theory $\mathbf{D}$, a set of concept names N_C , sets of abstract and concrete individuals N_{I_a} and N_{I_c} , respectively, and sets of abstract and concrete role names N_{R_a} and N_{R_c} , respectively.

The set of $\mathcal{SHOIN}(\mathbf{D})$ *concepts* is defined by the following syntactic rules, where A is an atomic concept, R is an abstract role, S is an abstract simple role, $T_{(i)}$ are concrete roles, d is a concrete domain predicate, a_i and c_i are abstract and concrete individuals, respectively, and n is a non-negative integer:

$$\begin{aligned} C \rightarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n S \mid \leq n S \mid \{a_1, \dots, a_n\} \mid \\ \mid \geq n T \mid \leq n T \mid \exists T_1, \dots, T_n.D \mid \forall T_1, \dots, T_n.D \\ D \rightarrow d \mid \{c_1, \dots, c_n\} \end{aligned}$$

A $\mathcal{SHOIN}(\mathbf{D})$ ontology O is a finite set of axioms of the form¹: concept inclusion axioms $C \sqsubseteq D$, transitivity axioms $\text{Trans}(R)$, role inclusion axioms $R \sqsubseteq S$ and $T \sqsubseteq U$, concept assertions $C(a)$, role assertions $R(a, b)$, individual (in)equalities $a \approx b$, and $a \not\approx b$, respectively. The common distinction between RBox, TBox and ABox is not relevant for this work. We denote the set of all possible ontologies with $\mathcal{O}$.

Example 1. As a running example, we will consider a simple ontology modelling a small research domain, consisting of the following axioms:

$\text{Researcher} \sqsubseteq \text{Person}$, $\text{Student} \sqsubseteq \text{Person}$ (students and researchers are persons), $\text{Article} \sqsubseteq \text{Publication}$ (articles are publications), $\top \sqsubseteq \forall \text{author}^-. \text{Publication}$, $\top \sqsubseteq \forall \text{author}. \text{Person}$, (the domain and range of author are publications, persons, resp.), $\text{Article}(\text{anArticle})$ (anArticle is an article), $\text{Researcher}(\text{peter})$, $\text{Researcher}(\text{ljiljana})$ (peter and ljiljana are researchers), $\text{author}(\text{anArticle}, \text{peter})$, $\text{author}(\text{anArticle}, \text{ljiljana})$ (peter and ljiljana are authors of anArticle).

¹ For the direct model-theoretic semantics of $\mathcal{SHOIN}(\mathbf{D})$ we refer the reader to [9].

3.2 Ontology Change Operation

Based on the ontology model, we can now define ontology change operations.

Definition 1 (Ontology Change Operations). *An ontology change operation $oco \in OCO$ is a function $oco : \mathcal{O} \rightarrow \mathcal{O}$. Here OCO denotes the set of all change operations.*

For the above defined ontology model of $SHOIN(\mathbf{D})$, we allow the atomic change operations of adding and removing axioms, which we denote with α^+ and α^- , respectively. Obviously, representing changes at the level of axioms is very fine-grained. However, based on this minimal set of atomic change operations, it is possible to define more complex, higher-level descriptions of ontology changes. Composite ontology change operations can be expressed as a sequence of atomic ontology change operations. The semantics of the sequence is the chaining of the corresponding functions: For some atomic change operations $oco_1, \dots, oco_n$ we can define $oco_{composite}(x) = oco_n \circ \dots \circ oco_1(x) := oco_n(\dots(oco_1)(x))$.

3.3 Semantics of Change

The semantics of change refers to the effect of the ontology change operations and the consistent management of these changes. The consistency of an ontology is defined in terms of consistency conditions, or invariants that must be satisfied by the ontology. We then define rules for maintaining these consistency conditions by generating additional changes.

Definition 2 (Consistency of an Ontology). *We call an ontology O consistent with respect to a set of consistency conditions $\mathcal{K}$ iff for all $\kappa \in \mathcal{K}$, O satisfies the consistency condition $\kappa(O)$.*

At this point, we do not make any restriction with respect to the representation of the consistency conditions. They may be expressed for example as logical formulas or functions. In the following, we will further distinguish between structural, logical and user-defined consistency conditions: $\mathcal{K}_S$, $\mathcal{K}_L$, and $\mathcal{K}_U$, respectively. We will call an ontology *structurally consistent*, *logically consistent* and *user-defined consistent*, if the respective consistency conditions are satisfied for the ontology.

Change Generation. If we have discovered that an ontology is inconsistent, i.e. some consistency condition is not satisfied, we need to resolve these inconsistencies by generating additional changes that lead to a consistent state. These changes are generated by *resolution functions*:

Definition 3 (Resolution Function). *A resolution function $\varrho \in \mathcal{P}$ is a function $\varrho : \mathcal{O} \times OCO \rightarrow OCO$, which returns for a given ontology and an ontology change operation an additional change operation (which may be composite).*

A trivial resolution function would be a function which for a given ontology and change operation simply returns the inverse operation, which effectively means a rejection of the change. Obviously, for a consistent input ontology, applying a change followed by the inverse change will result in a consistent ontology.

In general, there may be many different ways to resolve a particular inconsistency, i.e. different resolution functions may exist. We can imagine a resolution function that initially generates a set of alternative potential change operations, which may be presented to the user who decides for one of the alternatives. Such a resolution function that depends on some external input is compatible with our definition of a resolution function.

We can now define the notion of a resolution strategy:

Definition 4 (Resolution Strategy). A resolution strategy RS is a total function $RS : \mathcal{K} \rightarrow \mathcal{P}$ that maps each consistency condition to a resolution function. Further we require that for all possible ontologies $O \in \mathcal{O}$ and for all $oco \in OCO$ and all $\kappa \in \mathcal{K}$, the assigned resolution function $\varrho = RS(\kappa)$ generates changes $oco' = \varrho(O, oco)$, which – applied to the ontology $oco'(O)$ – result in an ontology that satisfies the consistency condition κ .

The resolution strategy is applied for each ontology change operation in straightforward manner: As long as there are inconsistencies with respect to a consistency condition, we apply the corresponding resolution function.

Please note that a resolution function may generate changes that violate other consistency conditions (resulting in further changes that in turn may violate the previous consistency condition). When defining a resolution strategy, one therefore has to make sure that the application of the resolution strategy terminates, either by prohibiting that a resolution function introduces inconsistencies with respect to any defined consistency condition, or by other means, such as cycle detection.

In the following chapters we will introduce various evolution strategies to maintain the structural, logical and user-defined consistency of an ontology.

4 Structural Consistency

Structural consistency considers constraints that are defined for the ontology model with respect to the constructs that are allowed to form the elements of the ontology (in our case the axioms). However, in the context of OWL ontologies, there exist various sublanguages (sometimes also called species), such as OWL DL, OWL Lite, OWL DLP [19]. These sublanguages differ with respect to the constructs that are allowed and can be defined in terms of constraints on the available constructs. The role of these sublanguages is to be able to define ontologies that are “easier to handle”, either on a syntactic level to for example allow easier parsing, or on a semantic level to trade some of the expressivity for decreased reasoning complexity. It is thus important that the ontology evolution process provides support for dealing with defined sublanguages: When an ontology evolves, we need to make sure that an ontology does “not leave its sublanguage”.

Because of the variety of the sublanguages, it is not possible to operate with a pre-defined and fixed set of structural consistency conditions. Instead, we allow to define sublanguages in terms of arbitrary structural consistency conditions along with the corresponding resolution functions that ensure that an ontology change operation does not lead out of the defined sublanguage. Please note that because of the definition of the

ontology model, we do not allow to construct ontologies outside of the OWL DL language.

4.1 Structural Consistency Conditions

We will in the following define what it means for an ontology to be structurally consistent with respect to a certain ontology sublanguage. A sublanguage is defined by a set of constraints on the axioms. Typically, these constraints disallow the use of certain constructs or the way these constructs are used.

Some constraints can be defined on a “per-axiom-basis”, i.e. they can be validated for the axioms individually. Other constraints restrict the way that axioms are used in combination. In the following we will show how such consistency conditions can be defined for a particular sublanguage.

Consistency Condition for the OWL Lite sublanguage. OWL Lite is a sublanguage of OWL DL that supports only a subset of the OWL language constructs [2]. We will now show how it can be defined in terms of a set of structural consistency conditions $\mathcal{K}_S$ ²:

- $\kappa_{S,1}$ disallows the use of disjunction $C \sqcup D$,
- $\kappa_{S,2}$ disallows the use of negation $\neg C$,
- $\kappa_{S,3}$ restricts the use of the concept $C \sqcap D$ such that C and D be concept names or restrictions,
- $\kappa_{S,4}$ restricts the use of the restriction constructors $\exists R.C$, $\forall R.C$ such that C must be a concept name,
- $\kappa_{S,5}$ limits the values of stated cardinalities to 0 or 1, i.e. $n \in \{0, 1\}$ for all restrictions $\geq n R$, $\leq n R$,
- $\kappa_{S,6}$ disallows the usage of the oneOf constructor $\{a_1, \dots, a_n\}$.

4.2 Resolving Structural Inconsistencies

Once we have discovered inconsistencies with respect to the defined sublanguage, we have to resolve them. An extreme solution would be to simply remove the axioms that violate the constraints of the sublanguage. This would certainly not meet the expected requirements. A more advanced option is to try to express the invalid axiom(s) in a way that is compatible with the defined sublanguage. In some cases, it may be possible to retain the semantics of the original axioms.

Resolution Strategies for OWL Lite. In the following we will present a possible resolution strategy for the OWL Lite sublanguage by defining one resolution function for each of the above consistency conditions in $\mathcal{K}_S$. Although OWL Lite poses many syntactic constraints on the syntax of OWL DL, it is still possible to express complex descriptions using syntactic workarounds, e.g. introducing new concept names and exploiting the implicit negation introduced by disjointness axioms. In fact, using these techniques,

² Please note that the constraints for the OWL DL language are already directly incorporated into the ontology model itself.

OWL Lite can fully capture OWL DL descriptions, except for those containing individual names and cardinality restrictions greater than 1 [8].

- $\varrho_{s,1}$ replaces all references to a concept $C \sqcup D$ with references to a new concept name $CorD$, and adds the following axiom: $CorD \equiv \neg(\neg C \sqcap \neg D)$,
- $\varrho_{s,2}$ replaces all references to a concept $\neg C$ in an added axiom with references to a new concept name $NotC$, and adds the following two axioms: $C \equiv \exists R.\top$ and $NotC \equiv \forall R.\perp$, where R is a newly introduced role name,
- $\varrho_{s,3}$ replaces all references to a concept C (or D), where C (or D) is not a concept name or restriction, in concepts $C \sqcap D$ with references to a new concept name aC (or aD), and adds the following axiom: $aC \equiv C$ (or $aD \equiv D$),
- $\varrho_{s,4}$ replaces all references to a concept C (where C is not a concept name) in restrictions $\exists R.C$ or $\forall R.C$ with references to a new concept name aC , and adds the following axiom: $aC \equiv C$.

While these first four resolution functions simply apply syntactic tricks while preserving the semantics, there exist no semantics-preserving resolution functions for the consistency conditions $\kappa_{S,5}$ and $\kappa_{S,6}$.

However, we can either try to approximate the axioms, or in the worst case, simply remove them to ensure structural consistency. We can thus define:

- $\varrho_{s,5}$ replaces all cardinality restrictions $\geq n R$ with restrictions $\geq 1 R$ and removes all axioms containing cardinality restrictions $\leq n R$,
- $\varrho_{s,6}$ replaces all occurrences of the concept $\{a_1, \dots, a_n\}$ with a new concept D and adds assertions $D(a_1), \dots, D(a_n)$.

Example 2. Suppose we wanted to add to the ontology from Example 1 the axiom $Publication \sqsubseteq \exists author.\neg Student$, i.e. stating that all publications must have an author who is not a student. As this axiom violates consistency condition $\kappa_{S,2}$, resolution function $\varrho_{s,2}$ would generate a composite change that adds the following semantically equivalent axioms instead: $Publication \sqsubseteq \exists author.NotStudent$, $Student \equiv \exists R.\top$, $NotStudent \equiv \forall R.\perp$, resulting in a structurally consistent ontology.

5 Logical Consistency

While the structural consistency is only concerned about whether the ontology conforms to certain structural constraints, the logical consistency addresses the question whether the ontology is “semantically correct”, i.e. does not contain contradicting information.

5.1 Definition of Logical Consistency

The semantics of the $\mathcal{SHOIN}(\mathbf{D})$ description logic is defined via a model-theoretic semantics, which explicates the relationship between the language syntax and the model of a domain: An interpretation $I = (\Delta^I, \cdot^I)$ consists of a domain set Δ^I , disjoint

from the datatype domain Δ_D^I , and an interpretation function $\cdot^I$, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively³. An interpretation $\mathcal{I}$ satisfies an ontology O , if it satisfies each axiom in O . Axioms thus result in semantic conditions on the interpretations. Consequently, contradicting axioms will allow no possible interpretations.

We can thus define a consistency condition for *logical consistency* κ_L that is satisfied for an ontology O if O is satisfiable, i.e. if O has a model. Please note, that because of the monotonicity of the logic, an ontology can only become inconsistent by adding axioms: If a set of axioms is satisfiable, it will still be satisfiable when any axiom is deleted. Therefore, we only need to check the consistency for ontology change operations that add axioms to the ontology.

Example 3. Suppose, we start out with the ontology from our Example 4.2, i.e. the initial example extended with the axiom $Student \sqsubseteq \neg Researcher$ (Students and Researchers are disjoint). This ontology is logically consistent.

Suppose we now wanted to add the axiom $Student(peter)$, stating that the individual *peter* is a student. Obviously, this ontology change operation would result in an inconsistent ontology, as we have stated that students and researchers are disjoint on the one hand, and that *peter* is a student and a researcher on the other hand.

Now, there may be many ways how to resolve this inconsistency. One possibility would be to reject the change $Student(peter)$. Alternatively, we could also remove the assertion $Researcher(peter)$. However, if both of these assertions are correct, the user may not be happy with either decision. The most intuitive one may be to retract the axiom $Student \sqsubseteq \neg Researcher$, but also this may not satisfy the user. A further, more complex change, would be to introduce a new concept *PhdStudent*, which need not be disjoint with researchers.

5.2 Resolving Logical Inconsistencies

In the following, we will present resolution functions that will allow us to define resolution strategies to ensure logical consistency. The goal of these resolution functions is to determine a set of axioms to remove to obtain a logically consistent ontology with “minimal impact” on the existing ontology. Obviously, the definition of minimal impact may depend on the particular user requirements. A very simple definition could be that the number of axioms to be removed should be minimized. More advanced definitions could include a notion of confidence or relevance of the axioms. Based on this notion of “minimal impact” we can define an algorithm that generates a minimal number of changes that result in a maximal consistent subontology.

However, in many cases it will not be feasible to resolve logical inconsistencies in a fully automated manner. We therefore also present a second, alternative approach for resolving inconsistencies that allows the interaction of the user to determine which changes should be generated. Unlike the first approach, this approach tries to localize the inconsistencies by determining a minimal inconsistent subontology.

³ For a complete definition of the interpretation, we refer the reader to [7].

Alternative 1: Finding a Consistent Subontology. In our model we assume that the ontology change operations should lead from one consistent ontology to another consistent ontology. If an ontology change operation (adding an axiom, α^+) would lead to an inconsistent ontology, we need to resolve the inconsistency by finding an appropriate subontology $O' \subset O$ (with $\alpha \in O'$) that is consistent. We do this by finding a maximal consistent subontology:

Definition 5 (Maximal consistent subontology). *An ontology O' is a maximal consistent subontology of O , if $O' \subseteq O$ and O' is logically consistent and every O'' with $O' \subset O'' \subseteq O$ is logically inconsistent.*

Intuitively, this definition states that no axiom from O can be added to O' without losing consistency. In general, there may be many maximal consistent subontologies O' . It is up to the resolution strategy and the user to determine the appropriate subontology to be chosen.

The main idea is that we start out with the inconsistent ontology $O \cup \{\alpha\}$ and iteratively remove axioms until we obtain a consistent ontology. Here, it is important how we determine which axioms should be removed. This can be realized using a *selection function*. The quality of the selection function is critical for two reasons: First, as we potentially have to search all possible subsets of axioms in O for the maximal consistent ontology, we need to prune the search space by trying to find the *relevant* axioms that cause the inconsistency. Second, we need to make sure that we remove the *dispensable* axioms. (Please note that a more advanced strategy could consider to only remove parts of the axiom.)

The first problem of finding the axioms that cause the inconsistency can be addressed e.g. using a notion of syntactic relevance by analyzing how the axioms are structurally connected:

We can realize a selection function based on *structural connectedness*:

Definition 6 (Connectedness). *Given a set of axioms O , two axioms α and β are directly structurally connected – denoted with $\text{connected}(\alpha, \beta) -$, if there exists an ontology entity $e \in N_C \cup N_{I_a} \cup N_{I_c} \cup N_{R_a} \cup N_{R_c}$ that occurs in both α and β .*

The second problem of only removing dispensable axioms requires more semantic selection functions. These semantic selection functions can for example exploit information about the confidence in the axioms that allows us to remove less probable axioms. Such information is for example available in probabilistic ontology models, such as [4], but will not be considered in this paper.

In the following, we present an algorithm (c.f. Algorithm 1) for finding (at least) one maximal consistent subontology using the definition of structural connectedness (c.f. Definition 6): We maintain a set of possible candidate subontologies Ω , which initially contains only $O \cup \{\alpha\}$ (c.f. line 1), i.e. the consistent ontology O before the change and the added axiom α . In every iteration, we generate a new set of candidate ontologies (line 3) by removing one axiom β_1 from each candidate ontology (line 7) that is structurally connected with α or an already removed axiom (in $O \setminus O'$, line 6), until at least one of the candidate ontologies is a consistent subontology (line 12). The termination is guaranteed based on the fact that once we have removed all axioms from $O \cup \{\alpha\}$ that are transitively connected with α , the ontology again must be consistent

Algorithm 1 Determine consistent subontology for adding axiom α to ontology O

```

1:  $\Omega := \{O \cup \{\alpha\}\}$ 
2: repeat
3:    $\Omega' := \emptyset$ 
4:   for all  $O' \in \Omega$  do
5:     for all  $\beta_1 \in O' \setminus \{\alpha\}$  do
6:       if there is a  $\beta_2 \in (\{\alpha\} \cup (O \setminus O'))$  such that  $\text{connected}(\beta_1, \beta_2)$  then
7:          $\Omega' := \Omega' \cup \{O' \setminus \{\beta_1\}\}$ 
8:       end if
9:     end for
10:   end for
11:    $\Omega := \Omega'$ 
12: until there exists an  $O' \in \Omega$  such that  $O'$  is consistent

```

(provided that α itself is consistent and O was consistent before adding α). As we remove exactly one axiom from each candidate ontology in one iteration, the resulting ontology will not only be maximal with respect to the above definition, but also maximal with respect to cardinality, i.e. the number of axioms in the ontology.

The corresponding resolution function $\varrho_{L,1}$ thus generates changes that remove the minimal set of axioms to ensure consistency: $O \setminus O'$, where O' is the maximal consistent ontology.

Alternative 2: Localizing the Inconsistency. In the second alternative, we do not try to find a consistent subontology, instead we try to find a minimal inconsistent ontology, i.e. a minimal set of contradicting axiom. We call this process *Localizing the inconsistency*. Once we have localized this minimal set, we present it to the user. Typically, this set is considerably smaller than the entire ontology, such that it will be easier for the user to decide how to resolve the inconsistency.

Definition 7 (Minimal inconsistent subontology). *An ontology O' is a minimal inconsistent subontology of O , if $O' \subseteq O$ and O' is inconsistent and for all O'' with $O'' \subset O' \subseteq O$, O'' is consistent.*

Intuitively, this definition states that the removal of any axiom from O' will result in a consistent ontology.

Again using the definition of connectedness, we can realize an algorithm (c.f. Algorithm 2) that is guaranteed to find a minimal inconsistent ontology: We maintain a set Ω with candidate ontologies, which initially only consists of the added axiom $\{\alpha\}$ (c.f. line 1). As long as we have not found an inconsistent subontology, we add one structurally connected axiom (line 6) to each candidate ontology (line 7).

Because of the minimality of the obtained inconsistent ontology, it is sufficient to remove any of the axioms to resolve the inconsistency. The minimal inconsistent ontology can be presented to the user, who can select the appropriate axiom to remove.

It may be possible that one added axiom introduced multiple inconsistencies. For this case, the above algorithm has to be applied iteratively.

Algorithm 2 Localize inconsistency introduced by adding axiom α to ontology O

```

1:  $\Omega := \{\{\alpha\}\}$ 
2: repeat
3:    $\Omega' := \emptyset$ 
4:   for all  $O' \in \Omega$  do
5:     for all  $\beta_1 \in O \setminus O'$  do
6:       if there is a  $\beta_2 \in O'$  such that connected( $\beta_1, \beta_2$ ) then
7:          $\Omega' := \Omega' \cup \{O' \cup \{\beta_1\}\}$ 
8:       end if
9:     end for
10:  end for
11:   $\Omega := \Omega'$ 
12: until there exists an  $O' \in \Omega$  such that  $O'$  is inconsistent

```

Example 4. We will now show how Algorithm 2 can be used to localize the inconsistency in our running example, which has been introduced by adding the axiom α *Student*(peter). Applying the algorithm, we start out with the candidate ontology $\Omega := \{\{\textit{Student}(\textit{peter})\}\}$. Adding the structurally connected axioms, we obtain:

$\Omega := \{\{\textit{Student}(\textit{peter}), \textit{Researcher}(\textit{peter})\}, \{\textit{Student}(\textit{peter}), \textit{Student} \sqsubseteq \textit{Person}\}, \{\textit{Student}(\textit{peter}), \textit{Student} \sqsubseteq \neg \textit{Researcher}\}, \{\textit{Student}(\textit{peter}), \textit{Student}(\textit{ljljljana})\}, \{\textit{Student}(\textit{peter}), \textit{author}(\textit{anArticle}, \textit{peter})\}\}$. All of these candidate ontologies are still consistent. In the next iteration, adding the structurally connected axiom *Student* $\sqsubseteq \neg \textit{Researcher}$ to the candidate ontology $\{\textit{Student}(\textit{peter}), \textit{Researcher}(\textit{peter})\}$ will result in the minimal inconsistent subontology $\{\textit{Student}(\textit{peter}), \textit{Researcher}(\textit{peter}), \textit{Student} \sqsubseteq \neg \textit{Researcher}\}$.

The removal of any of these axioms (which one is to be decided by the user), will lead to a consistent ontology.

6 User-Defined Consistency

The user-defined consistency takes into account particular user requirements that need to be expressed “outside” of the ontology language itself. While an ontology may be structurally consistent (e.g. be a syntactically correct ontology according to a particular OWL sublanguage) and may be logically consistent, it may still violate some user requirements. We can identify two types of user-defined consistency conditions: *generic* and *domain dependent*.

Generic consistency conditions are applicable across domains and represent e.g. best design practice or modeling quality criteria. For example, OntoClean [20] formalizes a set of meta-properties representing the philosophical notions of *rigidity*, *identity*, *unity*, and *dependence*. These meta-properties are assigned to properties (corresponding to concepts in DL terminology) of the ontology. Constraints on the taxonomic relationships define the consistency of the ontology, e.g. a non-rigid property cannot subsume a rigid property.

Domain dependent consistency conditions take into account the semantics of a particular formalism of the domain. An example are consistency conditions for the OWL-S process model [17] to verify web service descriptions.

In the following we exemplarily show how user-defined consistency conditions and corresponding resolutions function can be described to ensure *modeling quality conditions*. Such modeling quality conditions cover redundancy, misplaced properties, missing properties, etc. We refer the reader to [18] for a complete reference.

One example of redundancy is *concept hierarchy redundancy*. If a direct super-concept of a concept can be reached through a non-direct path, then the direct link is redundant. We can thus define a consistency condition that disallows concept hierarchy redundancy: $\kappa_{U,1}$ is satisfied if for all axioms $C_1 \sqsubseteq C_n$ in O there exist no axioms in O with $C_1 \sqsubseteq C_2, \dots, C_{n-1} \sqsubseteq C_n$. We can further define a corresponding resolution function $\varrho_{U,1}$ that ensures this consistency condition by generating a change operation that removes the redundant axiom $C_1 \sqsubseteq C_n$.

Example 5. Suppose, we start out with the ontology from our Example 4.2, i.e. the initial example extended with the axiom $Professor \sqsubseteq Person$ (a professor is a person). This ontology is consistent with respect to the consistency definition $\kappa_{U,1}$.

Suppose we now want to add the axiom $Professor \sqsubseteq Researcher$, stating that the a professor is a researcher. Obviously, this ontology change operation would result in an ontology that is inconsistent with respect to $\kappa_{U,1}$ since there is an alternative path (through the concept *Researcher*) between the concept *Professor* and its direct super-concept *Person*. The resolution function $\varrho_{U,1}$ would generate a change operation that removes the axiom $Professor \sqsubseteq Person$.

7 Related Work

In the last decade there has been very active research in the area of ontology engineering. The majority of research studies in this area are focused on construction issues. However, coping with the changes and providing maintenance facilities require a different approach. There are a very few approaches investigating the problems of inducing changes in ontologies.

[18] defines an ontology evolution process which we have adapted for our work. However, the semantics of change in [18] focuses on the KAON ontology model, which is fundamentally different from the OWL ontology model, as described earlier. A taxonomy of ontology changes for the OWL ontologies can be found in [10]. However, in [10] the ontology model follows a more object-oriented view, whereas we follow the axiomatic ontology model of [14].

While there exist significant differences between schema evolution and ontology evolution, as elaborated in [13], particular aspects of schema evolution in databases are relevant for our work. [16] provides an excellent survey of the main issues concerned. A sound and complete axiomatic model for dynamic schema evolution in object-based systems is described in [15]. This is the first effort in developing a formal basis for the schema evolution research. The authors define consistency of a schema with a fixed set of invariants or consistency conditions that are tailored to the data model.

However, in the context of OWL ontologies, the notion of consistency is much more multifaceted. First, the existing work only considers structural consistency. Not only is the set of structural constraints different due to the difference in the underlying models. We further support the evolution of various fragments (sublanguages) of OWL that are defined using different structural constraints. Furthermore, we consider the notions of logical and user-defined consistency.

Regarding the notion of logical consistency, the research done in belief revision is of interest: Here, the revision problem is concerned about resolving contradictions by minimal mutilation of a set of beliefs. The combination of classical approaches with description logics is subject of ongoing research [6].

Finally, there are several tools that support species validation (corresponding to our structural consistency) or localizing inconsistencies in ontologies. For example, the OWL Protege Plugin [11] provides species validation including explanations where certain problems occurred. The OWL Protege Plugin also provides explanations on ontology changes, i.e. new subsumptions that have been inferred, logical inconsistencies that have been introduced (based on RACER reasoning services). [1] presents a “symptom” ontology describing inconsistencies and errors in ontologies. It provides various levels of severity provides a classification of inconsistencies. However, there is no support for preserving consistency in the case that consistency conditions are violated in the presence of ontology changes.

8 Conclusion and Outlook for Future Work

In this paper we have presented an approach to formalize the semantics of change for the OWL ontology language (for OWL DL and sublanguages in particular), embedded in a generic process for ontology evolution. Our formalization of the semantics of change allows to define arbitrary consistency conditions – grouped in structural, logical, and user-defined consistency – and to define resolution strategies that assign resolution functions to that ensure these consistency conditions are satisfied as the ontology evolves. We have shown exemplarily, how such resolution strategies can be realized for various purposes.

The methods described in the previous sections have been implemented on top of KAON2⁴, an ontology management system and inference engine that supports a large subset of OWL DL. The implementation includes evolution strategies for various fragments of ontology languages, including OWL DL, OWL Lite, as well evolution strategies for logical consistency. Additionally it allows to plug-in further evolution strategies for structural consistency (to support additional sublanguages), logical consistency, and user-defined consistency. The approach will be evaluated in the context of various ongoing research projects, including SEKT and OntoGov.

In the future we plan to describe an ontology definition meta-model (ODM) in the style of [3] on top of our axiomatic ontology model and to define the semantics of change for the ODM to support users and applications that prefer a more object-oriented-like ontology model.

⁴<http://kaon2.semanticweb.org/>

Acknowledgments

Research reported in this paper has been partially financed by the EU in the IST project SEKT (IST-2003-506826) (<http://www.sekt-project.com/>) and IST project OntoGov (IST-2002-507237) (<http://www.ontogov.com/>). We would like to thank our colleagues for fruitful discussions.

References

1. Kenneth Baclawski, Christopher J. Matheus, Mieczyslaw M. Kokar, Jerzy Letkowski, and Paul A. Kogut. Towards a symptom ontology for semantic web applications. In McIlraith et al. [12], pages 650–667.
2. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. <http://www.w3.org/TR/owl-ref/>.
3. Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual modeling of owl dl ontologies using uml. In McIlraith et al. [12], pages 198–213.
4. Zhongli Ding and Yun Peng. A Probabilistic Extension to Ontology Language OWL. In *Proceedings of the 37th Hawaii International Conference On System Sciences (HICSS-37)*., Big Island, Hawaii, January 2004.
5. D. Fensel. Ontologies: dynamics networks of meaning. In *Proceedings of the 1st Semantic web working symposium*, Stanford, CA, USA, 2001.
6. Giorgos Flouris. Belief change in arbitrary logics. In *HDMS*, 2004.
7. I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4), 2004.
8. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1), 2003.
9. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
10. Michel Klein. *Change Management for Distributed Ontologies*. PhD thesis, Free University of Amsterdam, 2004.
11. Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In McIlraith et al. [12], pages 229–243.
12. Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors. *The Semantic Web - ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*. Springer, 2004.
13. N. F. Noy and M. Klein. Ontology evolution: not the same as schema evolution. In *SMI technical report SMI-2002-0926*, 2002.
14. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. <http://www.w3.org/TR/2004/REC-owl-semantics-20040210/>.
15. Randel J. Peters and M. Tamer Özsu. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Trans. Database Syst.*, 22(1):75–114, 1997.
16. John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

17. L. Stojanovic, A. Abecker, N. Stojanovic, and R. Studer. On managing changes in the ontology-based e-government. In *Proceedings of the 3rd International Conference on Ontologies, Databases and Application of Semantics (ODBASE 2004)*, number 3291 in Lecture Notes in Computer Science, pages 1080–1097, Agia Napa, Cyprus, November 2004. Springer Verlag.
18. Ljiljana Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.
19. Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, University of Karlsruhe, 2004.
20. Christopher A. Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data Knowledge Engineering*, 39(1):51–74, 2001.

Extending HCONE-Merge by Approximating the Intended Meaning of Ontology Concepts Iteratively

George A. Vouros and Konstantinos Kotis

Department of Information & Communications Systems Engineering,
University of the Aegean, Karlovassi, Samos,
83200, Greece
{georgev, kkot}@aegean.gr

Abstract. A central aspect of HCONE-merge is the mapping of ontology concepts to a hidden intermediate ontology by uncovering the intended meaning of concepts. Such a mapping is realized by a semantic morphism from ontology concepts to WordNet senses. Extending methods that have already been proposed, this paper proposes an iterative algorithm for approximating the intended meanings of ontology concepts in a fully automated way. Results from numerous experiments are thoroughly described and conclusions are drawn.

1 Introduction

Ontologies have been realized as the key technology to shaping and exploiting information for the effective management of knowledge and for the evolution of the Semantic Web and its applications. Ontologies establish a common vocabulary for community members to interlink, combine, and communicate knowledge shaped through practice and interaction, binding the knowledge processes of creating, importing, capturing, retrieving, and using knowledge. However, it seems that there will always be more than one ontology even for the same domain [1]. In distributed settings, where different conceptualizations of the same domain exist, information services must effectively answer queries bridging the gaps between conceptualizations of the same domain. Towards this target, networks of semantically related information must be created at-request. Therefore, coordination (i.e. mapping, alignment, merging) of ontologies is a major challenge for bridging the gaps between agents (software and human) with different conceptualizations.

There are many works towards the mapping/merging of ontologies (e.g. [2] [3] [4] [5] [6]). These works exploit linguistic, structural, domain knowledge and matching heuristics. Recent approaches aim to exploit all types of knowledge and further capture the intended meanings of terms by means of heuristic rules [2]. The HCONE-merge approach to the merging of ontologies [7] [8] exploits linguistic, structural and semantic knowledge and gives much emphasis on “uncovering” the intended informal interpretations of concepts specified in an ontology. Linguistic and structural knowledge about ontologies is exploited by the Latent Semantics Indexing method (LSI) [9] for associating concepts to their informal, human-oriented intended

interpretations realized by WordNet senses. Using concepts' intended interpretations, the proposed mapping/merging method translates formal concept definitions to a common vocabulary and merges the translated definitions by means of description logics' reasoning services.

The HCONE-merge approach, as it was originally proposed, requires humans to validate the interpretations suggested by LSI for every term in the ontology. Since this process is quite frustrating and error-prone, even for small ontologies, we contact research towards minimizing the required human involvement for mapping concepts to their intended interpretations. The ultimate achievement would be to fully automate the mapping of concepts to their intended interpretations, and consequently to fully automate merging. Towards this goal, we have developed techniques and heuristics for ontology mapping and merging that require varying degrees of human involvement [8]. Although we managed to achieve a high degree of precision, this has been gained with the cost of considerable human involvement in the process.

Based on the HCONE method, the present paper proposes an iterative and automatic method for computing the mapping of concepts to their intended informal interpretations. This method requires no human involvement and, as experiments show, it converges to a set of mappings with high precision.

Section 2 of the paper provides definitions of notions used throughout the paper. Section 3 gives an overview of the HCONE-merge method and of research towards automating the computation of the mapping of ontology concepts to their intended meaning. Section 4 describes the iterative approximation of intended interpretations and section 5 provides results of experiments conducted. Section 6 concludes the paper.

2 Background Definitions

An ontology is considered to be a pair $O=(S, A)$, where S is the ontological signature describing the vocabulary (i.e. the terms that lexicalize concepts and relations between concepts) and A is a set of ontological axioms, restricting the intended interpretations of the terms included in the signature. In other words, A includes the formal definitions of concepts and relations that are lexicalized by natural language terms in S .

Ontology mapping from ontology $O_1 = (S_1, A_1)$ to $O_2 = (S_2, A_2)$ is considered to be a morphism $f: S_1 \rightarrow S_2$ of ontological signatures such that $A_2 \models f(A_1)$, i.e. all interpretations that satisfy O_2 's axioms also satisfy O_1 's translated axioms [3] [12]. Consider for instance the ontologies depicted in Fig. 1: Given the morphism f such that $f(\text{Infrastructure}) = \text{Facility}$ and $f(\text{Transportation}) = \text{Transportation System}$, it is true that $A_2 \models \{f(\text{Transportation}) \sqsubseteq f(\text{Infrastructure})\}$, therefore f is a mapping. Given the morphism f' , such that $f'(\text{Infrastructure}) = \text{Transportation System}$ and $f'(\text{Transportation}) = \text{Transportation Means}$, it is not true that $A_2 \models \{f'(\text{Transportation}) \sqsubseteq f'(\text{Infrastructure})\}$, therefore f' is not a mapping.

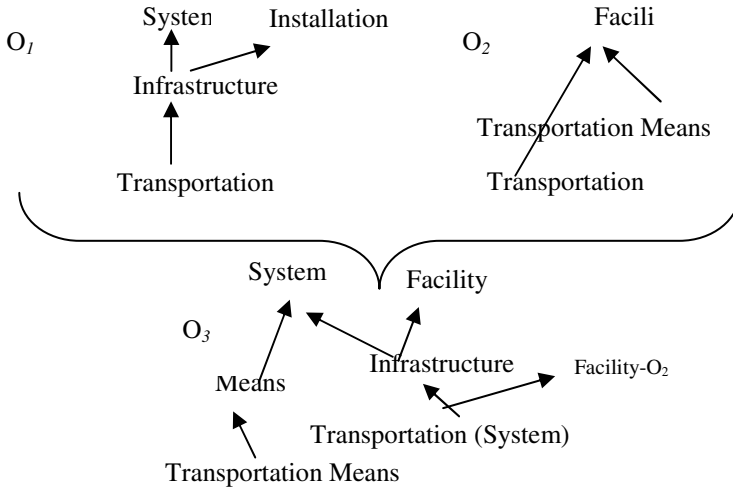


Fig. 1. Example Ontologies

However, instead of a function, we may articulate a set of binary relations between the ontological signatures. Such relations can be the inclusion ($\sqsubseteq$) and the equivalence ($\equiv$) relations. For instance, given the ontologies in Fig. 1, we can say that *Transportation* $\equiv$ *Transportation System*, *Installation* $\equiv$ *Facility* and *Infrastructure* $\sqsubseteq$ *Facility*. Then we have indicated an alignment of the two ontologies and we can merge them. Based on the alignment, the merged ontology will be ontology O_3 in Fig. 1. It holds that $A_3 \neq A_2$ and $A_3 \neq A_1$.

Looking at Fig. 1 in an other way, we can consider O_3 to be part of a larger intermediary ontology and define the alignment of ontologies O_1 and O_2 by means of morphisms $f_1: S_1 \rightarrow S_3$ and $f_2: S_2 \rightarrow S_3$. Then, the merging of the two ontologies is the minimal union of ontological vocabularies and axioms with respect to the intermediate ontology where ontologies have been mapped.

In the example of Fig.1, concepts *Transportation- O_1* and *Transportation System- O_2* will be found to have the same intended meaning, and therefore will be considered equivalent. The merging of their formal definitions will eventually result to:

$$Transportation\ System-O_2 \sqsubseteq Infrastructure-O_1 \sqcap Facility-O_2$$

However, the description logics classification mechanism will consider the axiom *Transportation System- $O_2 \sqsubseteq Facility-O_2$* to be redundant (see Fig. 1). Therefore O_3 will eventually contain only the axiom *Transportation System $\sqsubseteq$ Infrastructure*. Doing so, the merged ontology contains only the minimal set of axioms resulting from source ontologies mapping.

The ontologies merging problem (OMP) can be stated as follows: *Given two ontologies find an alignment between these two ontologies, and then, get the minimal union of their (translated) vocabularies and axioms with respect to their alignment.*

3 HCONE-Merge

The HCONE-merge method finds a morphism between each of the two original ontologies and the so-called “hidden intermediate” ontology. As it is shown in Fig. 2, where the overall method is depicted, WordNet plays the role of an “intermediate”. We consider that each sense in a WordNet synset describes a concept. WordNet senses are related among themselves via the inclusion (hyponym – hyperonym) relation. Terms that lexicalize the same concept (sense) are considered to be equivalent through the synonym relation.

Ontology concepts are being mapped to WordNet senses. This mapping indicates the informal intended interpretations of concepts and it is specified by the semantic morphism (*s-morphism*, symbolized by f_s). Using this mapping, HCONE-merge constructs the intermediate ontology that includes (a) a vocabulary with the lexicalizations of the specific senses of WordNet synsets corresponding to the ontologies’ concepts, and (b) axioms that are the translated axioms of the original ontologies. Having specified the mappings to the hidden intermediate ontology, the translated ontologies are merged following merging actions such as rename, merge, and classify.

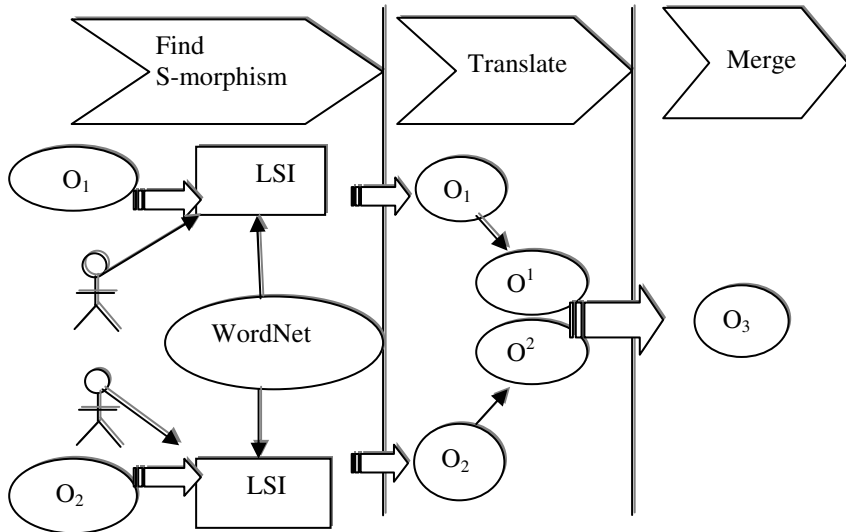


Fig. 2. The HCONE approach towards the OMP

It must be noticed that we do not consider WordNet to include any intermediate ontology, as this would be very restrictive for the specification of the original ontologies (i.e. the method would work only for those ontologies that preserve the inclusion relations among WordNet senses).

The computation of the semantic morphism is based on the lexical semantic indexing (LSI) method.

LSI [9] is a vector space technique for information retrieval and indexing. It assumes that there is an underlying latent semantic structure that it estimates using a matrix of term-document association data by means of statistical techniques. In our case the $n \times m$ space comprises the n more frequently occurred terms of the m WordNet senses the algorithm focuses on. Lexical Semantic Analysis (LSA) allows the arrangement of the semantic space to reflect the major associative patterns in the data. As a result, terms that did not actually appear in a sense may still end up close to the sense, if this is consistent with the major patterns of association in the data. Position in the space then serves as the new kind of semantic indexing.

Given a query (which in our case corresponds to an ontology concept), retrieval aims to locate a point in space that is close to the sense that expresses the intended meaning of this concept. The query to the retrieval mechanism is constructed by the concept names and the associated senses of all concepts in the vicinity of the given concept. The steps of the algorithm for finding the semantic morphism are shown in Fig. 3.

1. Choose a concept from the ontology. Let C be the concept name.
2. Get all WordNet senses $S_1, S_2, \dots, S_m$, lexicalized by C' , where C' is a linguistic variation of C . These senses provide the *focus of the algorithm for C*.
3. Get the hyperonyms and hyponyms of all C' senses.
4. Build the “*semantic space*”: An $n \times m$ matrix that comprises the n more frequently occurred terms in the *vicinity* of the m WordNet senses found in step 2.
5. Build a query string using the terms in the *vicinity* of C . The query string is a sequence of digits, each digit taking value 0 if a term in the *vicinity* of C does not exist in the set of n , and 1 if a query term exists in the set of n .
6. Find the ranked associations between C and C' senses by running the Latent Semantics Analysis (LSA) function and consider the association with the highest grade. LSA uses the query terms for constructing the query string and computes a point in the semantic space constructed in step (4).

Fig. 3. The algorithm for computing the s-morphism

The semantic space is constructed by terms in the vicinity of the senses $S_1, S_2, \dots, S_m$ that are in focus of the algorithm for a concept C . Therefore, we have to decide what constitutes the vicinity of a sense for the calculation of the semantic space. In an analogous way we have to decide what constitutes the vicinity of an ontology concept for the calculation of the query string.

Information that can be included in the semantic space includes:

- The term C' that corresponds to C . C' is a lexical entry in WordNet
- Terms that appear in C' WordNet senses
- Terms that constitute hyperonyms / hyponyms of each C' sense.
- Terms that appear in hyper(hyp)onyms of C' senses

Information that can be included in the query for a concept C includes:

- Concept's C primitive super-concepts.
- Concepts that are immediate super-concepts of C
- Concepts that are immediate sub-concepts of C
- Concepts that are related to C via domain specific relations
- The most frequent terms in WordNet senses that have been associated with the concepts directly related to C via inclusion and equivalence relations.

Formally, given an ontology concept C , the vicinity V_C of this concept includes a set of tuples (C', S') , where C' is the lexicalization of a concept directly related to C and S' is the WordNet sense that has been associated with this concept, or “null” in case there is no associated sense. Therefore, given a concept C and its vicinity V_C , the semantic morphism f_s computes S_C , which is the highest-ranked WordNet sense associated to C i.e. $f_s(C, V_C) = S_C$ where $V_C = \{(C_i, S_i) | C_i \text{ is in the vicinity of } C, \text{ and } f_s(C_i, V_i) = S_i \text{ where } V_i \text{ is the vicinity of } C_i\}$ S_C is assumed to express the intended interpretation of the concept specification.

Using the algorithm in Fig. 3 for the concepts in an ontology, each ontology concept is associated with a set of graded WordNet senses. For instance, the concept “facility” is associated with the five senses that WordNet assigns to the term “facility”. These senses range from “*something created to provide a service*” to “*a room equipped with washing and toilet facilities*”. The highest graded sense S_{facility} expresses the intended interpretation of the concept “facility” in the context of the given ontology.

It must be emphasized that although LSI exploits structural information of ontologies and WordNet, it ends up with semantic associations between terms. The algorithm is based on assumptions that influence the associations produced [7].

Using the intended meanings of the formal concepts, HCONE-merge constructs an ontology $O^n = (S^n, A^n)$, $n=1,2$, where, S^n includes the lexicalizations of the senses associated to the concepts of the ontology $O_n = (S_n, A_n)$, $n=1,2$, and A^n contain the translated inclusion and equivalence relations between the corresponding concepts. The ontology O^n is considered to be part of the hidden intermediate ontology. The construction of the intermediate ontology (by mapping the concepts of both original ontologies to WordNet senses) together with the minimal set of translated axioms results in ontologies’ merging [7].

Table 1. Comparison of the mapping methods for the HCONE-merge

	Fully-Automated	User-validated	Semi-Automated
Percentage of concepts validated by the user (in the best case)	0%	$\leq 100\%$	$> 0\%$
“Correct” mappings produced (in the best case)	80%	90%	90%

Given two ontologies O_1 and O_2 to be merged, and due to the crucial role of uncovering the intended meaning of concepts to the HCONE-merge method, we aim at automating the mapping of O_1 and O_2 to WordNet senses.

The goal is to achieve high precision in mapping concepts to their intended meaning with minimum human intervention.

Based on the algorithm for computing the s-morphism, we have shaped methods to ontology mapping, where human inspection and validation has been reduced down to the number of algorithm runs needed to correct the concept pairs whose associations produce inconsistencies with respect to the WordNet inclusion relations [8].

Table 1 compares the proposed methods [8] according to the amount of the automation they achieve and the “correct” mappings produced. The fully automated method requires the minimum number of user actions, but at the same time it achieves the lowest percentage of correct mappings. This is an iterative method that in each iteration re-computes concept mappings given the WordNet senses associated to the concepts during the last iteration. This approach is “unstable”, given that correct mappings computed during an iteration may result to non-correct mappings when re-computed in the next iteration and so on. Therefore, this method does not guarantee to converge to a set of concept mappings.

On the other hand, the user-based method achieves higher percentage of correct mappings, but the actions that are required by the user imply considerable effort, since the user has to validate the mapping of each ontology concept. It must be pointed that this case requires also a considerable number of additional algorithm runs, equal to the percentage of wrong mappings. The semi-automated method however, in addition to the high percentage of correct mappings can significantly reduce the number of concepts that need validation by the user. However, in the worst case, where each concept is involved in at least one inconsistency, validation of all concepts is required.

4 Approximating the Intended Interpretations Iteratively

The semantic morphism can be considered as a similarity function between ontology concepts and WordNet senses. For the computation of this similarity, as already explained, the s-morphism takes into account the vicinity V_C of each ontology concept C . The vicinity includes the concepts directly related to C , together with their intended meaning,

For instance, to compute the intended interpretation of the concept “*Infrastructure*” of the ontology depicted in Fig. 4, the algorithm has to take into account the intended

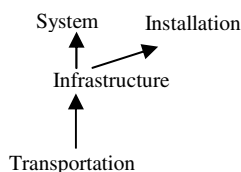


Fig. 4. Example ontology

meanings of the concepts “*Installation*”, “*System*” and “*Transportation*”. However, to compute the intended meaning of “*Installation*”, the algorithm has to take into account the intended meaning of the concept “*Infrastructure*”. As it is pointed in [10], this recursive dependency requires non-standard computation means. This problem has been approached by Bisson [11] and Euzenat [10] as an equation system where the similarity values are the solutions.

Given the ontology O_I in Fig. 4, the following system of equations has to be solved:

1. $f_s(system, \{(infrastructure, S_{infrastructure})\}) = S_{system}$
2. $f_s(installation, \{(infrastructure, S_{infrastructure})\}) = S_{installation}$
3. $f_s(infrastructure, \{(system, S_{system}), (installation, S_{installation}), (transportation, S_{transportation})\}) = S_{infrastructure}$
4. $f_s(transportation, \{(infrastructure, S_{infrastructure})\}) = S_{transportation}$

As it has been proposed in [10], given the recursive nature of these computations, we can still find the intended meaning of each concept through an iterative process that finds the most nearest reachable fixed point of a vector function. The iteration produces a sequence of vectors of tuples $((C_1, S_1) \dots (C_n, S_n))$, where each vector is an even more precise approximation of each concept’s intended meaning.

Given the above formalization, the algorithm proposed in [10] works as follows: The initial approximation is based on the lexicalization of each concept (i.e. on 0-level contributors). The approximations at step $(n+1)$ are computed using the vicinities computed in step n .

Using a variation of the above algorithm, the intended meanings of concepts are computed iteratively as shown in Table 2:

Table 2. The computation of the approximation

Repeat the following process until there is no change in the intended meaning of any concept in the ontology.
1. For each ontology concept C do the following:
1.1. For each concept in the vicinity V of C
In case there is no meaning associated to this concept compute the initial approximation based on its lexicalization
1.2. Repeat the following until there is no change in the approximation computed for the concept C
1.2.1 Compute the mapping of C using the approximations of concepts in V
1.2.2 Re-compute the approximations of concepts in V changing only the approximation of C

The computation of the approximation of each concept’s meaning (i.e. the internal loop in 1.2. that computes the approximated meaning of a concept C based on the concepts in its vicinity) converges after two or three iterations. According to our experiments, independently of the size of the ontology, the algorithm finds a fixed point for the set of concepts in the ontology in the second iteration, improving the precision of the resulted mappings.

5 Results

We have run experiments, using the proposed algorithm, with the ontologies shown in Fig. 5, Fig. 6, and Fig. 7.

For instance, running the algorithm for the version of the Transportation ontology O_1 in Fig. 5, we have observed the following results for the concept “car”: In the first approximation for this concept, given the initial approximations of all concepts in its vicinity, the algorithm computed the semantic morphism:

$$f_s(Car, \{(MotorVehicle, S_{MotorVehicle}), (Ambulance, S_{Ambulance}), (Bus, S_{bus})\})$$

The computation of this morphism returned the sense:

$S_{car} = \text{car, auto, automobile, machine, motorcar -- 4-wheeled motor vehicle,}$

In the second run for this concept, the algorithm found the same result, and therefore, this has been assumed to be concept’s “car” intended meaning, given the meanings of the concepts in its vicinity. Traversing the ontology for a second time and re-computing the semantic morphism for the concept “car”, given the new approximations of the concepts in its vicinity, the following results were returned for each iteration:

$S'_{car} = \text{cable car, car -- a conveyance for passengers or freight on a cable railway;}$

$S'_{car} = \text{car, auto, automobile, machine, motorcar -- 4-wheeled motor vehicle;}$

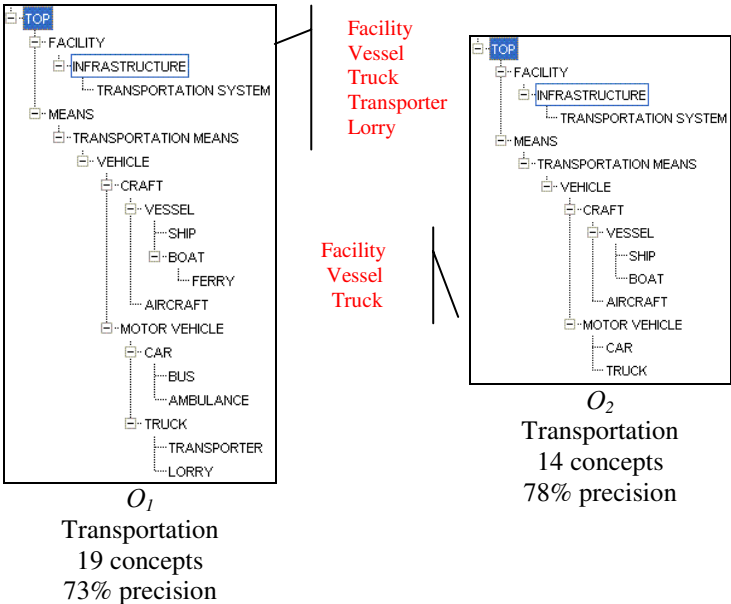


Fig. 5. Ontologies O_1 , O_2

The precision for “uncovering” concepts’ intended interpretation (in comparison to the meaning given by the engineer that devised these ontologies) is shown under each ontology snapshot. Concepts with incorrect mappings are shown within callouts drawings, attached to each snapshot.

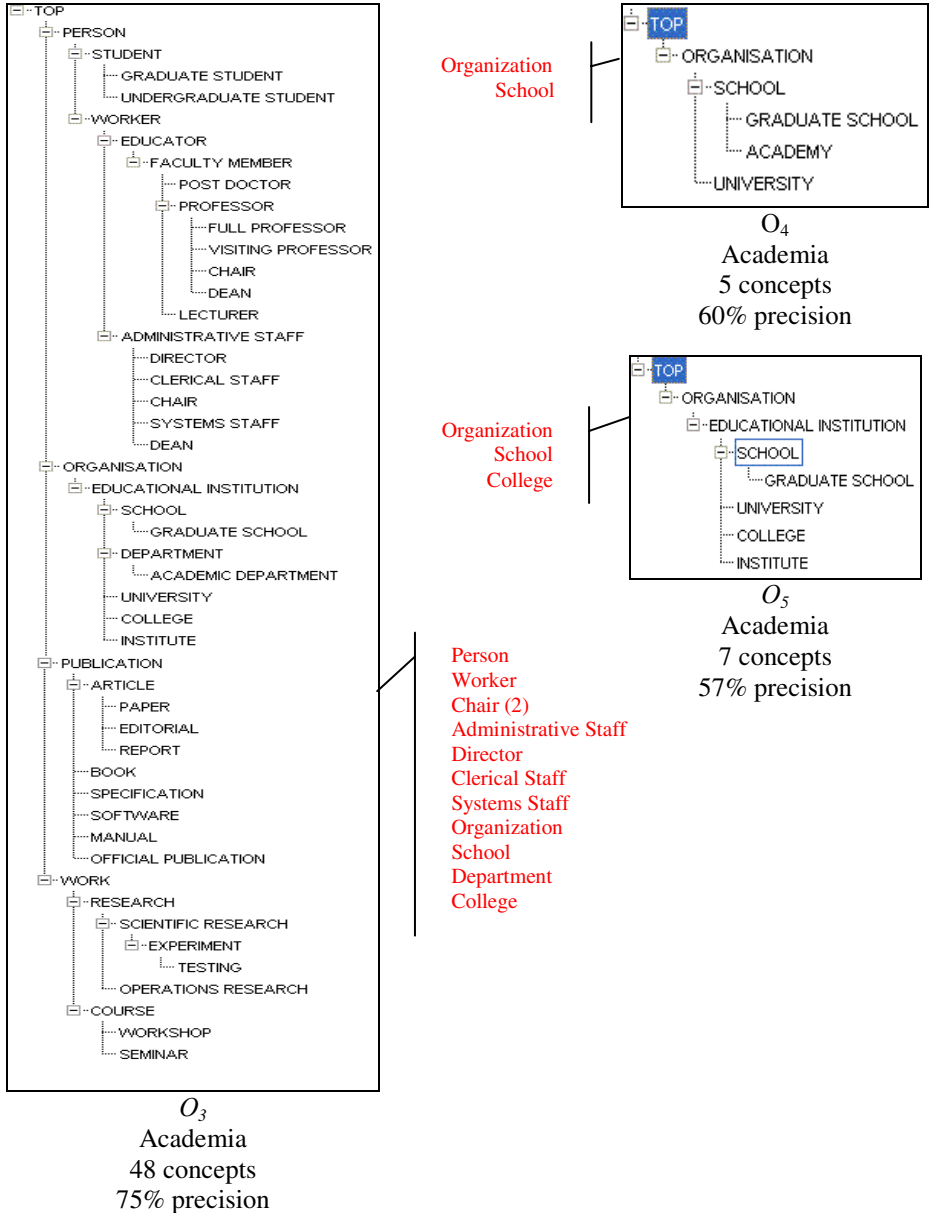


Fig. 6. Ontologies O₃, O₄, O₅

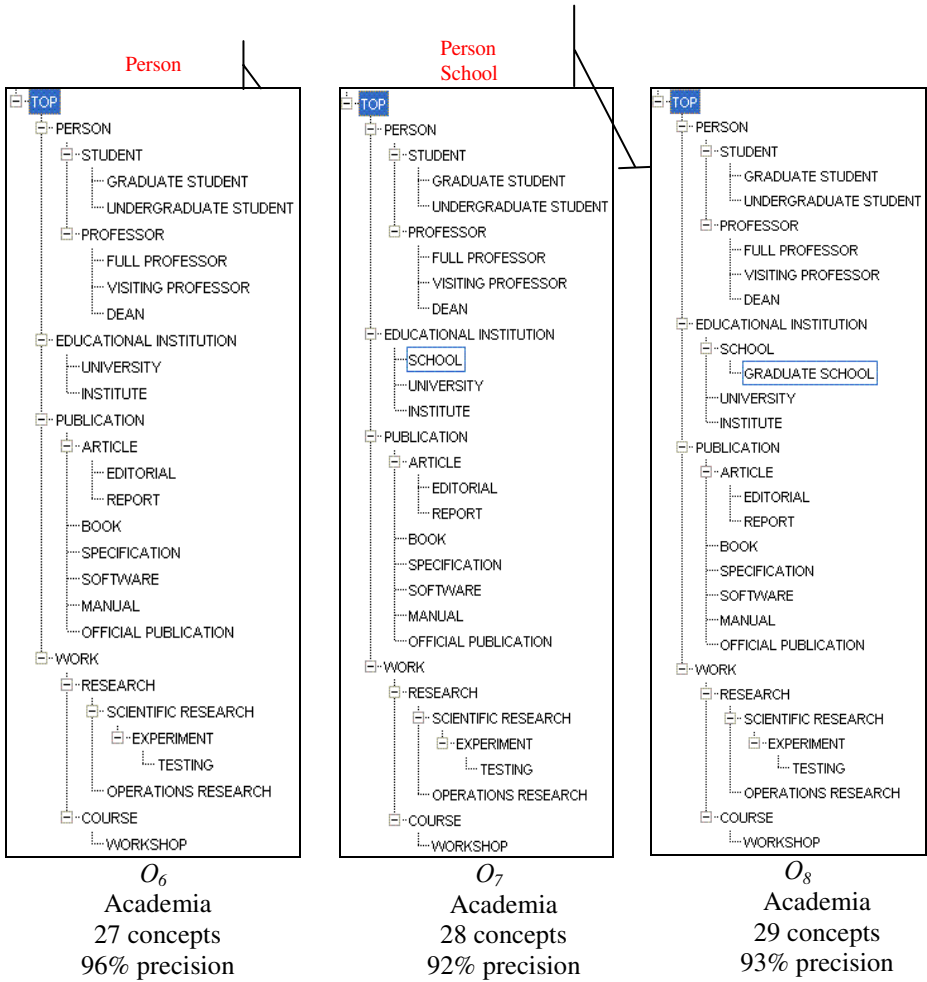


Fig. 7. Ontologies O_6 , O_7 , O_8

The computation for this concept converged in the third run. So, even if the first sense S'_{car} was different (and it is not the intended one) from the one found in the first iteration for the ontology, the algorithm converges again after two iterations to the intended concept meaning.

For the same ontology, similar results are given for the concepts “*means*”, “*infrastructure*”, and “*craft*”. The rest of the concepts do not change in the second iteration for the ontology. Therefore, the algorithm converges to a fixed point solution for the set of concepts in the second iteration for the ontology.

From the first experiments with ontologies O_1 , O_2 , and O_3 , an average precision of 74% was concluded. However, we have been experimented with variations of these ontologies in order to investigate the behaviour of the algorithm in a controlled manner.

The ontologies O_4 and O_5 include a small set of concepts for which the iterative algorithm did not converge to their intended meaning. For the ontology O_4 these are the concepts “*organization*” and “*school*” and for the ontology O_5 the same concepts in addition to the concept “*college*”. The rest 3 ontologies O_6 , O_7 and O_8 are variations of the ontology O_3 , and include a small percentage of concepts for which the algorithm did not compute their intended meaning (Fig. 7).

In more detail, the low precision achieved for the ontologies O_4 and O_5 is due to the failure of the algorithm to compute the correct mappings for the concepts “*organization*”, “*school*” and “*college*”. These concepts have not been mapped to their correct senses for O_3 as well. To explore the case that some concept mappings to WordNet cannot be “uncovered”, we have experimented with different variations of the O_3 ontology. For instance, we have run experiments with variations of O_3 that include concepts from ontologies O_3 , O_4 , and O_5 , which have been correctly mapped to WordNet, together with concepts whose computed intended meanings are not correct. Such a variation is the ontology O_6 . This includes the concept “*person*” of O_3 , whose mapping was not correct. The precision of mapping the ontology O_6 to WordNet is 96%, due to the wrong mapping of the concept “*person*”.

The addition of the concept “*school*” – chosen from the set of concepts of O_3 with wrong mappings - to ontology O_6 , results to the ontology O_7 . The precision of the algorithm for this ontology is 92% due to the wrong mappings of the concepts “*person*” and “*school*”.

Given that the sense $S_{Graduate\ school}$ computed for the concept “*graduate school*” in O_3 is the correct one, we may add this concept in ontology O_7 as a sub-concept of “*school*”. This has happened in ontology O_8 where the vicinity of the concept “*school*” has been increased with the concept “*graduate school*”. Although the algorithm computes the correct mapping for the concept “*graduate school*” in this new ontology, the algorithm computes a non-intended meaning for the concept “*school*”.

The above experiments show that the performance of the algorithm can not be improved, even if the concepts in the vicinity of the ontology concepts with wrong mappings increase. The same happens even when the “distant” concepts (not included in the vicinity) change.

6 Conclusions

Towards automating the HCONE-merge method for merging ontologies, this paper proposes a method for aligning the original ontologies with a hidden intermediate ontology in a fully automated way. Actually, the alignment is done by mapping ontology concepts to WordNet senses. These senses are supposed to express the human oriented informal intended meanings of ontology concepts.

The algorithm proposed is based on previous efforts to approximate similarities between concepts in an iterative way and, as it has been shown, produces mappings that are quite precise. Furthermore, the algorithm converges fast; in two or three iterations requiring no extensive computational time.

The results provided from our case studies show that the algorithm behaves according to our intuitions and is stable: The computations it produces do not change in case the vicinity of a concept does not change radically and do not change even if the ontology, but not the vicinity of the concept, changes. Therefore, the computation for an ontology is not drastically affected by “distant” concepts. This result agrees with the requirement on dependencies between concepts’ similarities within ontologies: The matching of a pair of concepts must depend on their local context and not to the entire ontology. However, concepts in the vicinity of a concept C must gather information from their own vicinity and further contribute such information to the computation of C ’s intended meaning.

Problems arise from the stability of the algorithm even for these concepts whose mapping is not correct. Experiments so far have not shown the exact reason for this to happen. However, future work concerns the combination of different information sources, except WordNet, for computing the intended meaning of such concepts.

References

1. Uschold M. and Gruninger M.: Creating Semantically Integrated Communities on the World Wide Web. Invited Talk, Semantic Web Workshop, WWW 2002 Conference, May, (2002)
2. Giunchiglia F. and Shvaiko P., Yatskevich M.: S-Match: An Algorithm and Implementation of Semantic Matching. *The Semantic Web: Research and Applications*, Lecture Notes in Computer Science, Vol. 3053, Springer-Verlag, (2004) 61-75
3. Kalfoglou Y. and Schorlemmer M.: Ontology mapping: the state of the art. *The Knowledge Engineering Review* 18(1):1-31 (2003)
4. Madhavan J., Bernstein P. A., and Rahm E.: Generic schema matching with Cupid. *VLDB Journal* (2001) 49-58
5. Doan A., Madhavan J., Domingos P., and Halvey A.: Learning to map between ontologies on the semantic web. In *Proc. Of WWW-02, 11th International WWW Conf.*, Hawaii (2002)
6. Noy N. and Musen M.: A. M.: PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of 7th National Conference on AI*, Austin (2000)
7. Kotis K. and Vouros G. A.: HCONE-Merge approach to ontology merging. *The Semantic Web: Research and Applications*, Lecture Notes in Computer Science, Vol. 3053, Springer-Verlag (2004) 137-151
8. Kotis K., Vouros G. A., Stergiou K.: Capturing Semantics towards Automatic Coordination of Domain Ontologies. To appear at the 11th International conference of Artificial Intelligence: Methodology, Systems, Architectures - Semantic Web Challenges - AIMS4 2004, Varna, (2004)
9. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science* (1990)
10. Jérôme Euzenat, Petko Valtchev, Similarity-based ontology alignment in OWL-Lite, Ramon López de Mantaras, Lorenza Saitta (eds), *Proc. 16th european conference on artificial intelligence (ECAI)*, Valencia (ES), pp333-337, 2004
11. Gilles Bisson. Learning in FOL with similarity measure. In *Proc. 10th American AAAI conference*, San-jose (CA US), 1992
12. Ghidini C., Giunchiglia F. A semantics for abstraction. In *Proceedings of the 16th European conference on Artificial Intelligence (ECAI-04)* Valencia, 22-27 August 2004

Soundness of Schema Matching Methods

M. Benerecetti¹, P. Bouquet², and S. Zanobini²

¹ Department of Physical Science – University of Naples – Federico II
Via Cintia, Complesso Monte S. Angelo, I-80126 Napoli (Italy)
bene@na.infn.it

² Department of Information and Communication Technology – University of Trento
Via Sommarive, 10 – 38050 Trento (Italy)
{bouquet, zanobini}@edit.unitn.it

Abstract. One of the key challenges in the development of open semantic-based systems is enabling the exchange of meaningful information across applications which may use autonomously developed schemata. One of the typical solutions for that problem is the definition of a mapping between pairs of schemas, namely a set of point-to-point relations between the elements of different schemas. A lot of (semi-)automatic methods for generating such mappings have been proposed. In this paper we provide a preliminary investigation on the notion of correctness for schema matching methods. In particular we define different notions of soundness, strictly depending on what dimension (syntactic, semantic, pragmatic) of the language the mappings are defined on. Finally, we discuss some preliminary conditions under which a two different notions of soundness (semantic and pragmatic) can be related.

1 Introduction

One of the key challenges in the development of open semantic-based systems is enabling the exchange of meaningful information across applications which may use autonomously developed schemata (database schemata, classifications, even directory trees on file systems in peer-to-peer applications) for organizing locally available data. The typical solution for that problem is the definition of a mapping between pairs of schemas, namely a set of point-to-point relations between the elements of different schemas. As in open system a beforehand agreement on the meaning of schemata seems impossible in practice, a large number of methods and systems have been proposed in order to (semi-)automatically compute on fly such mappings¹. The resulting mappings are then used as the basis for a runtime semantic-based coordination of such a network of autonomous applications.

Methods may differ along many dimensions: the type of structures to which they can be applied (e.g., trees, directed acyclic graphs, graphs); the type of result they return (e.g., similarity measures, model-theoretic relations, fuzzy relations); the resources they use to compute such a relation (e.g. external lexical resources, ontologies, string

¹ A very partial list includes [15, 14, 12, 11, 4, 6, 10, 2, 5, 9, 3]. A detailed description of these methods is out of the scope of this paper.

manipulators, graph matching techniques, instance-based techniques). In this paper, for reasons that will be explained in detail, we are mostly concerned with a class of methods that we call *semantic methods*. The general intuition underlying semantic methods is that they aim at discovering relations between (pairs of) entities belonging to different schemata *based on the meaning of the two entities*. However, beyond this point, there is a significant disagreement on what characterizes a semantic method from a non semantic method. For example, recent papers by Giunchiglia and Schvaiko [7, 8] propose to include among semantic methods only those methods that directly return a semantic relation (e.g., material implication or logical equivalence), namely a relation with a well-defined model-theoretic interpretation. This analysis is far from being shared in the community, as other people feel that a method is semantic if it uses semantic information to return its results, or if there is a principled way to assign an indirect semantics to its results (e.g., mapping numerical values on semantic relations through the definition of suitable thresholds).

In such a situation, it is not surprising that we still lack a clear definition of the conditions under which a semantic method can be said to work “correctly”. Suppose, for example, that we have a method α that takes in input two nodes n_A and n_B from two schemata S_A and S_B respectively and returns `True` if the two nodes represent equivalent concepts, `False` otherwise. Now, imagine that α is fed with the categories `/IMAGES/TUSCANY/FLORENCE` and `/PHOTOS/ITALY/FLORENCE`² belonging to two classification schemata, and that it returns `True`. Is the result “correct”? Why? And what if the result were `False`? Under what conditions would we accept this result as “correct”?

This paper aims at answering this kind of questions. First, we propose a simple theoretical model which allows us to classify methods for schema matching in three broad categories (syntactic methods, semantic methods, and pragmatic methods). Then we turn our attention to semantic methods, and propose a characterization of these methods and a notion of (semantic) soundness. Semantic methods have the advantage that are computationally quite good, but we’ll argue that in general they do not guarantee to capture the intuitive notion of a “good” schema matching method. We then introduce a notion of pragmatic methods (and the corresponding notion of soundness), and argue that they more closely corresponds to what is intuitively expected by a schema matching method. However, we also show that in general they can’t be effectively computed. Therefore, in the last part of the paper we try to identify some very general conditions under which a semantically sound method can guarantee pragmatic soundness as well, which is – in our opinion – the best we can get from a semantic method for schema matching.

2 The Problem of Schema Matching

Schema is a broad term, that applies to different kinds of structures. In [3], it was argued that it makes no much sense to speak about schema matching in general, and

² Throughout the paper we will use the notation $X/\dots/Y$ to refer to a path in schema in analogy with the notation for paths in a file system. If the schema is a tree, then $/$ represent the root node and $X/\dots/Y$ the unique path from X to Y .

that the analysis should be done case by case along the dimension of the intended use of a schema. Accordingly, in this paper we restrict our attention to a special kind of schemata, *hierarchical classifications*, whose explicit purpose is to classify objects (e.g., documents). This restriction does not affect the generality of our investigation, as the method of analysis can be applied to study the problem of matching other types of schema, such as database schemata, service descriptions, datatypes.

We start with a few definitions that characterize the kind of schemata we deal with, namely topic hierarchies used as classification schemata.

Definition 1 (Topic hierarchy). Let Λ be a set of labels (e.g., words in natural language). A topic hierarchy $\mathcal{S} = \langle K, E, l \rangle$ is a triple where K is a finite set of nodes, E is a set of arcs on K , such that $\langle K, E \rangle$ is a rooted tree, and l is a function from K to Λ .

Two simple examples of topic hierarchies are depicted in Figure 1.

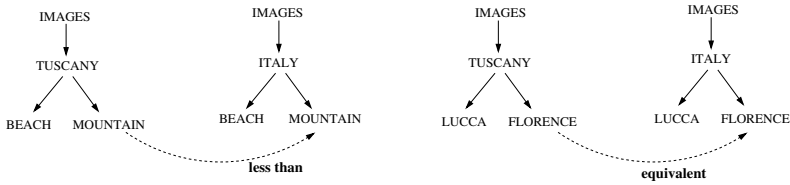


Fig. 1. Two simple topic hierarchies

A possible use for topic hierarchies is to classify documents. To express this formally, we introduce the notion of classification function.

Definition 2 (Classification function). Let D be a set of documents and $\mathcal{S}$ a topic hierarchy $\langle K, E, l \rangle$. A classification function over $\mathcal{S}$ is a function $\tau : D \rightarrow K$ from documents to nodes of $\mathcal{S}$.

A classification function places a document under a node in a topic hierarchy. We associate to each classification function a *retrieval function*, which is a function from nodes to the sets of documents attached to them in a topic hierarchy. It essentially plays the inverse rôle of the classification function.

Definition 3 (Retrieval function). Let D be a set of documents, $\mathcal{S} = \langle K, E, l \rangle$ a topic hierarchy, and τ a classification function over $\mathcal{S}$. The retrieval function of τ over $\mathcal{S}$ is a function $\mu_\tau : K \rightarrow 2^D$ satisfying the following condition:

$$\text{for every } d \in D, d \in \mu_\tau(\tau(d))$$

Finally, we can define a hierarchical classification (hereafter HC) simply as a topic hierarchy with an associated classification function τ . Formally:

Definition 4 (Hierarchical classification). Given a set of documents D , a hierarchical classification $\mathcal{H} = \langle \mathcal{S}, \tau \rangle$ is a pair where $\mathcal{S}$ is a topic hierarchy and τ is a classification function over $\mathcal{S}$.

Schema matching can be defined as the problem of computing relations between pairs of nodes belonging to different HCs. Let $\mathfrak{R}$ be a set of relations that may hold between two nodes belonging to two distinct schemata $\mathcal{S}_A$ and $\mathcal{S}_B$. Then a mapping is defined as follows:

Definition 5 (Mapping). A mapping $\mathcal{M}_{A \rightarrow B}$ between two HCs $\mathcal{H}_A = \langle \mathcal{S}_A, \tau_A \rangle$ and $\mathcal{H}_B = \langle \mathcal{S}_B, \tau_B \rangle$ is a set of triples $\langle n_A, n_B, r \rangle$, where:

- n_A and n_B are two nodes belonging to $\mathcal{S}_A$ and $\mathcal{S}_B$, respectively;
- $r \in \mathfrak{R}$ is a relation between n_A and n_B .

Each triple $\langle n_A, n_B, r \rangle$ belonging to a mapping is called a *mapping element*.

Finally, as our goal is to discuss properties of schema matching methods, we formally define a method as a function which returns true when a given relation holds between two elements of different schemata, false otherwise:

Definition 6 (Schema Matching Method). Let $\mathcal{M}_{A \rightarrow B}$ be a mapping between two HCs $\mathcal{H}_A$ and $\mathcal{H}_B$. A schema matching method $\alpha : \mathcal{M}_{A \rightarrow B} \rightarrow \{T, F\}$ is a function from mapping elements to boolean values.

Of course, it is more natural to view a method as a function which takes two nodes as input and returns a relation as output. Here we adopt this more abstract (but after all equivalent) characterization as it is more appropriate for our analysis.

3 A Three-Layer Model of Schema Matching

Before we proceed with our discussion of soundness for schema matching methods, we discuss a simple model which can help us in clarifying what the task of schema matching is from a theoretical point of view.

In a schema matching task, there are three levels that can be taken into account (see Figure 2):

Language level: the language level is the level of expressions (an alphabet and a grammar to build more complex expressions) that can be used to label a schema. Such a language is used to “publish” information about a schema, and possibly to exchange information about the schema with other applications. Therefore, by definition, this level must be publicly accessible. If we do not make any assumption on such a language, then labels should be regarded as mere syntax, with no special meaning. Therefore, if we restrict our analysis to this level, the only kind of mapping that can be found is purely syntactic, and the only information that can be used to compute such a mapping has to do with the syntactic properties of the strings that are used to label nodes, and their arrangement in the schema. However, as we will argue, there are good reasons to assume that labels are meaningful expressions (typically natural language terms), and this has important consequences on how they are treated in schema matching methods.

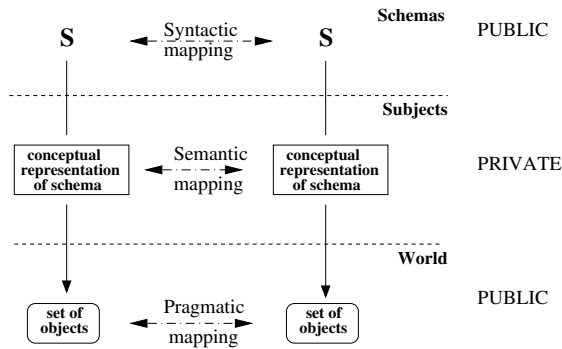


Fig. 2. Three level of schema matching

Concept level: at the concept level, we find a collection of concepts that correspond to the intended meaning of nodes in a schema. Intuitively, this level corresponds to what the creator (or the users) of a schema “had in mind” when the schema was created (or when the schema is used). The main difference between the language level and the concept level is that concepts are not directly accessible, and therefore cannot be used to publish a schema or to convey information about a schema. As a consequence, a schema matching method can compute a mapping between concepts only indirectly, e.g. by making conjectures about the most plausible interpretation of labels. However, this is clearly the level at which most schema matching methods aim, as we are typically interested in the relation between “meanings” and not between syntactic structures.

Object level: at the object level, we find the objects themselves, namely the objects that the schema is supposed to organize. For HCs, the relevant objects are documents, namely the entities that are associated to nodes in a classification schema, and a mapping may be a set-theoretic relation between pairs of sets of documents. Objects are by definition publicly available. However, as we will argue, the fact that the set of documents associated to a node in a schema is, for example, a subset of the set of documents associated to a node in another schema does not tell us much about the concepts associated to the two sets. In particular, it can’t even tell us that, whatever the relevant concepts are, one subsumes the other, as it may well be that the two sets are not sufficiently representative.

Given these three levels, we can imagine three broad classes of methods: (i) *syntactic methods*, namely methods that use only information at the language level to compute mappings across schemata; (ii) *semantic methods*, namely methods that use only information at the conceptual level; and *pragmatic methods*, namely methods that use only information at the object level. In practice, very few methods can be said to belong to a single category, and for good reasons. For example, as we said, most syntactic methods are (often implicitly) based on the assumptions that labels are meaningful (or even natural language expressions), and this justifies for example the use of thesauri that would not be allowed otherwise (if two nodes PICTURES and PHOTOS were mere abstract labels, what would be the justification for exploiting the idea of synonymy to

match them?). However, and even more important, semantic methods – as we will argue in detail – must start from the language level, as concepts cannot be represented directly; and thus moving from the language level to the concept level is a crucial step for any real world semantic method.

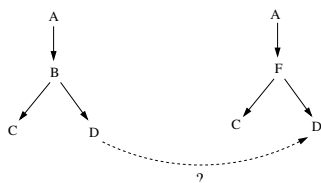
In the rest of the paper, we will disregard purely syntactic methods, as they are of little use in most applications of schema matching³. We will focus on semantic and pragmatic methods. For each of them, we will provide a precise characterization and a notion of soundness.

4 Soundness of Semantic Methods

Semantic methods are methods which return mappings across concepts associated to nodes in two (or more) schemata. As we argued in the previous section, semantic methods are intrinsically based on two macro steps:

Semantic elicitation: this step takes as input a linguistic description of a node in a schema and returns a representation of its meaning in terms of conceptual representation. As the idea is to design computer-based matching methods, such a meaning must be expressed by some formal object of a logical type, corresponding to the logical type of the node’s intended meaning. Notationally, if n is a node in a HC, then $\mathcal{T}(n)$ denotes the formal representation of its meaning;

³ A simple example will illustrate this claim. Consider the two simple abstract schemata below:



and compare the problem of discovering mappings between nodes of the two abstract schemata with the problem of discovering mappings across schemata with meaningful labels like those in 1. Nodes in abstract schemata do not have an implicit meaning, and therefore, whatever technique we use to map them, we will find that there is some relation between the two nodes labeled D in the two schemata, which depends only on the abstract shape of the two schemata. The situation is completely different for schemata with meaningful labels, as we can make explicit a lot of information that we have about the terms which appear in the graph, and their relations (e.g., that Tuscany is part of Italy, that Florence is in Tuscany, and so on). It is this kind information which allows us to understand why the semantic relation between the two nodes labeled MOUNTAIN and the two nodes labeled FLORENCE is different, despite the fact that the two pairs of schemata are structurally equivalent, and both are structurally isomorphic with the pair of abstract schemata. Indeed, for the first pair of nodes, the set of documents we would classify under the node MOUNTAIN on the left hand side is a subset of the documents we would classify under the node MOUNTAIN on the right; whereas the set of documents which we would classify under the node FLORENCE in the left schema is exactly the same as the set of documents we would classify under the node FLORENCE on the right hand side.

Semantic comparison: given two nodes n and m belonging to different HCs, a semantic method must return a relation which connects the concepts expressing the meanings of the schema elements under comparison. Such a relation must in turn have an interpretation defined with respect to the meaning of the compared elements.

So, according to the first step, a semantic method should explicitly interpret the elements of a HC as concepts, and provide a corresponding formal representation of type concept (e.g., using some Description Logic language [1]). For example, the meaning of the nodes FLORENCE of right and left hand side of schemas of Figure 1 approximately corresponds to the two concepts “Images of Florence in Tuscany” and “Images of Florence in Italy”. Notice that a schema describing how a web service works (basically, a finite state automaton) should be interpreted in a completely different way, as nodes would represent states that can be reached through actions associated to arcs.

In the second step, a semantic method is supposed to return a relation between concepts (e.g., subsumption, equivalence, and so on). Notice that here we will privilege classical model-theoretic relations, though it is possible to work with fuzzy-theoretic relations between concepts. Going back to the example of Figure 1, the relation between the two nodes FLORENCE (interpreted as concepts) is that they are equivalent. We note that, in this case, determining the relation between the two concepts intuitively requires to use further knowledge w.r.t. the one extracted from the two schemata – namely that Tuscany is in Italy. In the following, we will refer to this (possibly external) further knowledge as the *ontology* associated to a method. In analogy to what we said above, a relation between elements of two service description schemata would be completely different.

We are now ready to provide a formal notion of soundness for semantic methods. Given the two semantic steps discussed above, a semantic method α is defined by: (i) a language $\mathcal{L}$ suitable to explicitly represent the meaning of each schema element, (ii) a procedure for extracting the meaning of each element n ($\mathcal{T}(n)$), (iii) a (possibly empty) ontology $\mathcal{O}$ expressing knowledge about the domain, and (iv) a set of relations $\mathfrak{R}$ to be computed between pairs of nodes. The 4-tuple $\langle \mathcal{L}, \mathcal{O}, \mathcal{T}(), \mathfrak{R} \rangle$ is what we call the *semantic frame* of the method.

We now propose a notion of semantic soundness and completeness of a semantic schema matching method with respect to a semantic frame F . The intuition is the following: a method is *semantically sound* w.r.t. F if, whenever it computes a relation between two elements of distinct schemata, the relation follows from what the method knows about the meaning associated to the two elements; and is *semantically complete* if, whenever one of the relations in $\mathfrak{R}$ between the meaning of two nodes follows from what the method knows, then the method effectively returns that relation. More formally:

Definition 7 (Semantic Soundness). Let $F = \langle \mathcal{L}, \mathcal{O}, \mathcal{T}(), \mathfrak{R} \rangle$ be the semantic frame of a method α and $\mathcal{H}_A$ and $\mathcal{H}_B$ be two HCs. Then α is *semantically sound* w.r.t. F if and only if for any mapping element $\langle n_A, n_B, r \rangle$ the following holds:

$$\text{if } \alpha(\langle n_A, n_B, r \rangle) = T, \text{ then } \mathcal{O} \models_{\mathcal{L}} \mathcal{T}(n_A) \text{ } r \text{ } \mathcal{T}(n_B)$$

Definition 8 (Semantic Completeness). *Let $F = \langle \mathcal{L}, \mathcal{O}, T(), \mathcal{R} \rangle$ be the semantic frame of a method α and $\mathcal{H}_A$ and $\mathcal{H}_B$ be two HCs. Then α is semantically complete w.r.t. F if and only if for any two nodes n_A and n_B , the following holds:*

$$\text{if } \mathcal{O} \models_{\mathcal{L}} T(n_A) \text{ } r \text{ } T(n_B), \text{ then } \alpha(\langle n_A, n_B, r \rangle) = T$$

Though these notions of semantic soundness and completeness seem reasonable, it should be quite evident that they do not seem to capture what we have in mind when we say that a method is correct. Indeed, what we would like to say is that a method is sound when it computes the “right” relation between two elements, namely the relation that follows from the “correct” interpretation of the schemata and from the use of the “right” background knowledge. Instead, what the definitions above says is only that, given an ontology and a formal representation of the meaning of two nodes, then a semantic method is sound if and only if it derives only relations that logically follows from the background knowledge provided by its ontology. But this is tantamount as saying that a semantic method is sound if and only if the reasoner used to compute the relation between meanings is sound and complete, which would be a very trivial result. Indeed, imagine a dummy method that associate the same concept k to all the elements of two HCs, and always returns the equivalence relation for any pair of nodes (for all $k \in \mathcal{S}$ and $k' \in \mathcal{S}'$, $\alpha(k, k', \equiv) = T$). Since any concept is always equivalent to itself, then this method is semantically sound. But is this method of any interest?

Intuitively, the problem is that semantic soundness as we defined it (and a similar argument can be done for completeness) does not say anything about the appropriateness of the meaning elicitation performed by the method and on the relation between the meaning of nodes and the available ontology. In short, semantic soundness is a necessary but not sufficient condition to capture the intuitions we have about the correctness of a method. What we need as a sufficient condition is a way for excluding dummy methods like the one described above, namely methods that build arbitrary interpretations and use non pertinent knowledge about the meaning of schema elements.

However, this is an extremely tough problem not only in schema matching, but in general for any semantic theory based on formal logic. Indeed, as we know from classical results (see e.g. the model-theoretic argument discussed by the philosopher H. Putnam in [13]), there’s nothing we can do to prevent unintended interpretations of a formal language. The form in which Putnam discusses this problem is the following: even if two agents agree on the truth value of all the sentences of a language L (including modal propositions on the necessity of propositions), this is not sufficient to fix the interpretations of the terms they use, which means that they may still be talking about different things. From Putnam’s argument, we can derive an even stronger condition: even if subjects shared the function connecting the conceptual level to the linguistic level (and therefore they agreed on the conceptual representation of any statement at the language level), nothing assures us that the function connecting the semantic level to the pragmatic level is also shared. This means that two agents may agree at the conceptual level, but not at the pragmatic level.

Applying this considerations to schema matching methods means that even if we can guarantee that a method is semantically sound and complete, there is nothing that guarantees that (i) the two elements were correctly interpreted, and (ii) even if they were correctly interpreted, that the relation between the two nodes is the one we expect.

To sum up, assuming the existence of sound reasoners (SAT, DL reasoners, and so on), it seems relatively to come up with a semantically sound method. But this notion of soundness seems to be of little use if we cannot guarantee some form of pragmatic soundness. Let us turn now to this notion.

5 Soundness of Pragmatic Methods

A pragmatic method is a method which returns mappings across schema elements which depend on some relation between the sets of objects associated to the schema elements themselves. If we restrict our analysis to HCs, a pragmatic method is a method which computes relations between HC nodes through the analysis of set-theoretic relation between the sets of documents actually classified under the two nodes. The intuition underlying pragmatic methods is the following. Suppose we have a collection D of documents, and that a user is required to classify them into two different HCs. Then, if two nodes in two HCs have the “same” meaning, then we may expect that the user will classify the same set of documents under the two nodes; and if the meaning of a node is subsumed by the meaning of another node, then the user will classify under the first node a subset of the documents classified under the second node; and so on. Following this intuition, a pragmatic method may work backward and try to infer the relation between the meaning of two nodes from the relation between the collections of documents associated to the nodes themselves.

In the following, we shall try to make this intuition more precise. Let us first introduce a notation to refer to the set of documents classified under (all the nodes in) a subtree of a topic hierarchy, instead of a single node. The reason is the following. Consider the the right hand side pair of HCs in Figure 1. If we want to compare the nodes labeled TUSCANY and ITALY in the two HCs, it will not be in general sufficient to consider only the documents specifically classified under those two nodes. We should take into account the whole set of documents classified under all nodes belonging to the subtrees rooted in those two nodes. Indeed, any document classified under node FLORENCE is also implicitly classified under node TUSCANY (resp., node ITALY). What one expects in this case is that the set of documents classified in the subtree rooted in TUSCANY be a subset of the set of documents classified in the subtree rooted in ITALY. To capture this intuition, we introduce the notation $\mu_\tau(n \downarrow)$ to denote the set of documents classified under a subtree rooted at the node n . More formally, let $n \downarrow = \{k \in K \mid k \text{ is a descendant of } n\}$ denote the set of nodes in the subtree rooted at n , then $\mu_\tau(n \downarrow) = \bigcup_{m \in n \downarrow} \mu_\tau(m)$.

Let D be a set of documents and $\mathfrak{R}$ a set of relations between sets of documents (for example, $\mathfrak{R} = \{=, \subseteq, \supseteq, \perp\}$, where $\perp$ means disjoint). Furthermore, imagine that a classifier classifies all documents of D in two different HCs ($\mathcal{H}_A$ and $\mathcal{H}_B$). Then a first tentative definition of pragmatic soundness could be the following:

Definition 9 (Strong Pragmatic Soundness). *Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be two HCs and α a semantic method. Then α is strongly pragmatically sound if for any mapping element $\langle n_A, n_B, r \rangle$ (with $r \in \mathfrak{R}$) the following holds:*

$$\text{if } \alpha(\langle n_A, n_B, r \rangle) = T \text{ then } \mu_\tau(n_A \downarrow) \mathfrak{r} \mu_\tau(n_B \downarrow)$$

Intuitively, this means that if a semantic method α discovers a relation r between two nodes n_A and n_B , then the corresponding set-theoretic relation r also holds between the sets of documents classified by the function τ in the subtree rooted at the nodes n_A and n_B ⁴. Notice, however, that this definition presupposes two very strong assumptions:

1. *the set D must be the set of all possible documents*. Indeed, it may well happen that the set of documents actually classified is not sufficient to discriminate between some set-theoretical relations, such as $\subset$ and $=$. In other words, it may be the case that the set of documents considered is not enough to tell two nodes apart, while they will be if we had had more documents available;
2. *each document can be classified in a unique way*. This is not the case in general, as documents are typically *rich objects*, and can be classified under different categories, depending on what aspects of the document are taken as the relevant ones for a given classification task. For example, this paper could be classified under different categories (e.g. SEMANTIC INTEROPERABILITY, ONTOLOGY INTEGRATION, SCHEMA MATCHING, FORMAL MODELS), and each of these categories would reflect a legitimate point of view on the paper. Therefore, even if two categories in two different HCs – populated by the same classifier – are semantically related, we can’t guarantee that the sets of documents classified under those two categories will be in the same relation.

To overcome the second assumption, we provide a weaker notion of pragmatic soundness, which can take into account the possibility that a classifier (human or automatic) can legitimately classify the same document under different categories. To capture this intuition, we first introduce the following finer notion of classifier:

Definition 10 (Classifier). A classifier C is a set of classification functions $\{\tau_i\}$.

Associating a set of classification functions to a classifier allows us to capture the fact that it can classify the same set of document in different ways. Therefore, when populating a topic hierarchy, we allow classifiers to employ any of their classification functions. Intuitively, the set $\{\tau_i\}$ can be seen as a set of “acceptable” classification functions, in the sense that the classifier will be prepared to accept classifying a document under a given node if there is a classification function belonging to $\{\tau_i\}$ which would classify that document under the same node.

Based on the definitions above, we can now attempt a second definition of *pragmatic soundness* which, we believe, is the best we can expect from a schema matching method. Intuitively, we say that a schema matching method is *pragmatically sound* if whenever it derives a relation r between two nodes n_A and n_B , a classifier would consider this result as “acceptable” according to the possible ways he could classify a set of documents. By “acceptable” here we mean that whatever set of documents C has actually placed under n_A and n_B (using one of his classification functions), C could

⁴ With an abuse of notation, we use the symbol r to refer both to the (semantic) relation computed by a semantic method and the relation which holds between sets of documents. We rely on the intuitive mapping between semantic relations (say, subsumption between concepts) and set-theoretic relations between their interpretation (for subsumption, it would be set inclusion).

have placed under n_A , using a possibly different admissible classification function, a set of documents in the same relation r with the set of documents actually placed under n_B . This intuition is captured by the following definition:

Definition 11 (Pragmatic Soundness). *Let C be a classifier, and $\mathcal{H}_A$ and $\mathcal{H}_B$ be two HCs. A method α is pragmatically sound w.r.t. C if, for any mapping element $\langle n_A, n_B, r \rangle$, the following holds: if $\alpha(\langle n_A, n_B, r \rangle) = T$, then for any classification τ_2 of C there is a classification τ_1 of C such that, for any $r \in \mathfrak{R}$, $\mu_{\tau_1}(n_A \downarrow) r \mu_{\tau_2}(n_B \downarrow)$.*

Notice that while this definition allows us to relax the second assumption, the first assumption is still needed to allow for a sensible notion of soundness, and seems to be much harder relax.

Summarizing, differently to semantic methods, pragmatic methods seem to provide meaningful answers, at least in principle. On the other hand, due to the need to satisfy assumption 1, sound pragmatic methods do not seem to be possible in practice.

6 Can Semantic Methods be Pragmatically Sound?

The point we reached can be described as follows. On the one hand, it seems relatively easy to design and implement semantically sound methods, but the answer they provide can be of little use, as we can't guarantee its pragmatic adequacy. On the other hand, pragmatic methods can provide a provably adequate answer, but they can't be computed in real cases, due to the strong requirements they presuppose. The solution seems to be either trivial or impossible.

A possible way out of this situation would be to take the advantages of the two methods, while avoiding their drawbacks. This essentially amounts at (i) defining a schema matching method which is semantically sound (as it is 'easy' to design), and (ii) to create the conditions under what such method is, at the same time, pragmatically sound (as it provides 'meaningful' results). In this section we provide some *preliminary conditions* under which such a result would be possible.

In definition 10 we define a classifier as a set of classification functions. In real cases, we expect that there is a rationale behind the classification tasks of any 'reasonable' classifier. In other words, we expect that classifiers perform their task based on their knowledge about the documents to be classified and about the available categories. As an example, we expect that a classifier τ classifies a document d (say, a photo of Florence) under a node PHOTO/FLORENCE, or under a node PHOTO/TUSCANY (if the classifier knows that Florence is in Tuscany), and not, say, under a node BOOK/ANIMAL. In other words, a classifier classifies a documents with respect to the meaning it associates to the documents. Furthermore, we expect that in presence of both the nodes PHOTO/FLORENCE and PHOTO/TUSCANY, the classifier classifies the document d under the node PHOTO/FLORENCE and not under the node PHOTO/TUSCANY. Essentially, we expect the classifier classifies the documents in the more specific node. When a classifier respect such constraints, is said to be *pragmatically competent*. Such competence intuitively represents a sort of bridge between the semantic and pragmatic spheres: it says that a document is classified w.r.t. the intended meaning the *classifier associates* to documents itself.

Formally, let $D = \{d_1, d_2, \dots\}$ be the set of all the documents. Furthermore, let $M = \{\phi_1, \phi_2, \dots\}$ be the set of all the intended meanings that can be associated to documents in D . As an example, we could say that the document $d_k \in D$ is the present paper, and that d_k can be associated with the intended meaning ‘semantic web’ (e.g., the topic of the paper), or with the intended meaning ‘paper of 2004’ (e.g., the year when the paper has been written), or the intended meaning ‘submitted paper’ and so on. We assume all these meanings are contained in M . Notationally, we write $\phi^{\tau_j, d_k} \in M$ for indicating the intended meaning in M used by the classification function τ_j for classifying the document d_k .

Definition 12 (Pragmatic competence). *Let $C = \{\tau_i\}$ be a classifier, and $F^C = \langle \mathcal{L}^C, \mathcal{O}^C, \mathcal{T}^C(), \mathbb{R}^C \rangle$ be a semantic frame. C is pragmatically competent w.r.t F^C if, for any structure $\mathcal{H}$, for any document $d_k \in D$, for any node $n \in \mathcal{H}$ and for any classification function $\tau_j \in C$, the following holds:*

if $d \in \mu_j(n)$ then:

- a) $\mathcal{O}^C \models_{\mathcal{L}^C} \mathcal{T}^C(n) \sqsupseteq \phi^{\tau_j, d_k}$
- b) for no other node $m \in \mathcal{H}$, $\mathcal{O}^C \models_{\mathcal{L}^C} \mathcal{T}^C(m) \sqsupseteq \phi^{\tau_j, d_k}$ and $\mathcal{T}^C(n) \sqsupseteq \mathcal{T}^C(m)$

The definition above simply requires that a competent classifier classifies documents w.r.t. the intended meaning (requirement a) and under the most specific nodes (requirement b). In particular, a document is classified under a node n if the meaning associated to the document entails the meaning of that node, and if no more specific node m exists into the structure. As an example, imagine our document d_k is a photo of some church in a city of Tuscany (Italy), say Lucca. Imagine that the classifier τ_j associates the intended meaning ‘photo of Lucca’ to the document d_k , say ϕ^{τ_j, d_k} . Furthermore, imagine that the meaning of the node TUSCANY of right hand schema of Figure 1 ($\mathcal{T}^C(\text{TUSCANY})$) is ‘images of Tuscany’. Then, the classification function τ_j is competent if it classifies the document d_k in the node TUSCANY, as a ‘photo of Lucca’ is also an ‘image of Tuscany’ (requirement a) and it doesn’t exist any other more specific node where to classify the document⁵ (requirement b).

A important consequence of Definition 12 is the following:

Proposition 1. *Let $C = \{\tau_i\}$ be a pragmatic competent classifier w.r.t. a semantic frame $F^C = \langle \mathcal{L}^C, \mathcal{O}^C, \mathcal{T}^C(), \mathbb{R}^C \rangle$. Then, given any two nodes n_A in $\mathcal{H}_A$ and n_B in $\mathcal{H}_B$, for any $r \in \mathbb{R}^C$ the following holds: if $\mathcal{O}^C \models \mathcal{T}^C(n_A) r \mathcal{T}^C(n_B)$ then for any classification function $\tau_1 \in C$, there is another $\tau_2 \in C$ such that $\mu_{\tau_1}(n_A \downarrow) r \mu_{\tau_2}(n_B \downarrow)$.*

Proposition 1 simply states that if a classifier C associate a set of documents to some node n_A , and n_A is in a certain relation r with a second node n_B w.r.t. the semantic frame F^C , then C must be prepared, possibly by employing some other acceptable classification function of his (namely, a compatible classification function), to classify under the n_A a set of documents holding the same relation r with the set of documents attached to n_B . Notice that Proposition 1 is an immediate consequence of Definition 12.

⁵ The other node where is possible to classify the node, according to the requirement a, is the node IMAGES. But this is less specific than the node TUSCANY.

Assume now we have a semantically sound matching method α which can answer whether a semantic relation between two nodes holds or not. In this section we try to answer the question of what condition can guarantee that a sound semantic method α is also pragmatically sound⁶. We can state the following proposition:

Proposition 2. *Let $F = \langle \mathcal{L}, \mathcal{O}, \mathcal{T}(), \mathbb{R} \rangle$ be a semantic frame, α a method semantically sound w.r.t. F , and $C = \{\tau_i\}$ a pragmatically competent classifier with respect to a semantic frame $F^C = \langle \mathcal{L}^C, \mathcal{O}^C, \mathcal{T}^C(), \mathbb{R} \rangle$. If $\mathcal{O} \sqsubseteq \mathcal{O}^C$ and $\mathcal{T}^C(.) = \mathcal{T}(.)$, then α is pragmatically sound. Moreover, if $|C| = 1$, then α is strongly pragmatically sound.*

The proposition states that if (i) α is semantically sound, (ii) the ontology used by α is subsumed by a pragmatically competent classifier knowledge (i.e., it is a sound but not necessarily complete representation of the classifier knowledge), and (iii) the meaning assigned to the nodes by $\mathcal{T}^C(.)$ and $\mathcal{T}(.)$ is the same (namely, for any node m of any schema $\mathcal{S}$, $\models \mathcal{T}^C(m) \equiv \mathcal{T}(m)$), then α is also pragmatically sound. If, in addition, (iv) the classifier always uses the same classification function, then clearly α is also strongly pragmatically sound.

The first part of the proposition immediately follows from Proposition 1 and Definitions 7 and 11. The second part descends from Proposition 1 and Definitions 7, and 9. A sketch of proof follows. Since any relation between concepts that can be deduced from a less specific ontology ($\mathcal{O}$) can also be deduced by a more specific one ($\mathcal{O}^C$), Condition (i) together with Condition (ii) ensure that any relation discovered by the method α would also be inferred by any classifier. Moreover, if C is a pragmatically competent classifier, whatever classification function τ he has used to place documents under node n_A and n_B , by Proposition 1 there must be another acceptable classification function τ' of C using which C would have placed under n_A a set of documents holding the relation r with those placed by τ under n_B . Hence pragmatic soundness follows. Adding the additional constraint that the classifier only allows for a single classification function, immediately leads to strong pragmatic soundness.

Let us now briefly comment on the conditions we needed to guarantee pragmatic soundness of a semantic matching method. Condition (i) is quite easy to ensure, as we already pointed out in Section 4. A logic framework powerful enough to express the desired semantic relations between the concepts of interest, for which decidability is guaranteed will suffice. Condition (ii) seems to be a relatively weak requirement. This is an important observation, since providing a method with complete knowledge with respect to a classifier is likely to be a very hard task, let alone the problem of providing complete knowledge with respect to *any* classifier. Even though the first two conditions seem to be reasonably easy to satisfy, Condition (iii) turns out to be quite strong, as it states that we must determine the ‘right’ meaning (with the respect to the one assigned by the classifier) of each schema element. Notice that weakening the condition on the semantic elicitation functions is problematic. Indeed, a condition as $\models \mathcal{T}(m) \sqsubseteq \mathcal{T}^C(m)$ (which states that the meaning associated to each schema element by the matching method is consistent with the meaning associated to the same element

⁶ The problem of pragmatic completeness is significantly harder and out of the scope of this paper. We will not discuss it here.

by the classifier) preserves the soundness only with respect to the disjointness ($\perp$). Unfortunately, it doesn't hold with respect to none of the other relations we have been considering in the paper ($\sqsubseteq$, $\sqsupseteq$, $\equiv$).

7 Conclusions

The consequence of Proposition 2 is that semantic methods can be guaranteed to obtain pragmatically correct results under conditions (i)–(iii) (also (iv) if we want strong pragmatic soundness). As condition (i) is quite trivial, we can conclude that the roadmap to correct semantic methods is quite clear: (a) we need to build ontology which reflect the classifier's (or the user's) point of view on the world ($\mathcal{O}^C \sqsubseteq \mathcal{O}$) and (b) we need to design tools that interpret a schema element as the user interprets it. These two problems are not trivial, but they can be addressed with well-known methods belonging to disciplines like ontology engineering and knowledge representation. Ontology engineering can help us to design better ontologies, e.g. ontologies that appropriately represent what an individual or a community knows on a given domain; knowledge representation gives us methods for representing the meaning of different types of schemata, beyond classifications.

To conclude, we see our work as a small step towards a much more general goal, namely the construction of a theory which explains how semantically autonomous entities (agents) can communicate without presupposing a beforehand agreement on how things should be represented. In other words, a theory of the role of meaning coordination in a theory of (inter)action. A lot remains to be done, but this goes beyond the scope of this paper.

References

1. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, January 2003.
2. Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.
3. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: a new approach and an application. In K. Sycara, editor, *Second International Semantic Web Conference (ISWC-03)*, Lecture Notes in Computer Science (LNCS), Sanibel Island (Florida, USA), October 2003.
4. Jeremy Carroll and Hewlett-Packard. Matching rdf graphs. In *Proc. in the first International Semantic Web Conference - ISWC 2002*, pages 5–15, 2002.
5. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of WWW-2002, 11th International WWW Conference, Hawaii*, 2002.
6. J. Euzenat and P. Valtchev. An integrative proximity measure for ontology alignment. *Proceedings of the workshop on Semantic Integration*, October 2003.
7. F. Giunchiglia and P. Shvaiko. Semantic matching. *The Knowledge Engineering Review Journal*, 18(3):265–280, 2003.

8. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS*, pages 61–75, 2004.
9. Ryutaro Ichisem, Hiedeaki Takeda, and Shinichi Honiden. Integrating multiple internet directories by instance–base learning. In *AI AND DATA INTEGRATION*, pages 22–28, 2003.
10. Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *The VLDB Journal*, pages 49–58, 2001.
11. Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 122–133, 24–27 1998.
12. Marcello Pelillo, Kaleem Siddiqi, and Steven W. Zucker. Matching hierarchical structures using association graphs. *Lecture Notes in Computer Science*, 1407:3–??, 1998.
13. H. Putnam. *Reason, Truth, and History*. CUP, 1981.
14. Jason Tsong-Li Wang, Kaizhong Zhang, Karpjoo Jeong, and Dennis Shasha. A system for approximate tree matching. *Knowledge and Data Engineering*, 6(4):559–571, 1994.
15. K. Zhang, J. T. L. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems. In Z. Galil and E. Ukkonen, editors, *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching*, volume 937, pages 395–407, Espoo, Finland, 1995. Springer-Verlag, Berlin.

Debugging and Semantic Clarification by Pinpointing

Stefan Schlobach

Department of Computer Science,
Vrije Universiteit Amsterdam, The Netherlands
`schlobac@few.vu.nl`

Abstract. Ontologies are the backbone of the Semantic Web as they allow one to share vocabulary in a semantically sound way. For ontologies, specified in OWL or a related web ontology language, Description Logic reasoner can often detect logical contradictions. Unfortunately, there are two drawbacks: they lack in support for debugging incoherence in ontologies, and they can only be applied to reasonably expressive ontologies (containing at least some sort of negation).

In this paper, we attempt to close these gaps using a technique called *pinpointing*. In pinpointing we identify minimal sets of axioms which need to be removed or ignored to turn an ontology coherent. We then show how pinpointing can be used for *debugging* of web ontologies in two typical cases. More unusual is the application of pinpointing in the *semantic clarification* of underspecified web ontologies which we experimentally evaluate on a number of well-known web-ontologies. Our findings are encouraging: even though semantic ambiguity remains an issue, we show that pinpointing can be useful for debugging, and that it can significantly improve the quality of our semantic enrichment in a fully automatic way.

1 Introduction

Ontologies play a crucial role in the Semantic Web (SW), as they allow “intelligent agents” to share information in a semantically unambiguous way, and to reuse domain knowledge (possibly created by external sources). However, this makes SW technology highly dependent of the quality, and, in particular, of the correctness of the applied ontology. Two general strategies for quality assurance are predominant, one based on developing more and more sophisticated ontology modeling tools, the second one based on logical reasoning. In this paper we will focus on the latter. With the advent of expressive ontology languages such as OWL and its close relation to Description Logics (DL), non-trivial implicit information, such as the *is-a* hierarchy of classes, can often be made explicit by logical reasoners. More crucially, however, state-of-the art DL reasoners can efficiently detect incoherences even in very large ontologies. The practical problem remains what to do in case an ontology has been detected to be incoherent.

Suppose, for example, that an academic researcher wants to make information about his work available in OWL on the Internet. In the spirit of the Semantic

Web, he refers to an existing external ontology about research environments using the ontology *cerif* at [5]. Studying it with an available ontology browser, he finds that the available classes are very useful and that the hierarchy corresponds to his intentions. Unfortunately, the connected logical reasoner tells him that both the concepts *Faculty* and *Institute* are unsatisfiable. What is he supposed to do? Things are even more difficult when our researcher tries to use two or more different ontologies with multiple ownership. As an example, assume that you want to use both the *SUMO* and the *CYC* upper ontologies in a single document. Unfortunately, this attempt leads to over 1000 unsatisfiable concepts.

A second obstacle is that ontologies are often specified in a very limited way. Even though the OWL representation language offers high expressiveness most state-of-the-art online ontologies use fragments of minimal expressiveness.¹ The most prominent example is when the modeling ignores disjointness information of atomic classes, as this renders most logical method for quality assurance useless.² This means that, in order to re-introduce logical methods for quality control, we need to semantically enrich the ontology first.

To summarize, to ensure the quality of web-ontologies, there are two major problems to be solved: We call **semantic clarification** the process of automatically enriching ontologies by appropriate disjointness statements which reintroduces the logical functionality in the process of ontological quality control. Secondly, even though incoherences in ontologies might be detected easily, there is usually little support for the identification and elimination of the modeling errors, a problem we call **debugging**. Debugging is a non-trivial process as it is not unusual to deal with over 700 unsatisfiable classes in real applications.

This paper intends to give a *qualitative analysis* over the potential of non-standard reasoning techniques for debugging and semantic clarification. For this purpose, we propose a method called *pinpointing*. Pinpointing is a pragmatic and rather simple technique based on debugging methods we introduced in [16] and boils down to ignoring those ontological axiom which are most likely to be responsible for the incoherence. Before plunging into a more theoretical study of this simple type of *reasoning with inconsistency* we decided to empirically study the practical effect of pinpointing in the Semantic Web. For this purpose we discuss two applications of pinpointing for debugging of ontologies, as introduced in the two previously described scenarios. First, there is the most obvious case of published ontologies which are originally incoherent, and secondly, we consider incoherence as a result of the merging of two or more ontologies. To assess the application of pinpointing for semantic clarification we undertook a number of experiments. We automatically create disjointness statements for “underspecified” ontologies by assuming that all the direct siblings in a well-defined **is-a**

¹ Consider that not even 10% of the ontologies in [5] contain negation.

² Inconsistencies in an ontology can be caused by erroneous use of a variety of different language constructs, such as cardinality or role restrictions. The most common reason is disjointness of concepts, though, and we will focus on this class of problem in this paper. None of the described techniques, however, depends on, or is restricted to this particular kind of inconsistencies.

hierarchy should be disjoint. In practice, this assumption is often too strong, and introducing these disjointness statements usually results in a hugely incoherent ontology. We show that, by applying our pinpointing strategy, we can often reduce the number of erroneous disjointness statements significantly.

The main contribution of this paper is threefold: we combine previously known debugging techniques into a framework called pinpointing, which we present and evaluate in the Semantic Web context. Moreover, we show that the application of pinpointing can significantly improve the quality of semantic clarification, a process which in itself is useful for quality assurance of ontologies.

The remainder of this paper is organized as follows: after a brief overview of related work, we give the formal notion of pinpointing in Section 3. In Section 4 we describe three typical cases of pinpointing for debugging of web ontologies. Finally, Section 5 describes our experiments on Semantic Clarification.

2 Related Work

The building of ontologies has been discussed extensively in the literature, and many links can be found on the W3C website [18] about modeling methodology and ontology languages. For Description Logics the handbook [1] is an excellent reference. Explanation has been an issue in the DL community for several years, but most papers, such as [3], deal with subsumption. The number of papers dealing with theoretical studies of reasoning with inconsistency is enormous (just to mention [9, 15]). Our approach in this paper, however, is more humble, as we only investigate the effect of a simple reasoning strategy in an empirical way.

From a more practical point of view, closest to our work are the Chimaera and PROMPT tools ([10] and [12]), which provide support for the merging, analysis and diagnosis of a knowledge base but not, to our knowledge, for debugging. The techniques, that we use to calculate our minimal incoherence preserving sub-TBoxes (MIPs), however, are similar to those introduced in [2], where the authors use Boolean minimization to calculate minimal inconsistent ABoxes to construct an implicit minimal model for defaults. Finally, our work on semantic clarification is based on the ideas of [4].

3 Pinpointing in Web Ontologies

This paper is about debugging incoherent ontologies in the Semantic Web, i.e., about detection and elimination of logical contradictions in machine-readable formalisations of the shared vocabulary of a particular application domain. We first explain our use of the term ontology and formally define what we mean by incoherence, before introducing our debugging methodology.

3.1 Ontologies in the Semantic Web

In recent years, several semantic web languages have been developed to support the integration of ontologies, and OWL has now been accepted as the recommen-

dation of the W3C as the standard web ontology language [18]. It builds on the XML surface syntax, XML Schema datatypes, the RDF datamodel and RDFS semantics for hierarchies of classes and properties, and it adds more vocabulary to build complex classes and properties. Among others it provides relations between classes, cardinality, enumeration of individuals and Boolean operators. An OWL ontology consists mainly of axioms and facts, where the latter describe properties of individuals and where the former associate classes and properties with possibly complex information. Examples for OWL axioms in the abstract syntax are *Class(Mollusk partial Invertebrate)* and *disjointWith(Mollusk Worm Arthropod)* which states that mollusks are invertebrates but not worms or arthropods. In this paper, we will focus on axioms, and other language constructs available in OWL. OWL has a number of predecessor languages, such as DAML, OIL and DAML-OIL, and many ontologies currently available are simply defined in RDF or RDFS [8]. For the purpose of this paper, we make no distinction between the different types of ontologies and refer to any online collection of axioms in one of the above formalisms as a *web ontology*. On top of RDF, OWL is also rooted in other frameworks such as frames and Description Logics (DL). The connection to DL is useful, as it provides model-theoretic semantics with formally defined reasoning, for which there are several highly optimized and efficient reasoners.

Description Logic Reasoning for Ontologies. We shall not give a formal introduction to Description Logics here, but point to Chapter 2 of [1]. Briefly, DLs are set description languages with concepts, interpreted as subsets of a domain, and roles, interpreted as binary relations. In this paper, we will also use the terms concepts (roles) and classes (properties). In a terminological component $\mathcal{T}$ (called TBox), the interpretation of concepts can be restricted to the *models* of $\mathcal{T}$ by defining *axioms* of the form $C \sqsubseteq D$, where C and D are concepts.³ Based on this formal model-theoretic semantics, a TBox can be checked for *incoherence*, i.e., whether there are *unsatisfiable* concepts; concepts which are necessarily interpreted as the empty set in all models of the TBox. Other reasoning services include *subsumption* of two concepts (a subset relation w.r.t. all models of $\mathcal{T}$). Subsumption and incoherence are standard reasoning services available in all DL reasoners, such as FaCT [7] and RACER [6]. Although a DL reasoner can classify an ontology and check for the existence of unsatisfiable concepts efficiently, they offer little support for the detection and elimination of errors, i.e., for debugging.

In [16], we proposed a first step to close this gap by introducing a method to explain the incoherence using the notions MUPS, MIPS and cores, which we

³ Although the definitions and methods in this paper are quite general, the algorithms we implemented and applied in the experiments are restricted to unfoldable $\mathcal{ALC}$ TBoxes. $\mathcal{ALC}$ is a simple yet relatively expressive DL with conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation ($\neg C$) and universal ($\forall r.C$) and existential quantification ($\exists r.C$). A TBox is called *unfoldable* if the left-hand sides of the axioms are atomic, and if the right-hand sides contain no direct or indirect reference to the defined concept [11]. Overall, we will not consider assertional components in this paper.

will recall in the following section. But this is not sufficient for debugging, as we will have to repair an incoherent ontology before being able to use it. In Section 3.3 we propose a strategy for fixing the incoherence by pinpointing.

3.2 Explaining Logical Incoherences

In this section, we study ways of *explaining incoherences* in DL terminologies. The idea is to simplify a terminology $\mathcal{T}$ in order to reduce the available information until only the cause of the incoherence remains. More concretely, we exclude axioms that are irrelevant to the incoherence.

To debug an incoherent terminology, we have to identify and eliminate debugging-relevant axioms, where an axiom is *relevant* if a contradictory TBox becomes coherent once the axiom is removed or, at least, a particular, previously unsatisfiable concept becomes satisfiable. Consider the following (incoherent) TBox $\mathcal{T}_1$, where A, B and C are primitive and $A_1, \dots, A_7$ defined concept names:

$ax_1 : A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3$	$ax_2 : A_2 \sqsubseteq A \sqcap A_4$
$ax_3 : A_3 \sqsubseteq A_4 \sqcap A_5$	$ax_4 : A_4 \sqsubseteq \forall s. B \sqcap C$
$ax_5 : A_5 \sqsubseteq \exists s. \neg B$	$ax_6 : A_6 \sqsubseteq A_1 \sqcup$
$ax_7 : A_7 \sqsubseteq A_4 \sqcap \exists s. \neg B$	$\exists r. (A_3 \sqcap \neg C \sqcap A_4)$

The set of unsatisfiable concept names as returned by a complete DL reasoner is $\{A_1, A_3, A_6, A_7\}$. Although this is still of manageable size, it hides crucial information, e.g., that unsatisfiability of A_1 depends on unsatisfiability of A_3 , which is incoherent because of the contradiction between A_4 and A_5 . We will use this example to explain our explanation methods.

Unsatisfiability-preserving sub-TBoxes of a TBox $\mathcal{T}$ and an unsatisfiable concept A are subsets of $\mathcal{T}$ in which A is unsatisfiable. In general, there are several of these sub-TBoxes: and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

Formally, let A be a concept which is unsatisfiable in a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal unsatisfiability-preserving sub-TBox (MUPS)* of $\mathcal{T}$ if A is unsatisfiable in $\mathcal{T}'$, and A is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$. We will abbreviate the set of MUPS of $\mathcal{T}$ and A by $mups(\mathcal{T}, A)$. MUPS for our example TBox $\mathcal{T}_1$ and its unsatisfiable concepts are:

$$\begin{aligned}
 mups(\mathcal{T}_1, A_1) &= \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\} \\
 mups(\mathcal{T}_1, A_3) &= \{\{ax_3, ax_4, ax_5\}\} \\
 mups(\mathcal{T}_1, A_6) &= \{\{ax_1, ax_2, ax_4, ax_6\}, \\
 &\quad \{ax_1, ax_3, ax_4, ax_5, ax_6\}\} \\
 mups(\mathcal{T}_1, A_7) &= \{\{ax_4, ax_7\}\}
 \end{aligned}$$

MUPS are useful for relating unsatisfiability to sets of axioms but are also the basic ingredients for the calculation of *Minimal Incoherence Preserving Sub-terminologies*, which are the smallest subsets of an original TBox preserving unsatisfiability of at least one atomic concept.

Formally, let $\mathcal{T}$ be an incoherent TBox. A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal incoherence-preserving sub-TBox (MIPS)* of $\mathcal{T}$ if $\mathcal{T}'$ is incoherent, and every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent. We will abbreviate the set of MIPSs of $\mathcal{T}$ by $mips(\mathcal{T})$. For our example terminology $\mathcal{T}_1$ we get three MIPSs $mips(\mathcal{T}_1) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$. It can easily be checked that each of the three incoherent TBoxes in $mips(\mathcal{T}_1)$ is indeed a MIPS since taking away a single axiom renders each of the three coherent. The first one signifies, for example, that the first two axioms are already contradictory without reference to any other axiom, which suggests a modeling error already in these two axioms.

Minimal incoherence-preserving sub-TBoxes identify the smallest sets of TBox axioms that cause the original TBox to be incoherent. In terminologies such as DICE, which are created through migration from other representation formalisms, there are several such sub-TBoxes, each corresponding to a particular contradictory terminology. *Cores* are now sets of axioms occurring in several of these incoherent TBoxes. The more MIPSs such a core belongs to, the more likely it is that axioms are the cause of contradictions.

Formally, a non-empty intersection of n different MIPSs in $mips(\mathcal{T})$ (with $n \geq 1$) is called a *MIPS-core of arity n (or n -ary core)* for $\mathcal{T}$. Every set containing precisely one MIPS is, at least, a 1-ary core. The most interesting cores of a TBox, $\mathcal{T}$, are those with axioms that are present in as many MIPSs of $\mathcal{T}$ as possible, i.e., having maximal arity. On the other hand, the size of a core is also significant, as a larger core indicates that clusters of axioms cause contradictions in combination only. In our example, axiom ax_4 occurs both in $\{ax_3, ax_4, ax_5\}$ and $\{ax_4, ax_7\}$, which makes $\{ax_4\}$ a core of arity 2 and size 1 for $\mathcal{T}_1$, which is the core of maximal arity in this example.

There is no unique way of deciding which axioms in the MIPS are the most relevant for the incoherence. Our approach is pragmatic: we assume that an axiom is more likely to be erroneous the more often it occurs in the set of MIPS. Therefore, we look for cores with maximal arity.

The general definitions of MIPS, MIPS and cores do not depend on a particular ontology language and can easily be extended to include facts about individuals. However, the algorithms we implemented are restricted to unfoldable $\mathcal{ALC}$ terminologies. It was shown in [16] that the problem of calculating MIPS for an $\mathcal{ALC}$ terminology is in PSPACE. Nevertheless, in our experience calculating all the MIPS was practically feasible in all but a few cases.⁴ This is due to the relatively simple structure of the ontologies we considered. For more complex cases, approximative methods need to be investigated. Similarly, checking elements of $mips(\mathcal{T})$ for cores of maximal arity requires exponentially many checks in the size of $mips(\mathcal{T})$. For efficiency reasons, we therefore currently only check for cores of size 1.

⁴ The overall run-times for the experiments described in Section 4 and 5 were minutes rather than hours even for the most complex ontologies.

3.3 A Strategy for Fixing Incoherences

MIPSSs, MUPS and cores do not offer an immediate recipe for fixing an incoherent web ontology. For web ontologies we propose a strategy which iteratively calculates the cores (of size 1) of maximal arity, repeating the process on the remaining MIPSSs. Moreover, the algorithms for MIPS and MUPS have been implemented for terminological debugging of $\mathcal{ALC}$ terminologies. To apply them to web ontologies we need some preprocessing. Given a web ontology $\mathcal{O}$ we apply the following steps:

1. Remove the ABox, as well as property statements and axioms where the left-hand side is non-atomic.
2. Replace equivalence statements by implications
3. Collect all implications $C \sqsubseteq D_1, \dots, C \sqsubseteq D_n$ in a single conjunction $C \sqsubseteq D_1 \sqcap \dots \sqcap D_n$.
4. Call the resulting terminology $\mathcal{T}$ (not necessarily unfoldable)

The strategy for automatically fixing the incoherences by pinpointing is then as follows; calculate:

1. the set $Unsat(\mathcal{T})$ of unsatisfiable concept-names in $\mathcal{T}$ using RACER;
2. the MUPS $mups(\mathcal{T}, CN)$ for all concept-names $CN \in Unsat(\mathcal{T})$;
3. the MIPSSs $mips(\mathcal{T})$ for $\mathcal{T}$ from the MUPS;
4. let $M := mips(\mathcal{T})$, $P(\mathcal{O}) = \emptyset$. Now calculate while $M \neq \emptyset$:
 - (a) the core $\{ax\}$ of M of size 1 with maximal arity, and add it to $P(\mathcal{O})$;
 - (b) remove from M the mips containing ax .
5. $P(\mathcal{O})$ will be called the *pinpoint* of $\mathcal{O}$.
6. Finally, remove $P(\mathcal{O})$ from $\mathcal{T}$.

The pinpoint of the ontology $\mathcal{O}$ is a set of axioms in the preprocessed version. Every axiom corresponds precisely to a concept-name (because of step 3) or is a disjointness statement. By the pinpoint of an ontology, we will therefore refer both to sets of axioms as well as to sets of concept and disjointness statements. Note that for debugging and semantic clarification, we often focus on these pinpoints. This is because there is usually only a handful of those as compared to hundreds of MIPSSs, which can be quite complicated. However, in practice, MIPSSs will always have to be consulted if one wants to understand the underlying reasons for incoherence of an ontology.

Fixing an ontology by pinpointing offers a simple solution to the incoherence problem since by adjusting or removing the information about the pinpoints we can guarantee the restoration of logical coherence with (almost) minimal intrusion in the ontology. Pinpoints correspond to hitting-sets [14] for the MIPS, but they are not necessarily minimal. Take, as example, a set $M = \{\{ax_1, ax_2\}, \{ax_3, ax_4\}, \{ax_5, ax_1\}, \{ax_5, ax_3\}\}$ of MIPS, with $\{ax_5, ax_1, ax_3\}$ as possible pinpoint, even though $\{ax_1, ax_3\}$ is a minimal covering set for M . On the other hand, it has to be remarked that calculating minimal hitting-sets from the set of MIPS is NP-COMPLETE, as compared to linear time to calculate pinpoints from the MIPS.

We evaluated our strategy of debugging of web ontologies by pinpointing on a number of applications. The goal of our experiments was twofold, first to assess some case studies on more typical applications of debugging, but also to put forward a more unconventional approach to resolving semantic ambiguity that we claim can be supported by our proposed methods.

4 Debugging Web-Ontologies by Pinpointing

Pinpointing was initially developed for debugging of the medical terminology DICE. To evaluate whether the method use useful for and scales to web ontologies we looked at two case studies: first, importing an unsatisfiable ontology, and, secondly, merging several ontologies.

4.1 Import an *Institute* and Ignore the *Faculty*:

Remember our introductory example from the academic, who plans to use the *cerif* ontology, which defines two concepts in unsatisfiable ways. This example demonstrates the simplest application of debugging by pinpointing, namely when a user wants to import an incoherent ontology. Our suggested strategy for debugging is as follows: whenever we detect an incoherence in an ontology we trace down the most likely source of the logical contradiction by calculating the pinpoint for it. For the *cerif* ontology there are two MIPSs with a single concept *Faculty* occurring in both, i.e., the pinpoint contains only *Faculty*. This tells us that the source of the logical incorrectness of the definition of *Institute* is probably the incorrect definition of *Faculty*. At least theoretically, this would allow the user to “rescue” the *Institute* class, by overwriting or simply ignoring the *Faculty* concept.

Although the *cerif* is a real-world example, incoherent ontologies are rarely published on-line. In contrast, debugging at creation or migration time is quite typical. Let us describe how pinpointing was used when the DICE terminology was migrated from frames to a Description Logics representation.

4.2 Is a Physical Quantity Temporal or Mathematical?

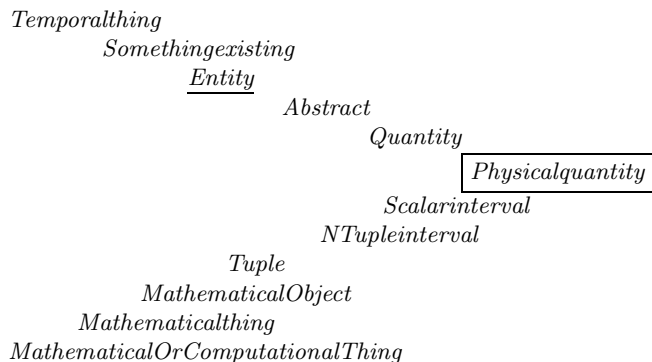
The previous example illustrated the use of debugging by pinpointing of a single ontologies. Most people, however, will use several ontologies and in this case, might be even more likely to encounter incoherence. In the Introduction we mentioned the example of using both the *SUMO* and *CYC* upper ontologies. As they are topic-related, and as *CYC* provides disjointness statements, there is indeed a high number of unsatisfiable concepts. Let us define our observation in more detail, starting with a description of the ontologies.

- *SUMO*: the *Suggested Upper Merged Ontology* is an upper level ontology suggested by the IEEE Suggested Upper Ontology Working group. It was created by the Teknowledge Corporation and was made publicly available at <http://ontology.teknowledge.com> [17].

- **CYC**: OpenCyc is the open source version of the Cyc technology, “the world’s largest and most complete general knowledge base.” **CYC** includes about 6,000 concepts – an upper ontology for all of human consensus reality – and 60,000 assertions about the 6,000 concepts, interrelating them, constraining them, in effect (partially) defining them. <http://www.opencyc.org>. [13]

In our experiments we removed the unique name-spaces from both **SUMO** and **CYC** and syntactically merged the two ontologies. We then used **RACER** to check for coherence, which resulted in an un-ordered list of 1093 unsatisfiable concepts. Initially this seems to be a discouraging result. However, the pinpoint of the joint ontology contains only 4 concept names *Event*, *Product*, *Entity* and *Tuple*. This is surprising as it means that we can ensure coherence of the merged ontology by ignoring (or fixing) the axioms defining *Event*, *Product*, *Entity* and *Tuple*.

Let us try to get a better intuition for the problem. Each of the four concepts defined by the axioms in the pinpoints occur in contradictions in a variety of different combinations of other concepts. Let us look at one of the MIPSs, here in form of the derivation of one of the contradictions, namely of the concept *Physicalquantity*. Here, the upward derivation stems from the **SUMO** hierarchy and the downward derivation from **CYC**, as a *Physicalquantity* is defined as a *quantity* in **SUMO** and a *Scalarinterval* in **CYC**. A contradiction occurs as the class *MathematicalOrComputationalThing* is defined to be disjoint from *Temporalthing* in **CYC**. Interestingly enough it is definition of the concept *Entity* as a *Temporalthing* that is to be ignored by our strategy, which is what constitutes one of the “philosophical” difference between the **SUMO** and **CYC** ontologies.



For the naive user of the two ontologies, the practical benefit of pinpointing is immediate. Discarding the axioms defining the four elements of the pinpoint will directly render the ontology coherent and all the remaining 1089 previously unsatisfiable concepts satisfiable, and therefore usable again. This restores most⁵ of the useful reasoning facilities which users have learned to expect from their ontology development tools.

⁵ Removing the axioms obviously has side effects. In the case of the concept *Entity* this implies that none of the descendants of *Entity* can be classified as a **TEMPORALTHING** any more. In this sense a certain **CYC** view of the world predominates.

5 Semantic Clarification by Pinpointing

To improve the quality of the DICE terminology, the developers migrated a frame-based to a DL-based representation. For this purpose, they made a number of very strong assumptions, as they had to decide on the semantic interpretations of operators such as slot-fillers or super-classes. One of the main issues was to make disjointness between classes explicit. In the frame-based version of DICE, it is impossible, for example, to state that nothing can be both a *Liver* and a *Kidney*, i.e., that the classes are disjoint. However, a controlled medical terminology should prohibit the definition of a particular organ as a subclass of $Liver \sqcap Kidney$. In [4] Cornet and Abu-Hannah discuss a number of assumptions they base their migration on. The most interesting for this paper is the:

Strong Disjointness Assumption (SDA): In a well-modeled terminology the direct siblings, i.e. children of a common parent in the subsumption hierarchy should be disjoint.

In a Semantic Web application, we face a similar dilemma as in the migration of DICE. Many publicly available ontologies have been created by migration from frame-based representations, but usually, relatively weak a priori assumptions were made in the migration process (such as not including any disjointness statements). In this paper, we propose a more rigid migration strategy that should help the user end up with a version of his/her ontology where at least some basic disjointness relation between atomic concepts has been established. This is achieved by applying the Strong Disjointness Assumption and using a mechanism to deal with exceptions. In our case, this mechanism is based on pinpointing. Let us describe the general strategy.

5.1 The Strategy for Semantic Clarification

Given a semantically weakly specified ontology $\mathcal{O}$ (without disjointness statements) we, first, add all possible disjointness statements according to the SDA to $\mathcal{O}$ before debugging the ensuing incoherences by pinpointing. More formally this consists of the following steps:

1. Use RACER to classify $\mathcal{O}$.
2. Let $D = \{\{C_1^1, \dots, C_{n_1}^1\}, \dots, \{C_1^m, \dots, C_{n_m}^m\}\}$ be the set of all sets of concept names which have at least one common parent in the subsumption hierarchy.
3. The set $sug\_disj(\mathcal{O}) = \{disjoint(C_1^1, \dots, C_{n_1}^1), \dots, disjoint(C_1^m, \dots, C_{n_m}^m)\}$ contains all the disjointness statements suggested given the SDA.
4. Add $sug\_disj(\mathcal{O})$ to $\mathcal{O}$ to create the possibly incoherent $\mathcal{O}^* = \mathcal{O} \cup sug\_disj(\mathcal{O})$.
5. Calculate the pinpoint $P(\mathcal{O}^*)$ for $\mathcal{O}^* = \mathcal{O} \cup sug\_disj(\mathcal{O})$
6. Remove the disjointness statements $D \in P(\mathcal{O}^*)$ from $\mathcal{O}^*$ to make it coherent.

5.2 Experiments

To evaluate clarification by pinpointing we applied our strategy on ontologies, first, to assess the quality of the added disjointness statements given our strong disjointness assumption, and secondly, to study how much pinpointing can improve on these results. We do this in three steps, first, we evaluate the relation of the size of an ontology with the damage inflicted on the ontology by adding disjointness statements. Secondly, we look at the quality of the disjointness statements themselves. Finally, we check whether more information improves the semantic clarification. First, however, we discuss the data used for the experiments.

The Data (Web Ontologies): The problem with an evaluation such as ours is that it requires domain knowledge to evaluate the quality of the disjointness statements. Therefore, we focused on general knowledge ontologies and on ontologies describing domains where we have some expertise (such as soccer). We used the following ontologies:

- **MGED**: provides standard terms for the annotation of microarray experiments in order to enable structured queries on those experiments;
- **UNSPSC**: a translated version of the *Universal Standard Products and Services Classification Code* which provides an open, global multi-sector standard for efficient, accurate classification of products and services;
- **soccer**: “concepts that are specific to soccer: players, rules, field, supporters, actions, etc. Used to annotate videos.” [5]
- **SUMO**: (as described above);
- **MILO**: a midlevel ontology that acts as a bridge between the high-level abstractions of the **SUMO** and the low-level detail of the domain ontologies;
- **Eco**: an ontology describing properties specific to the economy;
- **Trans**: an ontology of terms about transportation-related information;
- **Gov**: an ontology of government concepts;
- **Geo**: an ontology of geography.

The last 6 ontologies were all made available by the Teknowledge Corporation and more details can be found at [17]. Most information for the last 4 ontologies is taken from the CIA World Fact Book (2002), but many other sources are used.

Question 1: Is the Size of an Ontology Relevant for Semantic Clarification? In the first set of experiments, we wanted to study what role the structure and size of an ontology plays when adding disjointness statements for clarification. For this purpose, we added disjointness statements to the ontologies described above and calculated the set of unsatisfiable concepts and the MIPSs. We take a high number of unsatisfiable concepts and many MIPSs as an indicator that the Strong Disjointness Assumption is inadequate for the given ontology. Table 1 gives an overview of the number of implications (the only axioms we consider), the number of disjointness statements created, the number of unsatisfiable concepts in the ontology with the disjointness statements, and the number of MIPSs.

Table 1. Adding disjunctions to web ontologies

$\mathcal{O} =$	MGED	UNSPSC	soccer	SUMO	MILO	Eco	Trans	Gov	Geo
#axioms	370	9795	194	669	1764	409	455	50	417
#disj	34	1463	40	169	342	60	89	12	82
#Unsat($\mathcal{O}^*$)	72	0	0	175	46	213	145	0	11
#mips($\mathcal{O}^*$)	42	0	0	149	59	154	189	0	22

Note that we have small ontologies which become incoherent and that the big ontology UNSPSC remains coherent even after adding over 1000 disjointness statements. It shows that the size of an ontology is not the main factor in semantic clarification. There is not even a unique pattern for the 4 specialized ontologies provided by Teknowledge: whereas the geography ontology **Geo** has only very few (11) unsatisfiable concepts, there are now 89 unsatisfiable concepts in the transportation ontology **Trans** which is of similar size, although one would expect comparable modeling. It is clear that a more careful analysis is needed, taking a closer look at the created disjointness statements.

Question 2: How Useful are Disjointness Statements? In the next step of the experiments, each of the created disjointness statements was evaluated by a human assessor as true or false. In some cases, the evaluators did not have enough domain knowledge so the numbers do not always add up. Unfortunately, we could not further experiment with UNSPSC and MILO because of their size and we did not consider MGED because of our lack of expert knowledge. Table 2 summarizes the results for the evaluation of the quality of the created disjointness statements and our pinpointing based debugging method.

Remember that the soccer ontology **soccer** and both **Gov** and **Geo** were coherent when adding the disjointness statements so that, obviously, no MIPS could be found. Astonishingly, there is again a big discrepancy between the **Trans** and the **Geo** ontology, which does not even disappear after removing the pinpointed disjointness statements. It is difficult to see the reasons for this odd behavior of two ontologies that are, in principle, of a very similar structure, size and pedigree, but we suppose that it is due to differences in modeling. We look now at some examples to get a better understanding of what might go wrong.

Error Analysis: The **soccer** ontology adds a level of structure to the class of players by separating goalkeepers, other players and substitutes. This, however, is strange modeling practice, as goalkeepers can be substitutes. The disjointness statement (*disjoint Goalkeeper OtherPlayer Substitute*) which would be added by our clarification strategy would therefore be erroneous. As we do not have more information about goalkeepers and substitutes, this error cannot be found using our pinpointing strategy.

In the **Geo** ontology volcanoes and upland areas are both classified as landforms, which suggests the addition of (*disjoint ... UplandArea Volcano*) (we will

Table 2. Evaluating the Quality of the Disjointness Statements

$\mathcal{O} =$	soccer	SUMO	Eco	Trans	Gov	Geo
#disj	40	169	60	89	14	82
#false disj	5	63	14	49	7	19
#unknown	0	8	0	17	0	8
#correct disj	35	98	46	23	7	55
#correct in %	87%	60%	76%	31%	50%	74%
#improved by removing $P(\mathcal{O})$	0	25	5	15	0	5
#correct after improving in %	87%	76%	85%	52%	50%	81%

refer to this statement as DISS) according to our clarification strategy. Here, pinpointing can help, as we now have an unsatisfiable concept *VolcanicMountain* with two MIPs $\{Mountain, Volcano, VolcanicMountain, DISS\}$ $\{UplandArea, Mountain, VolcanicMountain, DISS\}$, and a subsequent pinpoint DISS.⁶ What this suggests is the following: pinpoints can be useful when there is enough specialized knowledge to make the exceptions to the disjointness statements explicit. In the above case, this is the knowledge about a volcanic mountain, which is both an Upland and a Volcano, helps us find the error.

The observation that more specialized knowledge helps to improve the quality of the pinpointing leads us to a final round of experiments on the SUMO ontology.

Question 3: Does More Knowledge Help Debugging? To assess the claim that additional information can make pinpointing more successful, we combined SUMO with the more specialized ontologies MILO, Eco, Trans, Gov and Geo from the Teknowledge family. Remember that with pinpointing, we were able to improve the quality of the added disjointness statements by 16% in our previous experiment, but 38 statements remained erroneous. For the last experiment, we added to SUMO the 169 disjointness statements following from the Strong Disjointness Assumption and removed from it the 25 occurring in the pinpoint. The resulting ontology is coherent. Adding MILO to this ontology, however, creates new incoherences, and we find a further 15 erroneous disjointness statements. Adding the 4 specialized ontologies Trans, Eco, Geo and Gov also helps detecting 2 more false disjunctions, which leaves the new SUMO (with pinpoints removed) with 86% correct disjointness statements (up from 60%).

Erroneous Disjointness Statements? Even after pinpointing we are left with 14% incorrect disjointness statements in the above experiment. This sounds dangerous, but has to be put in the perspective of the application. Remem-

⁶ If an incoherent terminology is the result of a semantic clarification process, it is part of the assumption that some of the new, artificially created, statements are likely to be incorrect. Therefore, whenever we have a choice, (as in this example) we include the disjointness statement into the pinpoint.

ber, that disjointness statements are only interesting for reasoning to establish coherence, and that they do not influence the subsumption hierarchy, given that the ontology is indeed coherent. What remains is the danger that logical contradictions occur when creating new objects referring to semantically enriched class. But then, again, pinpointing offers a simple solution: if it is indeed the disjointness axiom that causes the contradiction it alone will constitute the pinpoint, and will automatically be removed to render the new object satisfiable.

6 Conclusion and Further Work

With the arrival of more expressive ontology languages in the Semantic Web community, incoherences increasingly become a problem that can seriously hamper the construction and application of web ontologies. In this paper we presented a strategy for automatically identifying and fixing incoherences that is based on first explaining their causes and, secondly, choosing (and eliminating) axioms that most frequently participate in the underlying logical contradictions.

We discussed our pinpointing strategy with respect to standard debugging of incoherences. Then, we showed how pinpointing can help the semantic clarification of underspecified ontologies. For the first case-studies, we checked a number of web ontologies for coherence and explored what happens when the two upper-level ontologies *CYC* and *SUMO* are merged. The most interesting finding was that, although there was a very high number of unsatisfiable concepts, the pinpoint only consisted of 4 elements. We suggest that studying these concepts in more detail can help in clarifying the alternative modeling approaches.

The second case-study was an attempt to clarify the semantics of web ontologies for which no disjointness (or negation) of classes is specified. In this case, incoherence is impossible, but we can automatically add disjointness statements if we assume that direct siblings in a hierarchy should be disjoint. We evaluated this assumption on a number of web ontologies by first including these disjointness statements and, subsequently, applying our pinpointing strategy. The empirical and qualitative results showed that too many disjointness statements remained because not all exceptions to our assumption were covered. However, we already outlined a way out of this dilemma, which is to extend general ontologies with more specific ontologies to better identify exceptions and therefore to exclude the erroneous disjointness statements. Conceptually, it should be a simple extension to add facts, but the additional computational complexity might render the approach infeasible in practice.

Finally, it remains to study our pinpointing approach to reasoning with incoherent ontologies from a theoretical perspective. Surely enough ignoring the ontological axiom from the cores renders the ontology coherent, but nothing can be said so far about the quality (such as maximality or meaningfulness) of the resulting ontology.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. Technical Report RR-93-20, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1993.
3. A. Borgida, E. Franconi, and I. Horrocks. Explaining *ACC* subsumption. In *Proc. of the 14th Eur. Conf. on Artificial Intelligence*, pages 209–213, 2000.
4. R. Cornet and A. Abu-Hanna. Evaluation of a frame-based ontology. A formalization-oriented approach. In *Proceedings of MIE2002, Studies in Health Technology & Information*, volume 90, pages 488–93, 2002.
5. DAML ontology library. URL: <http://www.daml.org/ontologies/>.
6. V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, number 2083 in LNAI, 2001.
7. I. Horrocks. The FaCT system. In H. de Swart, editor, *Tableaux'98*, number 1397 in LNAI, pages 307–312, 1998.
8. I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
9. Pierre Marquis Jrme Lang. Removing inconsistencies in assumption-based theories through knowledge-gathering actions. *Studia Logica*, 67(2):179–214, 2001.
10. D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *The Seventeenth National Conference on Artificial Intelligence*, 2000.
11. B. Nebel. Terminological reasoning is inherently intractable. *AI*, 43:235–249, 1990.
12. N. Noy and M. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 450–455. AAAI Press / The MIT Press, 2000.
13. <http://www.openencyc.org/>.
14. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
15. M. Schaerf and M. Cadoli. Tractable reasoning via approximation. *Artif. Intell.*, 74(2):249–310, 1995.
16. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.
17. Sumo (Suggested Upper Merged Ontology). URL, as visited on April 16, 2004. <http://ontology.teknowledge.com/>.
18. Web-ontology (WebOnt) working group of the W3C. URL, as of April 16, 2004. <http://www.w3.org/2001/sw/WebOnt/>.

An Argumentation Ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT)

Christoph Tempich², H. Sofia Pinto¹, York Sure², and Steffen Staab³

¹ Dep. de Engenharia Informática, Instituto Superior Técnico, Lisboa, Portugal
sofia.pinto@dei.ist.utl.pt
<http://www.dei.ist.utl.pt/>

² Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{sure, tempich}@aifb.uni-karlsruhe.de
<http://www.aifb.uni-karlsruhe.de/WBS/>

³ ISWeb, University of Koblenz Landau, 56016 Koblenz, Germany
staab@uni-koblenz.de
<http://www.uni-koblenz.de/FB4/>

Abstract. A prerequisite to the success of the Semantic Web are shared ontologies which enable the seamless exchange of information between different parties. Engineering a shared ontology is a social process. Since its participants have slightly different views on the world, a harmonization effort requires discussing the resulting ontology. During the discussion, participants exchange arguments which may support or object to certain ontology engineering decisions. Experience from software engineering shows that tracking exchanged arguments can help users at a later stage to better understand the assumptions underlying the design decisions. Furthermore, as the constructed ontology becomes larger, ontology engineers might argue in a contradictory way without knowing so. In this paper we present an ontology which formalizes the main concepts which are used in an DILIGENT ontology engineering discussion and thus enables tracking arguments and allows for inconsistency detection. We provide an example which is drawn from experiments in an ontology engineering process to construct an ontology for knowledge management in our institute. Having constructed the ontology we also show how automated ontology learning algorithms could be taken as participants in the OE discussion. Hence, we enable the integration of manual, semi-automatic and automatic ontology creation approaches.

1 Introduction and Motivation

A prerequisite to the success of the Semantic Web are shared ontologies which enable the seamless exchange of information between different parties. The engineering of a shared ontology is a social process which (1) involves many participants – knowledge engineers, domain experts and users –, (2) may take place at many locations, (3) and is not a once-only process. However, currently available methodologies to support (*cf.* [1]) ontology engineering (OE) for the Semantic Web focus their attention mostly on the centralized development of static ontologies by knowledge engineers and a small

number of domain experts. In [2,3] we propose the DILIGENT OE process which will eventually result in a well tested methodology, to support DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. We believe that this process model is better suited for the requirements on OE in the Semantic Web.

The application of the DILIGENT process model has shown in several case studies that exchange of arguments constitutes a major part in collaborative ontology building. In the SEKT project¹, for example, it was decided that a core upper ontology (PROTON) will be used by all participants. The ontology will be the basis for application development and information sharing. For practical reasons one participant initially provided an initial version, more than eight partners are now discussing the further evolution of the ontology. The initial version does not yet meet the requirements of all partners. Since the participants are distributed in Europe the discussion takes place informally via email. Although the discussion started only recently, it is already difficult for newcomers to enter it since (1) tracing the exchanged arguments is virtually impossible, due to the amount of mails already exchanged, and (2) only some participants know which parts have already been discussed. These problems will be even harder to cope with in a complete distributed environment like the Semantic Web.

This observation from our case study is inline with the experiences made in software and requirements engineering. There, extensions of the IBIS methodology[4] are used to capture design deliberations, thus make them traceable, and formal models have been developed to allow for structured queries on the arguments[5]. They have shown that formal argumentation models enhance traceability of design decision, help in conflict resolution, enhance reusability and facilitates the integration of new participants in the design process. Although, these models are very general, we have identified several requirements – further elaborated in section 5 – for argumentation support and its formalization which are unique for OE processes:

1. General argumentation models allow for all types of arguments and are very flexible. However, we have shown that a restricted set of arguments can facilitate OE processes [2], thus a formal model for OE should take this into account.
2. Within general argumentation models, inconsistencies in the discussion can not be easily detected, since arguers do not formalize their arguments. Ontologies are themselves formal models, thus inconsistencies should be considered during the discussion.
3. Ontology Engineering is often augmented with input from Ontology Learning[6]. No methodology provides an integrated view on manual and automatically created ontologies. An ontology learning algorithm can be seen as an agent providing arguments for design decisions. This should be regarded an integral part in a formalized argumentation model for OE.

In this paper we present an ontology formalized in OWL DL² based on the IBIS argumentation model. With this ontology one can formalize the argumentation taking place in OE processes (*cf.* req. 1), by instantiating the ontology. Note, that the ontology

¹ <http://www.sekt-project.com>

² <http://www.w3.org/TR/owl-ref/>

does not aim at formalizing the exchanged arguments logically. With an appropriate reasoner, inconsistencies in the argumentation can be detected (*cf.* req. 2). Our formal model can be adapted to different types of argumentation, namely arguments from algorithms can easily be integrated (*cf.* req.3).

In the following, we start by providing an extensive review of the state of the art in ontology engineering, argumentation visualization and argumentation structures (section 2). We then present the DILIGENT process template (section 3). Before we go into detail in the argumentation ontology (section 6), we analyze some use cases (section 4) for it and analyze the corresponding requirements (section 5). Finally we give an example drawn from an ontology engineering experiment and conclude.

2 Related Work

The ideas just briefly introduced above and further elaborated in section 4, require research from different areas. In particular we looked into *ontology engineering*, *argumentation visualization*, *argumentation types* and *formal arguments*.

2.1 Ontology Engineering

Established methodologies for ontology engineering summarized in [1], focus on the centralized development of static ontologies. **METHONTOLOGY** [1] and the **OTK methodology** [7] are good examples for this approach. They offer guidance for building ontologies either from scratch, reusing other ontologies as they are, or re-engineering them. They divide OE processes into several sub steps which produce an evaluated ontology for a specific domain.

Holsapple et al.[8] focus their methodology on the collaborative aspects of ontology engineering but still aim at a static ontology. A knowledge engineer defines an initial ontology which is extended and changed based on the feedback from a panel of domain experts. However, no support for argumentation is provided.

In [3] we show how OntoEdit was adopted to support **DILIGENT** OE processes (*cf.* section 3). Discussions are an important part of these kind of processes. However, no support for argument analysis is given.

A methodology which integrates argumentation and ontology engineering in a distributed setting is **HCOME** [9]. It supports the development of ontologies in a decentralized setting and allows for Ontology evolution. It introduces three different spaces in which ontologies can be stored: In the *Personal Space* users can create and merge ontologies, control ontology versions, map terms and word senses to concepts and consult the top ontology. The evolving personal ontologies can be shared in the *Shared Space*. The *Shared Space* can be accessed by all participants. In the shared space users can discuss ontological decisions based on the IBIS [10] model. After some discussion and agreement, the ontology is moved into the *Agreed space*. However, they do not present any experiences or adaptations to the IBIS model for OE.

This is the focus of the work described in [11]. A three-phased knowledge mediation procedure is proposed and evaluated. This approach is especially conceived to integrate different perspectives and information needs into one consensual ontology. They identify useful questions which can guide actors in an ontological discussion. However, they

do not analyze the dominant types of arguments which are used in these discussions. The main finding is that a moderator greatly enhances the efficiency and effectivity of the discussion.

In [4], a case study in building an ontology combining three existing ones is described. In this case study the **Compendium** tool was used to guide the discussion in a synchronous meeting. The results show that structured argumentation is beneficial for ontology engineering. The traceability of the decisions was enhanced. However, the authors were more concerned with the evaluation of their tool than with the specific issues arising in a discussion about an ontology. The authors do not examine which kinds of arguments are exchanged and how the discussion could be made more efficient.

2.2 Argumentation Visualization

As structured argumentation support was identified as beneficial for OE we here summarize the development of this field briefly. The most accepted model of argumentation is the **IBIS methodology** [10]. IBIS was developed to provide a simple yet formal structure for the discussion and exploration of “wicked” problems. Wicked problems cannot be solved in the traditional sense, because one runs out of resources (time, money, energy, people, etc.) before a perfect solution can be implemented.

gIBIS [12] focuses on capturing collaborative deliberations during design activities in the form of graphs containing text at their nodes. It was the first graphical interface for the IBIS methodology. IBIS allows to capture different design deliberations. Appropriate tools can later on help to retrieve them in a sophisticated way. For example the requirements engineering community has long identified the need to capture the arguments exchanged during the design process to enhance traceability. [5] presents an early formalization of the IBIS model. However, the IBIS methodology was criticized due to its resilience to change and for being too abstract. In [13] it is argued that IBIS should be enhanced with domain specific knowledge. The work reported in [14] further enhances the IBIS methodology by introducing an acceptance and rejection mechanism. They emphasize the restriction to record only important considerations.

Compendium[15] builds on the gIBIS system. It is a semantic hypertext tool to capture arguments and visualize them. It offers a conceptual framework for argumentation, it promotes the use of a meeting facilitator and it includes tools to present the exchanged arguments customized for audiences. Compendium tools include *Question based templates* to ease the flow of the arguments. Hence, the discussion can be lead by pre-formulated questions which structure the discussion. A discussion is visualized by different maps, interlinking and connecting the exchanged arguments. Any idea can be expressed in Compendium since its notation is very flexible.

2.3 Types of Arguments

Argumentation models provide a conceptual model for the interaction of issues, ideas and arguments. However, they do not differentiate the different kinds of arguments arising in a discussion. In [2] we analyzed OE discussions with the help of Rhetorical

Structure Theory (RST) [16].³ In our experiments the actors in the OE discussions had the task to agree on a shared ontology to represent the research topics of our institute. We conducted two experiments. In the first the actors were free to discuss the ontology with little guidance, whereas in the second we restricted the types of arguments allowed in the discussion to the ones more relevant and effective to reach consensus identified in the first experiment. The result was that restricting the argument types to Elaboration, Evaluation, Justification, Contrast, Alternative, Example and Counter Example enhances the productivity of OE discussions. Furthermore, the actors kept better track of the discussion, the agreement process was facilitated, and agreement was reached faster.

2.4 Formal Arguments

Formalization of arguments is an important topic in the AI community. Although OWL provides us a formalism that allows to formally state arguments, we do not believe that ontological decisions can be discussed in a completely formal way, at least if the ontology is to be used by humans. Several results show the advantages of using formal models. For example [17] proposes a formal model of argumentation, using the IBIS argumentation model. Based on the formal arguments a preferable solution can be derived. Another interesting application is argument selection based on user needs. For example [18] presents how formal argumentation trees can be pruned to best correspond to users wishes. However, this is not the focus of our work.

2.5 Summary

Review of existing OE methodologies reveals that there is no completely elaborated methodology integrating collaborative, distributed and evolutionary aspects. Moreover, none currently supports the combination of manual and semi-automatic OE approaches. The use of argumentation methodologies for OE was recently recognized but is not yet concisely integrated into the OE process. A formal argumentation model to assist OE is completely missing.

3 DILIGENT Process

In order to provide enough background knowledge about the DILIGENT argumentation ontology, which we present in this paper, we here sketch the overall framework, in which it is embedded, i.e. the overall DILIGENT process (cf. [3]).

Scenario. In *distributed* development there are several experts, with different and complementary skills, involved in collaboratively building the same ontology. For instance, in Virtual Organizations, Open Source and Standardization efforts, experts belong to different competing organizations and are geographically dispersed. In these cases,

³ RST originally offers an explanation of the coherence of texts. It is assumed that for every part of a coherent text there is some function, thus has a particular argument type. Thirty different arguments types have already been identified and loosely defined.

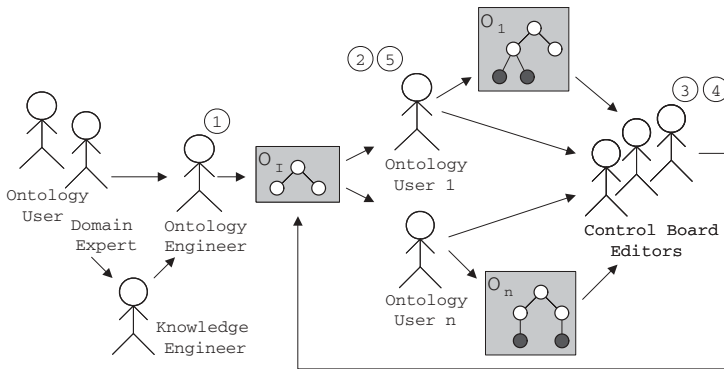


Fig. 1. Roles and functions in distributed ontology engineering

builders typically are also users and, although some users are not directly involved in changing the ontology, they take part in the process by using the ontology.

Process. We will now describe the general process, roles and functions in the DILIGENT process. It comprises five main activities: (1) **build**, (2) **local adaptation**, (3) **analysis**, (4) **revision**, (5) **local update** (cf. figure 1). The process starts by having *domain experts, users, knowledge engineers* and *ontology engineers* **building** an initial ontology. This can be supported by using ontology learning tools. In contrast to known ontology engineering methodologies available in the literature [1, 19, 20] our focus is distributed ontology development involving different stakeholders, who have different purposes and needs and who usually are not at the same location. Therefore, they require online ontology engineering support. The team involved in building the initial ontology should be relatively small, in order to more easily find a small and consensual first version of the shared ontology. Moreover, we do not require completeness of the initial shared ontology with respect to the domain.

Once the product is made available, users can start using it and **locally adapting** it for their own purposes. Typically, due to new business requirements, or user and organization changes, their local ontologies evolve in a similar way as folder hierarchies in a file system. In their local environment they are free to change the reused shared ontology. However, they are not allowed to directly change the ontology shared by all users. Furthermore, the control board collects change requests to the shared ontology.

The board **analyzes** the local ontologies and the requests and tries to identify similarities in users' ontologies. Since not all of the changes introduced or requested by the users will be introduced,⁴ a crucial activity of the board is deciding which changes are going to be introduced in the next version of the shared ontology. The input from users provides the necessary arguments to underline change requests. A balanced decision that takes into account the different needs of the users and meets user's evolving

⁴ The idea in this kind of development is not to merge all user ontologies.

requirements⁵ has to be found. The board should regularly **revise** the shared ontology, so that local ontologies do not diverge too far from the shared ontology. Therefore, the board should have a well-balanced and representative participation of the different kinds of participants involved in the process.

Once a new version of the shared ontology is released, users can **update** their own **local** ontologies to better use the knowledge represented in the new version. Even if the differences are small, users may rather reuse *e.g.* the new concepts instead of using their previously locally defined concepts that correspond to the new concepts represented in the new version.

We have applied this process model to the case of folder sharing via a Peer-to-Peer setting with centralized core folder structures and individual specific folder structures. Our experiences there have substantiated the validity of DILIGENT (cf. [3]).

Threads of Arguments. A central issue in the DILIGENT process is keeping track of threads of exchanged arguments. We can identify several stages in which arguments play an essential part:

- Ontology is defined as “a shared specification of a conceptualization” [21]. Although “shared” is an essential feature, it is often neglected. In DILIGENT experts exchange arguments while **building** the initial shared ontology in order to reach consensus;
- When users make comments and suggestions to the control board, based on their **local adaptations**, they are requested to provide the arguments supporting them;
- while the control board **analyzes** the changes introduced and requested by users, and balances the different possibilities, arguments are exchanged and balanced to decide how the shared ontology should change.

4 Use Case

As mentioned in the introduction the SEKT-project partners are currently building a common upper ontology (PROTON⁶). PROTON will be used in the applications developed in SEKT as background knowledge. Hence, PROTON will be used in case studies tackling knowledge management in a telecom company and a question answering system for legal education. Furthermore, PROTON serves as background knowledge for natural language processing and machine learning methods. Naturally, the applications have different requirements on the ontology. However, to maximize interchangeability of methods developed in SEKT, it was agreed to build a common ontology. From a DILIGENT process point of view, we are currently in the revision phase. Some of the partners have already used a previous version of PROTON and adapted it according to their needs⁷. Others have just recently joined the process. The discussion takes place on a mailing list. In the following we first summarize the experiences made so far and then introduce some additional features needed for the DILIGENT *Argumentation Ontology*.

⁵ This is actually one of the trends in modern software engineering methodologies (see Rational Unified Process).

⁶ <http://proton.semanticweb.org/>

⁷ The name was agreed only recently.

4.1 Traceability

As new partners get involved into the ontology building process, modelling decision are discussed more than once, since the modelling reasons of the existing version are not documented. Although, the actors in the current OE discussion present reasons for design modification, the number of e-mails makes it infeasible to retrieve them at a later stage. The ability to present the reasons and arguments for a modelling decision to the **new entrants** could speed up the design process. A similar problem arises, when the ontology is **revised** and the ontology engineers need to recall the reasons for the previous design. The users of the ontology can as well profit from a well documented ontology for a better understanding. Currently, they rely on the sparse explicit documentation, since documentation is a time consuming, often neglected task. A structured integration of the ongoing discussions can ease it.

Another issue is size. The current version of PROTON has more than two hundred concepts. Therefore, it is difficult to track which parts of the ontology are agreed and which are not. In an OE discussion actors often agree only implicitly with a certain modelling decision. For example a participant proposes B as subconcept of A without explicitly agreeing with A.

Besides the current experiences, first versions of PROTON had to be build from scratch. Although there are a number of ontologies available on the Semantic Web, this is not sufficient for an ontology to be reused. Only if the design rationales behind the model are available to others, can ontologies easily be included into applications.

4.2 Inconsistency Detection

During the argumentation process different participants exchange their opinions about the issue under discussion. A requirement on an efficient discussion is, that the arguments one participant brings forward are consistent with his previous arguments. A participant may change his opinion, but then he should discard earlier contradicting arguments. A model to conceptualize arguments should be able to detect at least some inconsistencies and point the arguer to the contradicting arguments 1.

Table 1. List of possible inconsistencies

Inconsistency	Description
Idea inconsistency	Arguer introduces Idea1 and Idea2 which are inconsistent
Argumentation inconsistency	Arguer argues first in favor and then against an issue. The lines of reasoning followed by the arguer lead to inconsistent ideas
Position inconsistency	Assuming Issue/Argument 1 and 2 are contradicting. An Actor produces a position inconsistency when he votes in favor of Issue/Argument 1 and then introduces Issue/Argument 2

4.3 Argument Selection

In the applications for the SEKT case studies the user of PROTON may wonder why some concepts, etc. were introduced in the ontology or he may ask why certain modelling decisions were made. However, even when we trace the underlying arguments, some of them may be very detailed and not understandable to normal users. Hence, if a user asks for the arguments underlying the ontology modelling decisions it would be beneficial to provide an answer which best fits the users needs. In this case we can assume that the best answer to such a query would be one which convinces the requester most. The selection of the appropriate arguments is only possible if not only the argumentation but also the arguments are formalized. Then we can build on models as presented in [18] that show how formal argumentation trees can be pruned to best correspond to the users wishes. On the other hand in tangled discussion it is not always obvious which proposal receives the strongest support. [17] presents a formal model to establish the winner of a discussion.

5 Requirements

We have identified several requirements for our *Argumentation Ontology* from the the SEKT PROTON case study and others where we have been involved such as IEEE SUO. Before we describe the ontology in the next section we now develop its requirements for it.

1. **Use common vocabulary** Research in argumentation and its visualization has a long history and is a mature field (*cf.* 2). To enhance acceptability for the ontology usage of the established vocabulary is essential.
2. **Focus on relevant arguments** As observed in [2] the restriction of available argument types can focus and speed up OE discussions. Hence the ontology should not model all possible kinds of arguments of a discussion, but focus on the relevant ones. This view is supported by [22] who have developed an ontology for a different domain but for a similar purpose and found that a smaller ontology enhances usability.
3. **Ontology focus** Following the results of [13], that IBIS should be enhanced with domain specific knowledge, the developed ontology should be particularly well suited for ontology design.
4. **Adaptivity** The *Argumentation Ontology* should allow for capturing the structure of argumentation. Hence, the design must take into account that e.g. humans discuss on a free text basis while ontology learning algorithms use formal, structured and detailed reasons for different proposals.
5. **Support entire argumentation** The *Argumentation Ontology* should support the full argumentation cycle. This includes issue raising, conflict mediation, bargaining, clarification and agreement. Participants should be aware of which issues are currently under discussion, postponed, agreed and discarded.
6. **Conceptual as well as formalization level** People might agree on the need for a certain conceptual model but not on its actual implementation. The model should support argumentation on both conceptual and formal models.

7. **Modularization** Although the ontology should support the ontology engineering process we do not aim to support every part of it. As described in [7] the ontology engineering process involves the definition of requirements, owners and other meta attributes like Dublin core meta data. These should not be modelled here.
8. **Formalism independence** The *Argumentation Ontology* should be independent of the formalism used to model the final ontology. Each formalism allows different sets of modelling decisions and all can be subject to discussion. However, the formal model of the finally agreed ontology should be a result of the instantiation of the *Argumentation Ontology*.
9. **Process awareness** The *Argumentation Ontology* is embedded into the DILIGENT process presented in section 3. Essential properties of this process are its collaborative aspects, its distributiveness and the asynchronous way participants can provide arguments.
10. **Argumentation formalization** Although we do not currently plan to provide the arguments themselves in a formal way, the *Argumentation Ontology* should allow us to do so. As our last use case has illustrated ontology engineering and ontology usage could gain from such a formalization.

6 An Argumentation Ontology for DILIGENT Processes

The DILIGENT *Argumentation Ontology* is visualized in figure 2⁸. The main concepts in our ontology are **issues**, **ideas** and **arguments**, which are represented as classes. These are in line with the terminology proposed by the IBIS methodology (req. 1). Issues introduce new topics in the discussion from a conceptual point of view. They are used to discuss what should be in the conceptual model of the ontology without taking into account how these items should actually be formalized and implemented in the ontology (req. 8). Ideas refer to how these concepts should be formally represented in the ontology, for instance as a class, an instance, etc. They relate to concrete ontology change operations⁹. Ideas are related to issues in the sense that they *respond to* them. Ideas refer to how issues should actually be implemented in the ontology. In this way discussions can take place in both the conceptual level and the formalization level (req. 6). Arguments are *arguments on* either one particular idea or one particular issue. Typically, our domain experts will start by proposing new issues to be introduced in the ontology. Arguments will be exchanged over them. Then, they discuss how these issues should be formalized through concrete ideas. Domain experts can also provide **elaborations**. These are issues that refine an issue under discussion, *elaborates on*.

Since concepts to be represented in an ontology should be consensual, this requires some consensus building discussions. In DILIGENT processes, concepts are only added to the ontology if they can be agreed upon, that is after some arguments have been exchanged, positions by different actors have been issued on them and some decisions

⁸ The corresponding OWL ontology will be available online in case of acceptance.

⁹ For example [23] presents a formal model for ontology change operations.

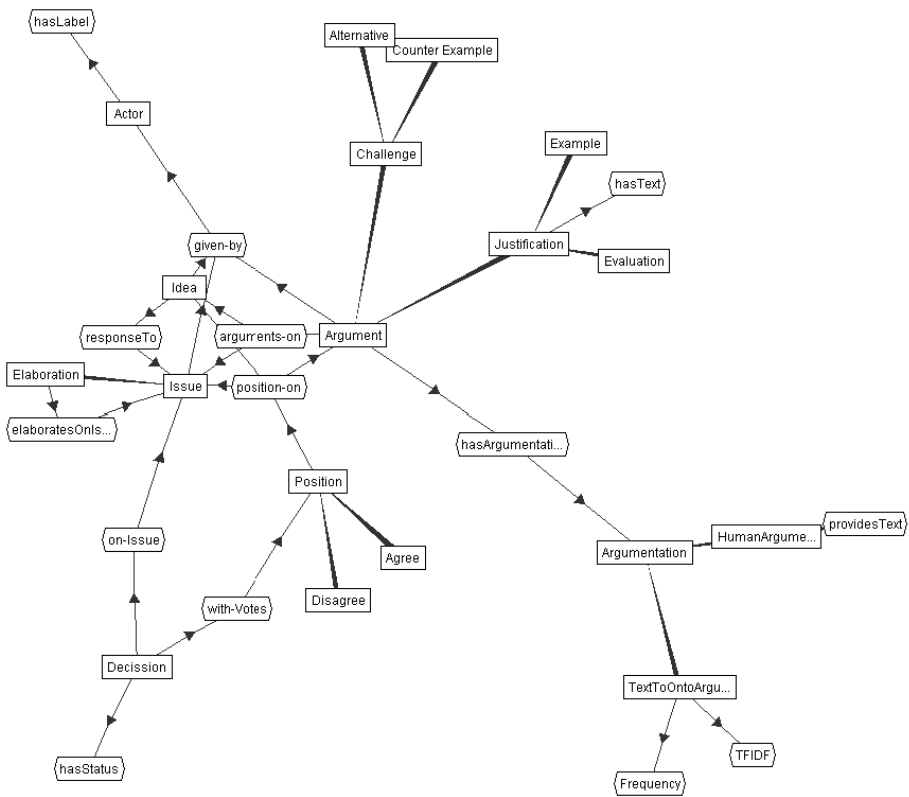


Fig. 2. The major concepts of the argumentation ontology and their relations

have been made. Arguments for (pro) an idea or issue are called justifications. Arguments against (con) an idea or issue are called challenges. In what regards arguments in favor, particularly useful OE processes, we identified examples and evaluation&justification. Two classes in challenges are also particularly used in OE discussions: counter examples and alternative&contrast. These arguments focus the IBIS argumentation methodology for Ontology Engineering (req 3).

Those involved in discussions can state positions. They clarify the *position on* one issue, one idea, or an argument under discussion. Either one agrees or disagrees. Once enough arguments have been provided and positions have been stated on them decisions can be made. In general, positions lead to decisions. Decisions are taken on issues. A decision has a status that can vary from under-discussion, postponed, discarded and agreed (req 5). A decision records not only the *issue on* which it was taken, but also both the positions issued when final *with-votes* (several positions) were cast and the *line of reasoning* (a sequence of arguments) underlying the decision on that issue. A decision can also state the *idea on-idea* underlying its issue. This allows one to focus on the relevant arguments (req 2).

Arguments are *given by* actors (req 9). We can have different kinds of **Actors**: either **Humans** or **Machines**. Different kinds of actors provide different argumentations (req. 4). In what regards **argumentation** humans (**HumanArgumentation**) tend to argue by providing strings of text stating (*provides text*) their reasons while machines tend to use other kinds of argumentation measures like **Frequency** and **TFIDF** [6]. For each algorithm used, new subclasses of argumentation need to be introduced to model the different kinds of measures.

7 Example: An Argumentation Ontology for DILIGENT Processes

The following discussion transcript was a part of an experiment performed at our institute (*cf.* [2], section 2.3). The participants were asked to build an ontology for modelling the research interests of our group. The experiment lasted for 90 min. and involved eleven actors. The participants provided their arguments in free text without formal restrictions. Hence, in the following example we model the discussion *ex post*. Moreover, we do not aim to model the entire discussion, but pick out an excerpt to exemplify our model.

...

cs: We have done quite a bit of research in distributed knowledge management (DKM) lately. So I suggest DKM as a topic plus a subtopic “peer to peer” (P2P)

The actor suggests on the one hand to introduce “DKM” and “P2P” in the ontology (Issues), and proposes on the other hand to model them as “topics” (Ideas).

Formalization

```
Individual(issue1 type(Issue) value(states “I suggest DKM”))
Individual(issue1 type(Issue) value(given-by actorCS))
Individual(justi1 type(Justification) value(hasArgumentation argumentation1))
Individual(justi1 type(Justification) value(arguments-on issue1))
Individual(argumentation1 type(HumanArgumentation) value(providesText “We have
... lately”))
Individual(idea1 type(Idea) value(respondsTo issue1))
Individual(idea1 type(Idea) value(ontoChange add(DKM:Topic)))
Individual(elaboration2 type(Elaboration) value(states “P2P subtopic DKM”))
Individual(idea2 type(Idea) value(respondsTo elaboration2))
Individual(idea2 type(Idea) value(ontoChange add(DKM supertopic P2P)))
```

ah: I suggest knowledge management (KM) as super concept of DKM because every DKM is a kind of KM

The second actor agrees implicitly with the suggestion to introduce “DKM” in the ontology. In contrast to the first one he proposes to model it as a “concept”.

Formalization

...

```
Individual(idea3 type(Idea) value(ontoChange add(KM:Concept)))
```

...

jt: Well I am now wondering whether P2P is DKM, because File exchange is not always KM is it?

A third actor agrees also implicitly, that “P2P” and “DKM” are important for the domain, but challenges that they should be modelled in the proposed way.

Formalization

```
Individual(counter1 type(CounterExample) value(hasArgumentation argumentation3))
Individual(counter1 type(CounterExample) value(arguments-on elaboration2))
Individual(argumentation2 type(HumanArgumentation) value(providesText “File exchange ... KM”))
```

ph: I suggest Distributed Comp. (DC) with P2P and Grid as subtopics; DKM as subtopic of DC and KM

The fourth actor presents a new issues which could resolve the conflict.

Formalization

```
...
Individual(issue2 type(Issue) value(states “I suggest DC”))
Individual(elaboration3 type(Elaboration) value(... ..))
...
```

do: PRO *ph* : because his approach separates KM and distributiveness

The actor “do” agrees with the suggestion and provides additional reasons for the design. Implicitly he also agrees that “KM” should be part of the ontology.

Formalization

```
Individual(position1 type(Agree) value(position-on elaboration3))
...
```

cs: I’d like to agree to *ph* and *do* suggestion.

...

The first actor agrees with the new solution and discards his original proposal.

This example demonstrates that OE discussion can be modelled with the DILIGENT *Argumentation Ontology*. The applicability of the ontology will depend on the available tool support. We do not intent to automatically annotate a free discussion. We rather envision a template based approach. Currently we use a WIKI to support the argumentation process. However, integration with reasoners and inclusion into existing OE environments is desirable, but remains to be done.

8 Conclusion

It is now widely agreed that ontologies are a core enabler for the Semantic Web vision. The development of ontologies in centralized settings is well studied and there are established methodologies. However, current experiences from projects suggest, that

ontology engineering should be subject to continuous improvement rather than a one-time effort and that ontologies promise the most benefits in decentralized rather than centralized systems.

In such settings, arguments play a major role in the process of consensus building between the involved participants. Based on the current state of the art in Ontology Engineering, Argumentation visualization and Argumentation structures, we propose an integrated formal argumentation model to be used in OE discussions, in particular in DILIGENT OE processes. This ontology supports the process in several ways. In discussions, it focuses the participants and helps to structure their arguments. In the usage and analysis phases, the exchanged arguments can be consulted to better understand the current version of the model. Moreover, since it is formal it allows for inconsistency detection in argumentations. Since the ontology covers all aspects of the discussion activity, namely issue raising, formalization of the issues and decision making, the participants are always informed about the current status of the discussion and the ontology they are building.

We demonstrate the applicability of our model by formalizing an OE discussion drawn from an experiment in our institute. The DILIGENT Argumentation Ontology will also be the basis for ontology discussions in the SEKT project. To support the discussion with appropriate tools we are currently investigating a combination of WIKI like argumentation support with ontology formalization in the KAON tool suit¹⁰. Argument selection based on formal arguments remains future work. In the further future we imagine that ontology learning methods can profit from the formalized discussion and learn from human ontology design decisions.

Therefore, the main contribution of this paper is the first formal argumentation model for Ontology Engineering, in particular for DILIGENT OE processes. This model is an adaptation of the IBIS argumentation model specifically for Ontology Engineering. It clearly distinguishes between phases: discussions should be about the conceptual model, about Issues, and about the formal model, about Ideas. Moreover, from our previous experiences in DILIGENT OE processes this model clearly states the arguments that have been identified as speeding and easing the consensus building process needed to build shared ontologies. Finally, this is the first model that attempts to integrate arguments from (semi-)automatic ontology building based on learning.

Acknowledgements. Research reported in this paper has been financed by EU in the the IST project SEKT (IST-2003-506826).

References

1. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering. Advanced Information and Knowledge Processing. Springer (2003)
2. Pinto, H.S., Tempich, C., Staab, S., Sure, Y.: Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In de Mántaras, R.L., Saitta, L., eds.: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), August 22nd - 27th, Valencia, Spain, IOS Press (2004) 393–397

¹⁰ <http://kaon2.semanticweb.org>

3. Pinto, H.S., Staab, S., Sure, Y., Tempich, C.: *OntoEdit Empowering SWAP: a Case Study in Supporting DIstributed, Loosely-Controlled and evolInG Engineering of oN-Tologies (DILIGENT)*. In Bussler, C., et al., eds.: *Proceedings of the 1st ESWS 2004*. (2004)
4. Buckingham Shum, S., Motta, E., Domingue, J.: *Augmenting design deliberation with compendium: The case of collaborative ontology design*. In: *HypA-CoM 2002: Facilitating Hypertext-Augmented Collaborative Modeling*. ACM Hypertext'02 Workshop, University Maryland, MD (2002) Retrieved November 24, 2004 from <http://kmi.open.ac.uk/projects/compendium/SBS-HT02-Compendium.html>.
5. Ramesh, B., Dhar, V.: *Supporting systems development by capturing deliberations during requirements engineering*. *IEEE Trans. Softw. Eng.* **18** (1992) 498–510
6. Maedche, A., Staab, S.: *Ontology learning for the semantic web*. *IEEE Intelligent Systems* **16** (2001)
7. Sure, Y., Studer, R.: *On-To-Knowledge methodology*. In Davies, J., et al., eds.: *On-To-Knowledge: Semantic Web enabled Knowledge Management*. J. Wiley and Sons (2002)
8. Holsapple, C.W., Joshi, K.D.: *A collaborative approach to ontology design*. *Commun. ACM* **45** (2002) 42–47
9. Kotis, K., Vouros, G.A., Alonso, J.P.: *HCOME: tool-supported methodology for collaboratively devising living ontologies*. In: *SWDB'04: Second International Workshop on Semantic Web and Databases 29-30 August 2004 Co-located with VLDB*, Springer-Verlag (2004)
10. Kunz, W., Rittel, H.W.J.: *Issues as elements of information systems*. Working Paper 131, Institute of Urban and Regional Development, University of California (1970)
11. Aschoff, F.R., Schmalhofer, F., van Elst, L.: *Knowledge mediation: A procedure for the cooperative construction of domain ontologies*. In Abecker, A., van Elst, L., Dignum, V., eds.: *Proceedings of Workshop on Agent-Mediated Knowledge Management at the 16th European Conference on Artificial Intelligence (ECAI'2004)*, Valencia, Spain (2004) 20–28
12. Conklin, J., Begeman, M.L.: *gibis: a hypertext tool for exploratory policy discussion*. In: *Proc. of the 1988 ACM conference on Computer-supported cooperative work*. (1988)
13. Potts, C., Bruns, G.: *Recording the reasons for design decisions*. In: *Proceedings of the 10th international conference on Software engineering*, IEEE Computer Society Press (1988)
14. Gotel, O., Finkelstein, A.: *Extended requirements traceability: Results of an industrial case study*. In: *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, IEEE Computer Society (1997) 169
15. Selvin, A., Buckingham Shum, S., Sierhuis, M., Conklin, J., Zimmermann, B., Palus, C., Drath, W., Horth, D., Domingue, J., Motta, E., Li, G.: *Compendium: Making meetings into knowledge events*. In: *Knowledge Technologies*, Austin, TX (2001)
16. Mann, W.C., Thompson, S.A.: *Rhetorical structure theory: A theory of text organization*. In Polanyi, L., ed.: *The Structure of Discourse*. Ablex Publishing Corp., Norwood, N.J. (1987)
17. Gordon, T.F., Karacapilidis, N.: *The zeno argumentation framework*. In: *Proceedings of the sixth international conference on Artificial intelligence and law*, ACM Press (1997) 10–18
18. Hunter, A.: *Towards higher impact argumentation*. In McGuinness, D.L., Ferguson, G., eds.: *AAAI2004*, AAAI Press / The MIT Press (2004) 275–280
19. Pinto, H.S., Martins, J.: *A Methodology for Ontology Integration*. In: *Proc. of the First Int. Conf. on Knowledge Capture (K-CAP2001)*, New York, ACM Press (2001) 131–138

20. Uschold, M., King, M.: Towards a methodology for building ontologies. In: Proc. of IJCAI95 WS, Montreal, Canada (1995)
21. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N., Poli, R., eds.: Formal Ontol. in Conc. Analysis and Knowl. Rep., Kluwer Acad. Pub. (1993)
22. Buckingham Shum, S., Gangmin Li, V.U., Domingue, J., Motta, E.: Visualizing internet-worked argumentation. In Kirschner, P.A., et al., eds.: Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making. Springer (2003) 185–204
23. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW, Madrid, Spain (2002)

Towards an Ontology-Based Distributed Architecture for Paid Content^{*}

Wernher Behrendt¹, Aldo Gangemi², Wolfgang Maass³,
and Rupert Westenthaler¹

¹ Salzburg Research GmbH, Jakob-Haringer Strasse 5/II,
5020 Salzburg, Austria
{Wernher.Behrendt, Rupert.Westenthaler}@salzburgresearch.at

² Institute of Cognitive Sciences and Technology (CNR),
Via Nomentana 56, 00161, Roma, Italy
a.gangemi@istc.cnr.it
<http://www.mcm.unisg.ch>

³ =mcmstitute, University of St. Gallen, Blumenbergplatz 9,
9000 St. Gallen, Switzerland
wolfgang.maass@unisg.ch
<http://www.mcm.unisg.ch>

Abstract. Business models on the basis of digital content require sophisticated descriptions of that content, as well as service-oriented carrier architectures that allow to negotiate and enforce contract and license schemes in heterogeneous digital application environments. We describe Knowledge Content Objects (KCO), that provide expressive semantic descriptions of digital content, based on an ontology of Information Objects, built under the DOLCE, DnS and Plan Ontologies (DDPO). In particular, we discuss how this structure supports business requirements within the context of paid content. Interactions between agents are embedded into digital infrastructures that are implemented on an advanced knowledge content carrier architecture (KCCA) that communicates via a dedicated protocol (KCTP). We show how this architecture allows to integrate existing digital repositories so that the content can be made available to a semantically rich digital environment.

1 Introduction

The WWW can be perceived as a huge information market where supply and demand meet. If content obtains high value to certain demand sides, it will generate market prices. This kind of content is generally termed "paid content" as a special form of information goods and is viewed as a digital product ([4, 16]). [12] define the term *information good* very broadly. Based on the definition of [16] anything one can send

^{*} This work is part-funded by the European Union (6th Framework Programme under the strategic objective IST-2002-2.3.1.7. - Semantic-based Knowledge and Content Systems) and the Swiss Federal Government.

and receive over the Internet has the potential to be a digital product. The term *paid content* is in this article used as the non-free sales and distribution of information-based content products.

A hurdle for effective markets for paid content is the non-existence of appropriate transaction mechanisms that support search, usage and control of paid content so that suppliers can implement sustainable business models and users can efficiently obtain information about content properties and can efficiently use purchased content. Paid content needs to support two kinds of situations: *trading situations* and *usage situations* [11]. During a trading situation a consumer and a vendor negotiate the terms under which a consumer gains rights that can be executed on a particular content. To gain advantages of search products in a trading situation, consumers must be supported by product information on the (1) *utility of content* in respect to intended application situations, (2) *resource restrictions* and (3) *application requirements* that are given by the intended application environment. Resource restrictions encompass organisational, temporal, spatial, presentational and financial dimensions that are relevant during the trading act. The latter category describes in which application environments a particular instance of a paid content is intended to be used in principle. This encompasses its (1) situational requirements, i.e. when it can be used by whom, (2) business requirements, i.e. which contractual obligations, pricing and license schemes apply, (3) constraints on the technical environment in which it can be used, (4) how it presents itself and (5) which requirements are to be met while using it.

Any mismatch of these three categories decreases the utility of a particular content and influences the consumer's buying decisions negatively.

Throughout this article we will explore the potential of semantic annotations of paid content that provide on one hand an opportunity for interoperable markets for paid content and on the other hand a means for product self-descriptions which has strong influence on consumer buying decisions. First, we will briefly discuss application situations from which we will derive requirements for the intended semantically annotated content structure and the underlying technical content carrier architecture. In subsequent sections, we will introduce a semantically enriched content carrier structure (KCO) that is used as a flexible and expressive container for digital content. KCOs are exchanged over a transmission infrastructure that is based on a generic content carrier architecture (KCCA), which enables the interoperation of heterogeneous content repositories. Finally, we will summarize the status of our current work and give an outlook on our future work.

2 Semantic Modeling of Content Objects

An analysis of several hundred existing paid content business models ([13]) resulted in a classification of five central elements to which digital content has to respond during different phases of its life cycle.

1. *Content descriptions*: provides the propositional content that is announced by a digital content on an abstract level.

2. *Community descriptions*: information about the organisational structure (roles, rights and obligations) by which a content product can be used and information that influences trust such as certificates and brand name information.
3. *Business descriptions*: describes the business and legal requirements during information and negotiation phases. After a purchase, contracts will be enforced according to mutually agreed rights and obligations.
4. *Presentation descriptions*: describes the presentation modes to which the information of a content product can be adapted to by rendering and other application-specific means.
5. *Trust and Security descriptions*: content must be associated with some measure of trust for the end user, and for the content provider, there must be some security features which guarantee that the content will not be illegally copied or otherwise misappropriated.

As these results show, digital content needs to be semantically annotated so that it can respond to these five elements. We will now incrementally introduce the concept of a Knowledge Content Object (KCO) that is intended to provide this structure on a computational level because it is intended to be implemented in digital infrastructures. From the foundational ontological viewpoint given by DOLCE and its extensions (a modular library called DDPO), a KCO is to be distinguished from the abstract concept of an information object that carries meaning on cognitive and abstract level, independent on any technical realisation. Because we want to leverage the advantages of foundational ontologies for the exchange and translation of meanings on technical but also non-technical level, KCOs are embedded into DDPO.

2.1 An Ontology of Information Objects

We lay down here a semantic foundation for KCOs, based on an ontology of information objects.

Our ontology for information objects is an extension of *DOLCE* (Descriptive Ontology for Linguistic and Cognitive Engineering), *DnS* (Ontology of Descriptions and Situations), and *Plans* Ontologies. Parts of the reused ontologies have been originally developed within the WonderWeb [8] and Metokis EU [6] projects. We will refer here to this extended ontology as DDPO [6].

The main distinctions in the reused ontologies, which are imported here, include:

- the topmost class is called *particular* (any entity)
- *objects* (e.g. a *dog*) and *events* (e.g. a *barking*) belong to disjoint classes
- *physical* (e.g. a *brick*) and *social* (e.g. a *contract*) objects belong to disjoint classes
- attributes of particulars (e.g. a *color*, or a *spatio-temporal location*) are represented as *regions* within *quality spaces*, with a possible associated metrics
- social objects include *descriptions* (the public, communicable counterpart of agents' conceptualizations, including also *plans*), which can define *concepts* (the *customer role*), encode (or be expressed by) *information objects* (a *sentence*, or a *music chart*), provide unification criteria for *collections* (a *group of people*), etc.
- concepts can be either *roles* played by objects, *tasks* executed during actions (e.g. a *door opening task*), etc.

- concepts from descriptions provide constraints for other particulars: if a configuration of particulars satisfy those constraints, a *situation* emerges that satisfies the concepts' description (typical applications of constraint unification include *regulations, plans, social relationships*, etc.).

The previous distinctions are supported by a large axiomatization that cannot be reported here.¹ We'd rather concentrate on their application as a foundation for KCO implementation and deployment.

For example, a usage context of a content object may require to talk about the digital reproduction of a painting that is owned by an institution, and such institution is willing to commercialize the reproduction at certain conditions that include differentiation for users, pricing, regulations to be followed, inclusion of content metadata, explanations, interpretations, ways of rendering it, etc. This context is complex, and require a subtle understanding of the different entity types involved in it.

According to DDPO, a content (information) transferred in any modality is a kind of social object called *Information Object* (IO). Information objects are *spatio-temporal reifications* of pure (abstract) information as described e.g. in Shannon's communication theory, hence they are assumed to be in time, and realized (materialized) by some entity.

Information objects are the core notion of a *semiotic ontology design pattern*, which employs typical *semiotic relations*, as explained here. The complete IO ontology is quite complex, and is presented elsewhere [6][8].

We present the axiomatization of KCOs in OWL abstract syntax. We firstly present the definition of `DnS:information-object`, which encodes the basic axioms of an ontology of semiotics extending the basic DDPO ontology:

```
Class(DnS:information-object complete
  intersectionOf(
    DOLCE:social-object
    restriction(DnS:about allValuesFrom(DOLCE:particular))
    restriction(DnS:realized-by
someValuesFrom(DOLCE:information-realization))
    restriction(DnS:interpreted-by
allValuesFrom(Actions:agent))
    restriction(DnS:expresses allValuesFrom(DnS:description))
    restriction(DnS:ordered-by someValuesFrom(DnS:information-
encoding-system)))
```

The definition says that information objects:

- are necessarily *encoded* by some information encoding system
- must be *realized* by some particular
- can *express* a description, and, if that description is satisfied by a situation
- can be *about* that situation, or some entity in its setting
- can be *interpreted* by agents that can conceive the description expressed by said IOs.

¹ The full OWL axiomatization of DOLCE, DnS, DDPO, IO, etc. can be downloaded from: <http://dolce.semanticweb.org>.

For example, Dante's Divine Comedy is an IO, it is *ordered* by Middle Age Italian language (the information encoding system), is *realized* by e.g. a paper copy of the 1861 edition with Doré's illustrations, *expresses* a certain plot and its related meanings (literal or metaphorical), as *interpreted* by an agent with an average knowledge of MA Italian and literary studies, and it is *about* certain entities and facts.

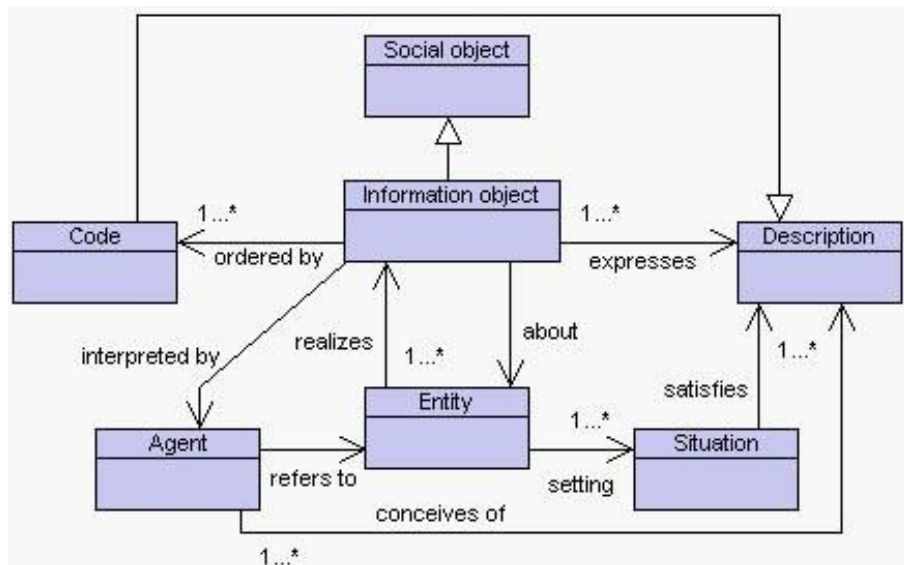


Fig. 1. The IO pattern: an information object is a social object, *ordered* by a code, and *realized* by some concrete entity. It *expresses* a description *conceived* by some agent, *about* some entity. Situations exist for the setting of *realization* (“communication”), as well as for *aboutness* (“reference”); agents *refer* to entities that IOs are about while interpreting them

The relations *realizes*, *expresses*, *about*, and *interprets* must be taken as *temporally indexed*, but such indexing cannot be expressed directly in the OWL property definition; for example, the definition of `DnS:realizes`:

```

ObjectProperty(DnS:realizes
  inverseOf(DnS:realized-by)
  domain(DOLCE:information-realization)
  range(DOLCE:information-object))

```

needs to be complemented by an OWL axiom stating that something that realizes an IO must be present at least in some time interval at which that IO is also present:²

² The axiom does not completely catch the semantics of a real ternary relation: *realizes*(*x*,*y*,*t*), where *t* is a time interval, but it is a useful approximation anyway. Further refinements could be made by using SWRL [7].

```

SubClassOf(
  restriction(DnS:realizes      someValuesFrom(DnS:information-
object))
  restriction(DOLCE:present-at someValuesFrom(intersectionOf(
  DOLCE:time-interval
  restriction(DOLCE:time-of-presence
someValuesFrom(DnS:information-object))))))

```

These *semiotic* relations constitute a typical *ontology design pattern*, so that any composition of relations can be built starting from any node in the pattern or in an application of the pattern.

The pattern has also some additional axioms, for example, the property *interprets* implies that an expressed description is *conceived* by the agent (i.e., when an agent interprets an IO, it conceives the description expressed by the IO; of course two agents can conceive different descriptions, then resulting in different interpretations).

Once introduced the concept of an information object, we will now describe informally, the intuition and resulting general structure of a Knowledge Content Object (KCO). This work is based on results from predecessor projects: in the CULTOS project (www.cultos.org) we introduced the notion of an enhanced (by explicitly stated domain knowledge) multimedia meta object (EMMO). The logic description of KCOs is based on these ideas, firstly formulated in [9] and initially formalized without ontological grounding by [10]. The business related aspects of KCOs are based on lessons learned in the INKASS project ([1]).

3 Semantic Modeling of Knowledge Content Objects (KCO)

The notion of knowledge content objects is based on business requirements (see section 1), and builds upon previous approaches to multimedia and hypermedia document models. Related work includes [3, 14, 17]. The strength of KCOs lies in the combination of business- and domain-specific semantics that are tied into DDPO. We have come to distinguish four abstraction levels of a KCO:

1. *Generic KCO (physical) schema*: a physical data structure (subClassOf “computational object” in DDPO) that realizes an abstract data structure (subClassOf “information object” in DDPO).
2. *Tradeable domain KCO (physical) schema*: a a physical data structure that specializes of a generic KCO schema, including an ontology of a domain (e.g. a CD ontology), and an ontology for the related business semantics. By default, we assume that this abstraction layer also features the description of a particular business semantics (see next section). There can be several levels of domain schema specialization.
3. *Instantiated KCO Prototype (Master Copy)*: a physical data structure implementing the same facets (and business semantics) from a tradable domain KCO schema, but that also contains, at a certain point in time, a particular digital content including all semantic annotations required by the corresponding tradable domain KCO schema.
4. *KCO Instance (Copy)*: the clone of an instantiated KCO prototype. It is distinguished here for business reasons; for example, when a user is granted access to a content object, and depending on the contract semantics, this copy could

change over time, through alterations made by the owner. For example, somebody may buy a backing track for a pop song, in order to add her own voice to the recording.

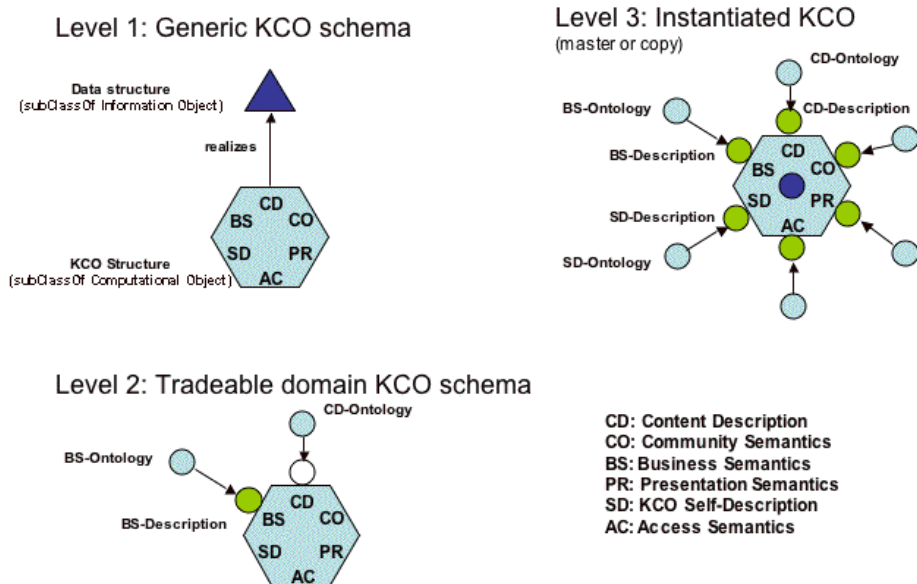


Fig. 2. Levels of KCO abstraction and KCO facets. All levels include physical data structures

Derived from the analysis of business models and paid content products, we have developed a KCO structure consisting of eight facets. Several of these facets are subdivided into further KCO sub-facets. At the "atomic" level, it is intended that each of the leaf elements is associated with well-defined operational semantics, in order to enable organisations to quickly deploy KCOs as part of their information infrastructure.

While KCOs are also rooted in semantic web technology - using an extension of DDPO for their definition - our application interest is more strongly geared towards the following question: "what information and knowledge exchange processes can be actively (i.e. operationally) supported by semantic web technology". In particular, we are interested in how traditional digital content can be enhanced in order to qualify as "knowledge" content.

The first facet is what we call the *content description*. The KCO carries a list of media references which are intended to point to real media files. So the collection of these referenced media files is actually, the full intended content of the KCO. In order to make this content accessible for machines, we provide a simple referential mechanism to associate arbitrary logic descriptions to the media files. For this, the propositional content facet is linked to a domain ontology which represents the universe of discourse for all content descriptions of this KCO. When an instantiation

of a KCO is created, then arbitrary selections of the multimedia assets can be associated with statements that are valid according to the ontology. The semantic annotation is very flexible as it can relate to segments of a media asset (e.g. a scene in a video or a region in an image) or even to a relationship that holds between some media assets. (For example, it holds for the novel "Don Quixote", that it is a parody of the chivalrous romantic epic of "Orlando Furioso". This relationship holds between the "prototypes" of the two novels, and between all derivatives of the two. So by stating it once, we assert this knowledge to all instantiations (i.e. copies) of the novels. The details of creating media semantic networks have been described elsewhere [e.g. 14].

The second facet is the specification of time-based spatial *presentation* of complex content. Given some media tokens, we specify on one or more temporal "tracks" which describe *when* the associated media data will be rendered, and *where* they will be rendered (in terms of spatial arrangements). In an operational environment, this component may use elements of the SMIL multimedia synchronisation language for its implementation.

Table 1. KCO facets

Elements	Facets	Sub-Facets
Content Description (CD)	(1) Propositional Content	Media references
		Logic descriptions
Presentation Description (PR)	(2) Spatio-temporal rendition	
	(3) Interaction-based rendition	
	(4) Multimedia characterisation	Media properties
		Content classification
Community Description (CO)	(5) Usage context	User task
		User community
		Usage history
Business Description (BS)	(6) Business and legal semantics	Negotiation protocol
		Pricing scheme
		Contract
Trust & Security (TS)	(7) Semantics and pragmatics of confidence in virtual goods	
Self-description (SD)	(8) The description of the KCO's semantic structure (schema)	

The third facet deals with *interaction* and *dialogue*. Here, the semantic annotation specifies whether the presentation is entirely pre-programmed, whether it is entirely open (e.g. web based navigation) or whether it follows some dialogue pattern where humans and system take conversational turns in order to navigate the knowledge/information structure. This description defines one or more discourse structures that can be associated with the content for its rendering.

The fourth facet contains *interfaces to existing metadata standards*, notably those in the cataloguing and media management areas. Its purpose is to enable the migration

of media and their associated meta data into the KCO structure. We envisage this to be the place where meta data harmonisation can be done.

The fifth facet, *community description*, describes the context in which a content can be used. This covers three sub-facets: *user tasks*, which are formally described by reference to an ontology of plans and tasks; *user community*, which describes the situations with corresponding roles (rights and obligations) that users would take in order to manipulate or consume the content; and *usage history*, keeping traces of previous use in order to support workflow systems as well as collaborative filtering systems. The latter can be achieved by keeping track of user data when the KCO is being "touched" by that user. Depending on legal and contractual aspects of this facet, the filtering may be more or less anonymous.

The sixth facet, *business description*, contains a specification of the business semantics associated with the KCO. This comprises the sub-facet negotiation protocol which describes the business scripts by which a contract is being negotiated. A negotiation protocol is described as a DDPO *plan* that can be represented and processed in OWL-DL format [6]. The pricing scheme is used for restricting the price policies that can be applied during the negotiation. It is grounded in DDPO as the *regulation* concept. In the simple case of a fixed-price scheme, the negotiation is reduced to a simple over-the-counter (OTC) purchase. The pricing scheme is required for price differentiation strategies that are defined by the seller on the basis of a differentiating factor such as age, quantity discount or date of content origin (see [13, 15]). The resulting contract is also a plan in the DDPO ontology that describes the situation in which agentive roles can be taken by agents and act by using described tasks. We distinguish between pre-existing content and prospective content. Pre-existing content is already available at contracting time while prospective content is produced during execution time of the contract.

The seventh facet, *trust and security*, is currently deemed out of the scope of our project although we acknowledge the need for inclusion of the issue, in the overall framework of METOKIS.

The framework is rounded off with the eighth facet, *self-description*, which exports the basic structure and formal semantics of the KCO to external systems that may want to make use of the KCO structure and its supported semantics.

In the following section, we will translate the KCO model into an OWL ontology, which specializes the IO branching of DDPO foundational ontology.

3.1 KCO Formalization

The IO design pattern can be used and extended in order to characterize Content Objects (CO) and Knowledge Content Objects (KCO). As introduced, IOs can be *realized* by any sort of entities. The realization relation provides us the expressivity to talk about KCOs: a KCO is described here as a *physical data structure*, a subClassOf *computational object*, which is a subClassOf *information realization* (that can be any entity, ultimately relying on some physical object, substance, event, etc.). A physical data structure realizes a (abstract) *data structure*, which is a kind of IO.

KCOs are distinguished from digitalized COs (*content objects*), which are another kind of computational object. A KCO provides an implemented data structure (a *frame*) to COs. The relation between a KCO and a CO is provided by the property: `KCO:realizes-frame-for`:

```
ObjectProperty(KCO:realizes-frame-for
  domain(KCO:physical-data-structure)
  range(KCO:content-object))
```

Such a property is complemented by an appropriate axiom that states that the KCO realizes a data structure that is a *frame* for a content object.

These assumptions allow us to give a foundation to the operationalized KCO model in the KCCA architecture: each *facet* in that model is formalized here as an OWL property, linked to the DDPO ontology.

Some basic distinctions are made firstly to catch the different states of the implemented data structures: in the KCO ontology, the implementation of the most generic data structure (the facets without any values) is called `KCO:generic-KCO`. The class of domain-oriented data structures (the facets with specified “types” for the value of the facets), and a given business semantics, is called `KCO:domain-KCO`. The class of KCOs themselves (the implemented physical data structure with at least one value filling a facet type) is called: `KCO:KCO`. Two properties of KCOs are conceived: `KCO:master-of`, used to characterizes the first implementations of each KCO (this is specially relevant with reference to the masters of content objects that are framed by the KCO data structure), and `KCO:copy-of`, which is used to characterize the copies of the master. Different modification rights, contracts, and pricing schemes apply to the masters.

Once introduced the intended meaning of KCO, and its reference to abstract data structures and content objects, we lay down our characterization of facets in terms of OWL properties.

Firstly, we summarize the properties as from the OWL definition of KCO:

```
Class(KCO:KCO complete
  KCO:physical-data-structure
  restriction(DnS:realizes someValuesFrom(intersectionOf(
    KCO:data-structure
    restriction(KCO:instantiates someValuesFrom(intersectionOf(
      KCO:data-structure
      restriction(DnS:realized-by someValuesFrom(KCO:domain-
KCO)))))))
  restriction(KCO:realizes-frame-for
someValuesFrom(KCO:content-object))
  restriction(KCO:content someValuesFrom(DOLCE:particular))
  restriction(KCO:time-based-rendition
someValuesFrom(KCO:script))
  restriction(KCO:interaction-based-rendition
someValuesFrom(KCO:script))
  restriction(KCO:usage-context someValuesFrom(DnS:plan))
  restriction(KCO:user-task someValuesFrom(DnS:task))
  restriction(KCO:content-user someValuesFrom(unionOf(
    DnS:organization Collectives:collective DnS:agent-driven-
role)))
  restriction(KCO:contract-semantics
```

```

    someValuesFrom(CoreLegal:contract))
    restriction(KCO:negotiation-semantic
someValuesFrom(negotiation-protocol))
    restriction(KCO:pricing-semantic
someValuesFrom(KCO:pricing-scheme))
    restriction(KCO:trust-value           someValuesFrom(KCO:trust-
region))
    restriction(KCO:mappable-to
allValuesFrom(KCO:ForeignClass)))

```

We mean that:

- KCO is a subclass of the class *physical data structure*. All the facets of a KCO data structure are modelled in OWL as “restrictions”:
- a KCO realizes a data structure that *instantiates* the data structure provided by a domain KCO (this represents the “self-description” facet)
- a KCO realizes a frame for one or more content objects (even past or future)
- a KCO propositionally represents (in this case, through OWL) the content of the content objects which it realizes a frame for
- a KCO provides scripts for the time-based, and interaction-based renditions of the content objects
- a KCO provides (eventually propositional) representations of the usage context (the *plan*) in which the content objects are supposed to be involved. Plans are axiomatized in the *plan ontology*, another extension of DOLCE and DnS [1][7]
- a KCO has at least one user task. Tasks are also defined in the plan ontology
- a KCO has at least one content user, that can be either organizations, roles played by agents (e.g. *author*), or collectives of any kind
- a KCO implements at least one (eventually propositional) contract semantics for the content objects: such semantics is representable within a *contract*, which is a kind of DnS:description
- a KCO implements at least one (eventually propositional) negotiation semantics for the content objects: such semantics is representable within a *negotiation protocol*, which is a kind of DnS:description
- a KCO implements at least one (eventually propositional) pricing semantics for the content objects: such semantics is representable within a *pricing scheme*, which is a kind of DnS:description
- a KCO provides a trust value for the content objects, here represented with reference to a *trust region* (an attribute), but in principle, it is possible to provide explicit (propositional) descriptions of *trustworthiness*, on which basis the trust regions can be parametrized
- a KCO can be mappable to one or more instances represented according to foreign classification schemes or ontologies. For example, given the CIDOC-CRM classification scheme [8], a KCO framing a digital edition of Dante’s Comedy can be mapped to an individual that has `rdf:type:CIDOC:E73.Information_Object`. Also parts of the content object, or its interpretations, references, etc. can be mapped using the same style.

All the properties that have been introduced have complementary axioms that allow to formally explicate their intended meaning on the basis of DDPO, and the IO extension. For example, the property `KCO:content` has the following complementary

axioms, which state that anything the information of a content object is *about*, or *expressed by* it, is a content for the KCO:

```
SubClassOf(
  restriction(KCO:realizes-frame-for someValuesFrom(
    restriction(DnS:realizes someValuesFrom(
      restriction(DnS:about someValuesFrom(DOLCE:particular))))))
  restriction(KCO:content someValuesFrom(DOLCE:particular)))

SubClassOf(
  restriction(KCO:realizes-frame-for someValuesFrom(
    restriction(DnS:realizes someValuesFrom(
      restriction(DnS:expresses
someValuesFrom(DOLCE:particular))))))
  restriction(KCO:content someValuesFrom(DOLCE:particular)))
```

Another example of axiomatic complementation of properties is given by the property *user task*, which is formalized as *equivalent* to having as usage context a *plan that defines (that) task*:

```
EquivalentClasses(
  restriction(KCO:user-task someValuesFrom(DnS:task))
  restriction(KCO:usage-context someValuesFrom(intersectionOf(
    DnS:Plan
    restriction(DnS:defines someValuesFrom(DnS:task))))))
```

The formalization detail of KCOs is justified by the intricate relationships holding between content, information, users, tasks, contexts, regulations, business requirements, etc. Being precise about these relationships helps the implementation of KCOs, the eventual interoperability with other knowledge management and content metadata systems, as well as paving the way towards ontology-driven management of content objects.

4 Architecture for a Distributed Content Infrastructure

In order to make use of the semantic richness that can be expressed with KCOs we need an infrastructure whose components support the functionality afforded by the KCO. The Knowledge Content Carrier Architecture (KCCA) does this in the shape of services which are logically clustered in the KCCA's components. Assuming a three-layer conceptual architecture with presentation / interaction, business logic and back-end data storage the KCCA specifies the middleware of the business layer. This gives rise to the following structural core components:

- KCO Service API** - offering the functions described by the facets in table 1
- KCCA Registry and Manager** - managing a federation of KCO-aware nodes
- KCTP Service** - a protocol to exchange service requests across KCCA nodes
- KCCA Profiles** - Services for the wrapping and integration of data sources

One of the assumptions of our work is that eventually, most information systems will make use of two further components: firstly, reasoning services based on ontologies and secondly, a task execution environment that will support the definition and execution of flexible workflows. KCOs are designed to support such an

architecture through their content description (this is where reasoning services can access the KCO) and through their community description (describing the tasks for which this KCO is useful and the roles of actors that would do the tasks). We envisage future publishing environments to use an integrated framework consisting of the components described. This will leave the application builder to focus on application and domain specific adaptations, and on the tailoring of the presentation /interaction layer to the needs of the customer.

The following architectural overview shows the full picture combining KCCA components, reasoning and task execution environment, as well as domain specific adaptations and the application layer.

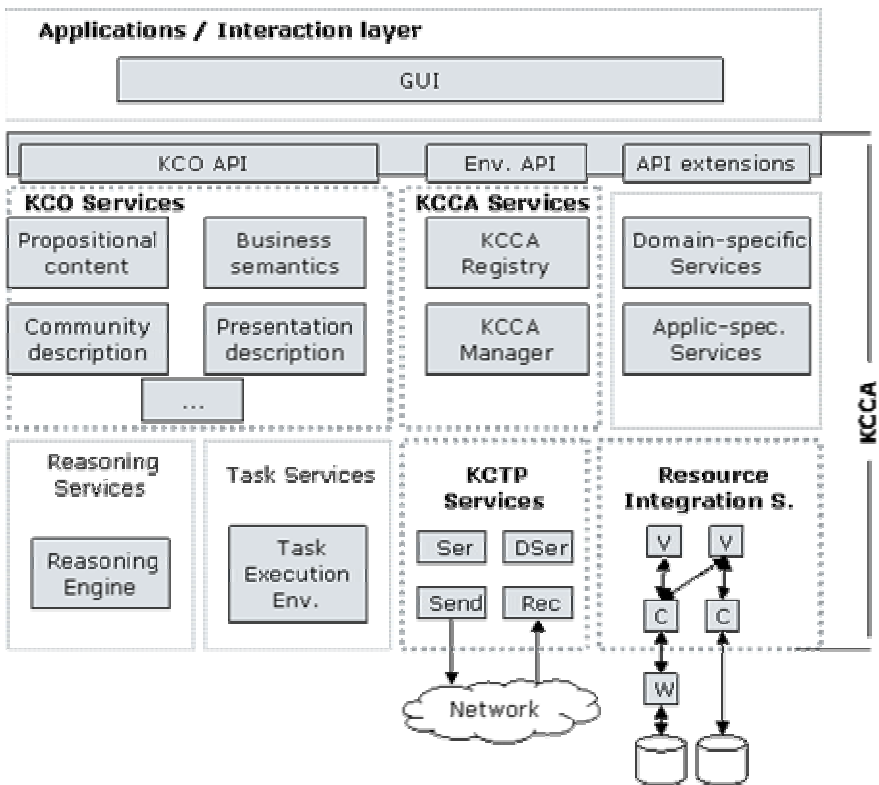


Fig. 3. Knowledge Content Carrier Architecture (KCCA)

The KCO services offer access to the operational semantics of the KCO facets. To achieve this, the structures that are defined in the *generic* KCO are mapped to an according O-O Model which is used for the implementation of the services. For *domain* KCOs, both the O-O model and its attendant services can be extended to cater for the application-specific functionality and semantics. It is assumed that the assignment of instance data (provided by KCO prototypes/instances) to types

(defined at the generic or domain level of the KCO) is done outside this building block (e.g. by reasoning based on the semantics defined at *generic KCO* and *domain KCO* Level).

The KCCA Registry and Manager component keeps track of how a federation of KCO aware information systems is set up. The KCCA environment keeps information about information sources, wrappers and maintains state in user sessions that may span requests and transactions across the federation.

The KCTP Services define a stateful protocol that allows communication between KCCA nodes by exchanging serialised RDF graphs. One specific service is the serialisation and de-serialisation (marshalling/unmarshalling) of KCOs in messages. The protocol is FIPA-based and can be implemented on top of SOAP or http.

The KCCA Integration Services give assistance in binding non-KCCA resources to a KCO aware system. This is done by a two-stage mapping process. The external information source is first mapped into an equivalent RDF schema which we call "context profile". This can be a "naive" mapping to RDF. Next, a view is defined over the context profile and this view is made KCO compliant. We call this the "view profile". The provider of an external information source needs to write a wrapper which provides the context profile for the resource. The KCCA integrator uses the context profile to create the view profile.

5 Summary and Open Issues

KCO are a flexible container structure for paid content that is enriched by dedicated semantic annotations grounded in foundational ontologies. They provide a solid basis for interoperable applications.

The foundational approach is used on one hand as a guideline for an efficient design of domain ontologies for content annotation, and provides on the other hand a minimal but shareable model for content interoperation between heterogeneous applications. The latter property will be leveraged by semantic search queries across KCCA infrastructures, i.e. such queries can be formulated on web objects classifiable as KCO level 2 (domain-specific KCOs) or KCO level 3 (instantiated KCOs). On level 2, partial instantiations can be added to a domain-specific KCO and used as requests.

The formal foundation of KCOs also secures that accessible KCO-compliant content repositories will deliver valid responses only. Because of the rich semantic structure of a KCO, requests can be defined on all content object aspects, i.e. descriptive content, reference community, business elements, and presentation issues.

The development of corresponding query languages and infrastructures that provide operational semantics is part of our current research within the EU project METOKIS (e.g. for KCO matching and composition, and interoperability with Semantic Web Services infrastructures). This will be used to integrate commercial content repositories in operational settings for three different domains: educational content production, clinical trial design and distributed news publishing.

References

1. Abecker, A., Apostolou, D., Maass, W., Mentzas, G., Reuschling, C. and Tabor, S., Towards an Information Ontology for Knowledge Asset Trading. in *ICE 2003 - 9th International Conference of Concurrent Enterprising*, (Espoo, Finland, 2003).
2. Bloom, P. A Decision Model for Prioritizing and Addressing Consumer Information Problems. *Journal of Public Policy & Marketing*, 8 (1). 161-180.
3. Boll, S. and Klas, W. ZYX - A Multimedia Document Model for Reuse and Adaptation. *IEEE Transactions on Knowledge and Data Engineering, DS-8 Special Issue*, 4.
4. Clarke, R. Electronic Commerce Definitions, 2000.
5. Franke, G., Huhmann, B. and Mothersbaugh, D. Information Content and Consumer Readership of Print Ads: A Comparison of Search and Experience Products. *Journal of the Academy of Marketing Science*, 32 (1). 20-31.
6. Gangemi, A., Borgo, S., Catenacci, C. and Lehmann, J. Task Taxonomies for Knowledge Content, METOKIS Deliverable, D07, 2004.
7. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosz, B.N. and Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML <http://www.w3.org/Submissions/SWRL/>, 2004.
8. Masolo, C., Borgo, S., A., G., Guarino, N. and Oltramari, A. The WonderWeb Library of Foundational Ontologies, 2003.
9. Reich, S., Behrendt, W., Eichinger, C. and M., M.S., Document Models for Navigating Digital Libraries. in *International Conference on Digital Libraries*, (Kyoto, 2000), 277-284.
10. Schellner, K., Westermann, U., Zillner, S. and W., K., CULTOS: Towards a World-Wide Digital Collection of Exchangeable Units of Multimedia Content for Inter-textual Studies. in *Conference on Distributed Multimedia Systems (DMS 2003)*, (Miami, Florida, 2003).
11. Schmid, B.F. and Lindemann, A., Elements of a Reference Model for Electronic Markets. in *HICSS*, (Kohala Coast, Hawaii, 1998), IEEE Computer Society, 193-201.
12. Shapiro, C. and Varian, H.R. *Information rules - A Strategic Guide to the Network Economy*. Harvard Business School Press, 1999.
13. Stahl, F., F., S. and Maass, W. Paid Content - Paid Services: Analysis of the German Market and Success Factors of 280 Business Models, =mcminstitute, University of St. Gallen, St. Gallen, 2004, 163.
14. van Ossenbruggen, J., Geurts, J., Cornelissen, F., Rutledge, L. and Hardman, L., Towards Second and Third Generation Web-Based Multimedia. in *The Tenth International World Wide Web Conference*, (Hong Kong, 2001), 479-488.
15. Varian, H.R., Markets for Information Goods. in *Monetary Policy in a World of Knowledge-Based Growth, Quality Change, and Uncertain Measurement*, (2000).
16. Whinston, A.B., Stahl, D.O. and Choi, S.Y. *The Economics of Electronic Commerce*. Macmillan Technical Publishing, Indianapolis, 1997.
17. Zillner, S., Westermann, U. and Winiwarter, W. EMMA - A Query Algebra for Enhanced Multimedia Meta Objects. *CoopIS/DOA/ODBASE*, 2. 1030-1049.

Efficient Semantic Matching

Fausto Giunchiglia¹, Mikalai Yatskevich¹, and Enrico Giunchiglia²

¹ Dept. of Information and Communication Technology,
University of Trento,
38050 Povo, Trento, Italy
{fausto, yatskevi}@dit.unitn.it

² DIST – Università di Genova,
Viale Causa 13, 16165, Genova, Italy
enrico@dist.unige.it

Abstract. We think of Match as an operator which takes two graph-like structures and produces a mapping between semantically related nodes. We concentrate on classifications with tree structures. In semantic matching, correspondences are discovered by translating the natural language labels of nodes into propositional formulas, and by codifying matching into a propositional unsatisfiability problem. We distinguish between problems with conjunctive formulas and problems with disjunctive formulas, and present various optimizations. For instance, we propose a linear time algorithm which solves the first class of problems. According to the tests we have done so far, the optimizations substantially improve the time performance of the system.

1 Introduction

We think of matching as the task of finding semantic correspondences between elements of two graph-like structures (e.g., conceptual hierarchies, classifications, database schemas or ontologies). Matching has been successfully applied in many well-known application domains, such as schema/ontology integration, data warehouses, and XML message mapping. In this paper we concentrate on classifications with tree structures.

Semantic matching, as introduced in [1, 5], is based on the key intuition that labels at nodes, which are written in natural language, are translated into propositional formulas which codify the intended meaning of the labels themselves. This allows us to codify the matching problem into a propositional unsatisfiability problem, which can then be efficiently implemented using state of the art propositional satisfiability (SAT) solvers [8, 9]. We call *concept of a label* the propositional formula which stands for the set of documents that one would classify under a label it encodes. We call *concept at a node* the propositional formula which represents the set of documents which one would classify under a node, given that it has a certain label and that it is in a certain position in a tree [5]. As from [5], all previous approaches, though implicitly or explicitly exploiting the semantic information codified in graphs, differ substantially from our approach in that they compute a syntactic “similarity” coefficients between labels in the [0,1] range (see for instance [3, 10]).

The system we have developed, called *S-Match* [6], takes two classifications and computes the strongest semantic relation holding between any pair of nodes. The matching problem is articulated into two macro steps, namely element and structure level matching. Element level matchers consider only the information on the atomic level [7] (the labels of nodes), while structure level matchers consider also the structure of the trees. Our goal in this paper is to describe the structure level matching algorithm, as it has been implemented within *S-Match*, and present a set of optimizations. In particular, we distinguish between two main classes of problems. In the first class all the concepts at nodes are *atomic* or *conjunctive* formulas. In the second class the concepts at nodes may also contain *disjunctive* formulas. In the case of conjunctive concepts at nodes we present a modification of the original algorithm which solves the node matching problem in linear time. With disjunctive concepts we present various techniques, which, among the other things, allow us to avoid the exponential space explosion which arises when converting disjunctive formulas into Conjunctive Normal Form (CNF). This modification is required since all state of the art SAT deciders take CNF formulas in input.

We have evaluated the time performance of the optimized algorithm against its basic version and several state of the art matching systems. The optimizations seem to improve substantially the time performance of *S-Match*. In all cases *S-Match* performs better or much better than the unoptimized version and always competes well with the other matching systems. In particular, it outperforms them on trees with hundreds or thousands of nodes.

The rest of the paper is organized as follows. Section 2 provides an overview of the *S-Match* tree matching algorithm. Section 3 discusses the basic node matching algorithm. The next two sections are dedicated to the two classes of node matching problems we have identified. Node matching problems with *conjunctive* concepts at nodes (and their optimizations) are discussed in Section 4, while the node matching problems with *disjunctive* concepts at nodes (and their optimizations) are described in Section 5. We discuss the evaluation results in Section 6. Section 7 concludes the paper.

2 The Tree Matching Algorithm

As from [6], the *S-Match* algorithm is organized according the following four macro steps:

- *Step 1*: for all labels in the two trees, compute concepts of labels;
- *Step 2*: for all nodes in the two trees, compute concepts at nodes;
- *Step 3*: for all pairs of labels in the two trees, compute the semantic relations between concepts of labels;
- *Step 4*: for all pairs of nodes in the two trees, compute the semantic relations between concepts at nodes.

The first two steps represent the pre-processing phase, while the third and the fourth steps correspond to the element-level and structure-level matching respectively. The semantic relations we consider are: *equivalence* (=); *more general*

($\supseteq$); *less general* ($\subseteq$); *disjointness* ($\perp$); *overlapping* ($\cap$). When none of the relations holds, the special *Idk* (I don't know or (?)) relation is returned.

The version of the algorithm defined in this paper assumes that:

- There are no negated atomic concepts of labels (one example of negated concept of label is $C_{\text{except apple}} = \neg C_{\text{apple}}$)
- The information we use, namely the labels of nodes and the knowledge residing in WordNet (see below) is all globally consistent. Under this assumption the only reason why we get an unsatisfiable formula is because we have found a match between two nodes

In order to understand how the algorithm works, consider for instance the two trees depicted in Figure 1a.

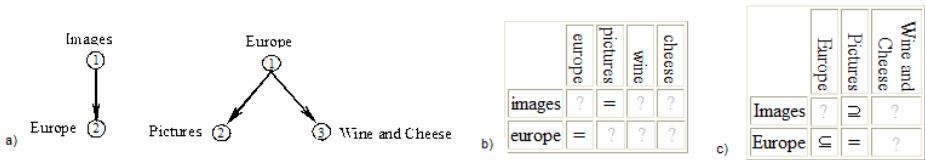


Fig. 1. (a): Two trees. (b): The matrix of relations between concepts of labels. (c): The matrix of relations between the concepts at nodes (matching result)

During *Step 1* we first tokenize labels. For instance “*Wine and Cheese*” becomes $\langle \text{Wine, and, Cheese} \rangle$. Then we lemmatize tokens. Thus for instance “*Images*” becomes “*image*”. Then, an Oracle (at the moment we use WordNet 2.0) is queried in order to obtain the senses of the lemmatized tokens. Afterwards, these senses are attached to atomic concepts. Finally, complex concepts are built suitably composing atomic concepts. Thus, the concept of the label *Wine and Cheese* is computed as $C_{\text{Wine and Cheese}} = \langle \text{wine, } \{\text{senses}_{\text{WN}\#4}\} \rangle \vee \langle \text{cheese, } \{\text{senses}_{\text{WN}\#4}\} \rangle$, where $\langle \text{cheese, } \{\text{senses}_{\text{WN}\#4}\} \rangle$ is taken to be the union of the four WordNet senses, and similarly for *wine*. Notice that natural language *and* is converted into logical disjunction rather than conjunction.

Step 2 takes into account the structural schema properties. The logical formula for a concept at a node is constructed most often as the conjunction of the concept of a label formulas in the concept path to the root [5]. For example, the concept C_2 for the node *Pictures* in Figure 1a is computed as $C_2 = C_{\text{Europe}} \wedge C_{\text{Pictures}}$.

Element level semantic matchers are applied during *Step 3*. They determine the semantic relations holding between pairs of atomic concepts of labels. For example, from WordNet we can derive that *image* and *picture* are synonyms, and therefore, $C_{\text{Images}} = C_{\text{Pictures}}$. Notice that *Image* and *Picture* have 8 and 11 senses in WordNet, respectively. In order to determine the senses which are relevant in the current context, sense filtering techniques are applied (see [11] for more details). The relations between the atomic concepts of labels for the trees depicted in Figure 1a are reported in Figure 1b.

Element level semantic matchers provide the input to the structure level matcher, which is applied in *Step 4*. This matcher produces the set of semantic relations between concepts at nodes (see Figure 1c for example). On this step the tree matching problem is reformulated into the set of node matching problems, one for each pair of nodes. Further, each node matching problem is reduced to a propositional validity problem.

The pseudo code of the Steps 3 and 4 of the semantic matching algorithm is reported in Figure 2. **treeMatch** takes 2 trees of Nodes (*source*, *target*) and returns the matrix of semantic relations between concepts at nodes in both trees (*cNodesMatrix*). First, **fillCLabMatrix** exploit element level semantic matchers library in order to fill the matrix of relations between concepts of labels in both trees (*cLabsMatrix*) (line 11). This action corresponds to the third step of the tree matching algorithm. Afterwards, two loops over all nodes of *source* and *target* trees are executed (lines 12-20 and 15-20). Within these loops, the propositional formulas corresponding to the concepts at nodes (*context_A*, *context_B*) are computed by **getNodeFormula** (lines 14, 17).

```

1. Node: struct of
2.     int nodeId;
3.     String label;
4.     String cLabel;
5.     String cNode;

6.String[] [] treeMatch(Tree of Nodes source, target)
7. Node sourceNode, targetNode;
8. String[] [] cLabsMatrix, cNodesMatrix, relMatrix;
9. String axioms, contextA, contextB;
10.int i,j;
11.cLabsMatrix=fillCLabMatrix(source,target);
12.For each sourceNode in source
13.    i=getNodeId(sourceNode);
14.    contextA=getNodeFormula (sourceNode);
15.    For each targetNode in target
16.        j=getNodeId(targetNode);
17.        contextB=getNodeFormula (targetNode);
18.        relMatrix=extractRelMatrix(cLabsMatrix,
                                   sourceNode, targetNode);
19.        axioms=mkAxioms (relMatrix);
20.        cNodesMatrix[i][j]=nodeMatch(axioms, contextA,
                                   contextB);
21. return cNodesMatrix;
```

Fig. 2. The pseudo code of the tree matching algorithm

relMatrix is calculated in the inner loop by **extractRelMatrix** (line 18). It contains the part of the *cLabsMatrix* relevant to the particular node matching problem. *axioms* (line 19) contains the conjunction of the propositional formulas in *relMatrix*. For example, the semantic relations in Figure 1b, which are considered

when we match *Europe* and *Pictures* are $Europe_A = Europe_B$, $Images_A = Pictures_B$. In this case $axioms$ is $(Europe_A \leftrightarrow Europe_B) \wedge (Images_A \leftrightarrow Pictures_B)$. Notice that, subscripts designate the *context* (either *A* or *B*) to which a propositional variable (or concept) belongs. The detailed description of **nodeMatch** is provided in the next section.

3 The Node Matching Algorithm

nodeMatch input formulas are combined to obtain the following formula:

$$(axioms) \rightarrow rel(context_A, context_B), \quad (1)$$

where $axioms$, $context_A$, $context_B$ are as defined in **treeMatch** (Figure 2), while $rel(context_A, context_B)$ is the formula corresponding to the semantic relation being checked, (namely equivalence, less or more generality, or disjointness). As from [5], two nodes match if and only if Eq. 1 is valid, namely if it is *true* for all possible truth assignments to its propositional variables. Given that most of the available propositional solvers are satisfiability checkers, the negation of the matching formula is checked for unsatisfiability. This yields the following formula

$$axioms \wedge \neg rel(context_A, context_B) \quad (2)$$

Table 1 reports the resulting matching formulas as a function of the semantic relation being tested. Notice that the check for equality is omitted. In fact $A = B$ holds iff $A \subseteq B$ and $A \supseteq B$ hold.

Table 1. The relationship between semantic relations and propositional formulas

$rel(a, b)$	Translation of $rel(a, b)$ in propositional logic	CNF translation of Eq. 2
$a = b$	$a \leftrightarrow b$	<i>N/A</i>
$a \subseteq b$	$a \rightarrow b$	$axioms \wedge context_A \wedge \neg context_B$
$a \supseteq b$	$b \rightarrow a$	$axioms \wedge context_B \wedge \neg context_A$
$a \perp b$	$\neg(a \wedge b)$	$axioms \wedge context_A \wedge context_B$

Consider the pseudo code of the node matching algorithm, as described in Figure 3.

nodeMatch constructs the formulas needed for testing less generality (line 120) and more generality (line 150), it converts them to CNF (lines 130, 160) and checks for unsatisfiability (lines 140, 170). If both relations hold, then the equivalence relation is returned (line 190). Afterwards, the same procedure is repeated for disjointness test. If all the tests fail “*Idk*” is returned (line 290).

Prior to the discussion of optimizations to our basic solution, let us classify the concepts of labels and concepts at nodes. We distinguish between four categories of concepts of labels:

```

110. String nodeMatch(String axioms, contextA, contextB)
120. String formula=And(axioms, contextA, Not(contextB));
130. String formulaInCNF=convertToCNF(formula);
140. boolean isLG=isUnsatisfiable(formulaInCNF)
150. formula=And(axioms, Not(contextA), contextB);
160. formulaInCNF=convertToCNF(formula);
170. boolean isMG=isUnsatisfiable(formulaInCNF);
180. if (isMG && isLG)
190.   return "=";
200. if (isLG)
210.   return "⊆";
220. if (isMG)
230.   return "⊇";
240. formula= And(axioms, contextA, contextB);
250. formulaInCNF=convertToCNF(formula);
260. boolean isOpposite= isUnsatisfiable(formulaInCNF);
270. if (isOpposite)
280.   return "⊥";
290. return "Idk";

```

Fig. 3. The pseudo code of the node matching algorithm

- **Atomic:** the concept of a label is an atomic proposition. For example, the concept of the label *Europe* is $C_{Europe} = \langle Europe, \{senses_{WN\#I}\} \rangle$, where $senses_{WN\#I}$ stands for a WordNet sense.
- **Conjunctive:** the concept of a label is a conjunction. For example, the concept of the label *transmission gearbox* is $C_{transmission\ gearbox} = C_{transmission} \wedge C_{gearbox}$.
- **Disjunctive:** the concept of a label is a disjunction. For example, the concept of the label *jet and trains and cars* is $C_{jet\ and\ trains\ and\ cars} = C_{jet} \vee C_{train} \vee C_{car}$.
- **Full proposition at logic:** the concept of a label contains both conjunctions and disjunctions. For example the concept of the label *computers and electrical equipment* is $C_{computers\ and\ electrical\ equipment} = C_{computer} \vee (C_{electrical} \wedge C_{equipment})$.

This classification allows us to further distinguish between two classes of *concepts at nodes*, which are at the basis of our optimizations:

- **Conjunctive concepts at nodes:** the concept at a node is a conjunction.
- **Disjunctive concepts at nodes:** the concept at a node contains both conjunctions and disjunctions in any order.

4 Conjunctive Concepts at Nodes

4.1 Node Matching Problems

Consider the two trees depicted in Figure 4a. Notice that they have only atomic concepts of labels. Let us consider the matching of *gearbox* and *clutch*.

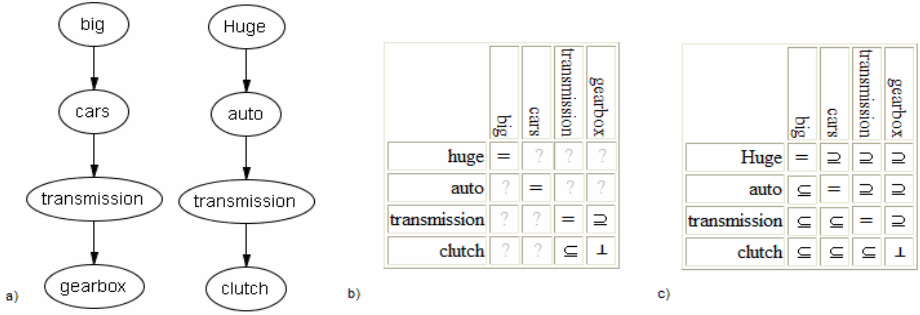


Fig. 4. (a): Two trees. (b): The matrix of relations between concepts of labels. (c): The matrix of relations between concepts at nodes (matching result)

The relevant semantic relations between concepts of labels are depicted in Figure 4b. As from Table 1, *axioms* is:

$$(big_A \leftrightarrow huge_B) \wedge (car_A \leftrightarrow auto_B) \wedge (transmission_A \leftrightarrow transmission_B) \wedge (gearbox_A \rightarrow transmission_B) \wedge (clutch_B \rightarrow transmission_A) \wedge \neg(clutch_B \wedge gearbox_A) \quad (3)$$

which, translated in CNF, becomes:

$$(\neg big_A \vee huge_B) \wedge (big_A \vee \neg huge_B) \wedge (\neg car_A \vee auto_B) \wedge (car_A \vee \neg auto_B) \wedge (\neg transmission_A \vee transmission_B) \wedge (transmission_A \vee \neg transmission_B) \wedge (\neg gearbox_A \vee transmission_B) \wedge (\neg clutch_B \vee transmission_A) \wedge (\neg clutch_B \vee \neg gearbox_A) \quad (4)$$

As from Step 2 in Section 2, $context_A$ and $context_B$ are constructed by taking the conjunction of the concepts of labels in the path to root. Therefore, $context_A$ and $context_B$ are:

$$big_A \wedge car_A \wedge transmission_A \wedge gearbox_A \quad (5)$$

$$huge_B \wedge auto_B \wedge transmission_B \wedge clutch_B \quad (6)$$

while their negations are:

$$\neg big_A \vee \neg car_A \vee \neg transmission_A \vee \neg gearbox_A \quad (7)$$

$$\neg huge_B \vee \neg auto_B \vee \neg transmission_B \vee \neg clutch_B \quad (8)$$

Let us consider the formula to be checked for unsatisfiability, as from Table 1. The first observation is that *axioms* remains the same for all the tests, and it contains only clauses with two variables, where a clause is a finite disjunction of literals. In the worst case it contains $2 * n_A * n_B$ clauses, where n_A and n_B are the number of atomic concepts of labels in the paths to the root (in our example n_A and n_B are equal to 4). The second observation is that the formulas for less and more generality are very similar and differ only in the context formula which is negated. Thus, for instance, in the less generality test $context_B$ is negated. This means that Eq. 1 contains one clause with n_B variables (Eq. 8) in addition to n_A clauses with one variable derived from $context_A$ (Eq. 5). Finally, again from Table 1, in the case of disjointness test $context_A$

and $context_B$ are not negated. Therefore, Eq. 1 contains $n_A + n_B$ clauses with one variable (Eq. 5 and Eq. 6).

So far we have concentrated on atomic concepts of labels. The propositional formulas remain the same if we move to conjunctive concepts at labels. Consider the trees depicted in Figure 5a. Let us consider the matching between *transmission gearbox* and *transmission clutch*.

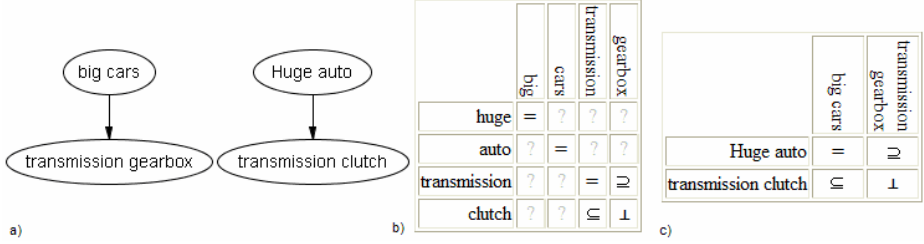


Fig. 5. (a): Two trees. (b): The matrix of relations between concepts of labels in the trees. (c): The matrix of relations between concepts at nodes (matching result)

Compare the matrices on the Figure 5b and Figure 4b. They are the same. The matrix of the relations between concepts of labels unambiguously determines *axioms* (see Eq. 3 and 4). Furthermore, as from Step 2 in Section 2, the propositional formulas for $context_A$ and $context_B$ are the same for atomic and for conjunctive concepts of labels as long as they “globally” contain the same formulas. In fact, concepts at nodes are constructed by taking the conjunction of concepts at labels. Splitting a concept of a label with two conjuncts into two atomic concepts has no effect on the resulting matching formula.

4.2 Optimizations

Let us consider first more and less generality and then disjointness.

4.2.1 Less and More Generality Tests

As from Section 4.1, formula (Eq. 1) in this case is as follows:

$$\underbrace{\bigwedge_{q=0}^{n*m} (\neg A_s \vee B_t) \wedge \bigwedge_{w=0}^{n*m} (A_k \vee \neg B_l) \wedge \bigwedge_{v=0}^{n*m} (\neg A_p \vee \neg B_r)}_{Axioms} \wedge \underbrace{\bigwedge_{i=1}^n A_i}_{Context_A} \wedge \underbrace{\bigvee_{j=1}^m \neg B_j}_{\neg Context_B} \quad (9)$$

where n is the number of variables in $context_A$, m is the number of variables in $context_B$. A_i 's belong to $context_A$, and B_j 's belong to $context_B$. s, k, p are in the $[0..n]$ range, while t, l, r are in the $[0..m]$ range. *Axioms* can be empty. Eq. 9 is composed of clauses with 1 or 2 variables plus one clause with possibly more variables (the clause

corresponding to the negated context). The key observation is that the formula in Eq. 9 is Horn: each clause contains at most one positive literal. Therefore, the satisfiability problem can be decided in linear time by the unit resolution rule [2]. Notice, that DPLL-based SAT solvers require quadratic time in this case [15].

In order to understand how the linear time algorithm works, let us prove the unsatisfiability of Eq. 9 in the case of *gearbox* and *clutch*. In this case, Eq. 9 becomes

$$\begin{aligned}
 & ((\neg \mathbf{big}_A \vee \mathbf{huge}_B) \wedge (\mathbf{big}_A \vee \neg \mathbf{huge}_B) \wedge (\neg \mathbf{car}_A \vee \mathbf{auto}_B) \wedge (\mathbf{car}_A \vee \neg \mathbf{auto}_B) \wedge \\
 & (\neg \mathbf{transmission}_B \vee \mathbf{transmission}_A) \wedge (\mathbf{transmission}_B \vee \neg \mathbf{transmission}_A) \wedge \\
 & (\neg \mathbf{gearbox}_A \vee \mathbf{transmission}_B) \wedge (\neg \mathbf{clutch}_B \vee \mathbf{transmission}_A) \wedge \\
 & (\neg \mathbf{clutch}_B \vee \neg \mathbf{gearbox}_A) \wedge \mathbf{big}_A \wedge \mathbf{car}_A \wedge \mathbf{transmission}_A \wedge \mathbf{gearbox}_A \wedge \\
 & (\neg \mathbf{huge}_B \vee \neg \mathbf{auto}_B \vee \neg \mathbf{transmission}_B \vee \neg \mathbf{clutch}_B)
 \end{aligned} \tag{10}$$

where the variables from $context_A$ are written in bold.

First, we assign *true* to all unit clauses occurring in Eq 10 positively. Notice that these are all and only the clauses in $context_A$. This allows us to discard the clauses where $context_A$ variables occur positively (in this case: $\mathbf{big}_A \vee \neg \mathbf{huge}_B$, $\mathbf{car}_A \vee \neg \mathbf{auto}_B$, $\neg \mathbf{gearbox}_A \vee \mathbf{transmission}_B$ and $\neg \mathbf{clutch}_B \vee \mathbf{transmission}_A$). The resulting formula is

$$\begin{aligned}
 & \mathbf{huge}_B \wedge \mathbf{auto}_B \wedge \mathbf{transmission}_B \wedge \neg \mathbf{clutch}_B \wedge \\
 & (\neg \mathbf{huge}_B \vee \neg \mathbf{auto}_B \vee \neg \mathbf{transmission}_B \vee \neg \mathbf{clutch}_B)
 \end{aligned} \tag{11}$$

Notice that this formula does not contain any variable derived from $context_A$. Notice also that, by assigning *true* to $\mathbf{huge}_B$, $\mathbf{auto}_B$ and $\mathbf{transmission}_B$ and *false* to $\mathbf{clutch}_B$ we do not derive a contradiction. Therefore, (Eq. 10) is satisfiable. In fact, a (Horn) formula is unsatisfiable if and only if the empty clause is derived (and satisfiable otherwise).

Consider again Eq. 11. For this formula to be unsatisfiable all the variables occurring in the negation of $context_B$ ($\neg \mathbf{huge}_B \vee \neg \mathbf{auto}_B \vee \neg \mathbf{transmission}_B \vee \neg \mathbf{clutch}_B$ in our example) should occur positively in the unit clauses obtained after resolving *Axioms* with the unit clauses in $context_A$ ($\mathbf{huge}_B$, $\mathbf{auto}_B$ and $\mathbf{transmission}_B$ in our example). But for this to happen, for any B_j in $context_B$ there must be a clause of form $\neg A_i \vee B_j$ in *axioms*, where A_i is a formula of $context_A$. But formulas of the form $\neg A_i \vee B_j$ occur in Eq. 9 if and only if we have the axioms of the form $A = B_j$ and $A_i \subseteq B_j$. These considerations suggest the following algorithm for testing satisfiability:

- Step 1. Create an array of size m . Each entry in the array stands for one B_j in Eq. 9.
- Step 2. For each axiom of type $A_i = B_j$ and $A_i \subseteq B_j$ mark the corresponding B_j .
- Step 3. If all the B_j 's are marked, then the formula is unsatisfiable.

nodeMatch can be modified as in Figure 6 (the numbers on the left indicate where the new code must be positioned):

```

111.  if (contextA and contextB are conjunctive)
112.    isLG=fastHornUnsatCheck (contextA, axioms, "⊆");
113.    isMG=fastHornUnsatCheck (contextB, axioms, "⊇");
114.  else

301. boolean fastHornUnsatCheck(String context, axioms,
                                rel);

302. int m=getNumOfVar(String context);
303. boolean array[m];
304. for each axiom in axioms
305.   if ((getAType(axiom)="=") || (getAType(axiom)=rel))
306.     int j=getNumberOfSecondVariable(axiom);
307.     array[j]=true;
308. for (i=0; i<m; i++)
309.   if (!array[i])
310.     return false;
311. return true;

```

Fig. 6. Less and more generality tests optimization pseudo code

fastHornUnsatCheck implements the three steps above. Step 1 is performed in lines (302-303). Then, a loop on *axioms* (lines 304-307) implements Step 2. The final loop (lines 308-310) implements Step 3.

4.2.2 Disjointness Test

Using the same notation as in Section 4.2.1, formula (Eq. 1) is as follows:

$$\overbrace{\bigwedge_{q=0}^{n*m} (\neg A_s \vee B_t) \wedge \bigwedge_{w=0}^{n*m} (A_k \vee \neg B_l) \wedge \bigwedge_{v=0}^{n*m} (\neg A_p \vee \neg B_r)}^{Axioms} \wedge \overbrace{\bigwedge_{i=1}^n A_i}^{Context_A} \wedge \overbrace{\bigwedge_{j=1}^m B_j}^{Context_B} \quad (12)$$

For example, the formula for testing disjointness between *gearbox* and *clutch* is

$$\begin{aligned}
 &(\neg \mathbf{big}_A \vee \mathbf{huge}_B) \wedge (\mathbf{big}_A \vee \neg \mathbf{huge}_B) \wedge (\neg \mathbf{car}_A \vee \mathbf{auto}_B) \wedge (\mathbf{car}_A \vee \neg \mathbf{auto}_B) \wedge \\
 &(\neg \mathbf{transmission}_B \vee \mathbf{transmission}_A) \wedge (\mathbf{transmission}_B \vee \neg \mathbf{transmission}_A) \wedge \\
 &(\neg \mathbf{gearbox}_A \vee \mathbf{transmission}_B) \wedge (\neg \mathbf{clutch}_B \vee \mathbf{transmission}_A) \wedge \\
 &(\neg \mathbf{clutch}_B \vee \neg \mathbf{gearbox}_A) \wedge \mathbf{big}_A \wedge \mathbf{car}_A \wedge \mathbf{transmission}_A \wedge \mathbf{gearbox}_A \wedge \\
 &\quad \mathbf{huge}_B \wedge \mathbf{auto}_B \wedge \mathbf{transmission}_B \wedge \mathbf{clutch}_B
 \end{aligned} \quad (13)$$

Here again, the formula in Eq. 12 is Horn and thus, similarly to Section 4.2.1, the satisfiability of the formula can be decided by unit propagation. After assigning *true* to all the variables in *context_A* and propagating the results we obtain the following formula:

$$\mathbf{huge}_B \wedge \mathbf{auto}_B \wedge \mathbf{transmission}_B \wedge \neg \mathbf{clutch}_B \wedge \mathbf{huge}_B \wedge \mathbf{auto}_B \wedge \mathbf{transmission}_B \wedge \mathbf{clutch}_B \quad (14)$$

If we further unit propagate $huge_B$, $auto_B$ and $transmission_B$ (this means that we assign true to them), then get the contradiction $clutch_B \wedge \neg clutch_B$. Therefore, the formula is unsatisfiable. This contradiction arises because $(\neg clutch_B \vee \neg gearbox_A)$ occurs in Eq. 13, which, in turn, is derived (as from Table 1) from the disjointness axiom ($clutch_B \perp gearbox_A$). In fact, all the clauses in Eq. 12 contain one positive literal except for the clauses in *axioms* corresponding to disjointness relations. Thus, the key intuition here is that if there are no disjointness axioms, then Eq. 12 is satisfiable. On the other hand, if there is a disjointness axiom, atoms occurring there are also ensured to be either in $context_A$ or in $context_B$ and thus Eq. 12 is unsatisfiable. Therefore, the optimization consists of just checking the presence/absence of disjointness axioms in *axioms*.

The pseudo code of **nodeMath** can therefore be modified as follows:

```
231. If (contextA and contextB are conjunctive)
232.   If (there is disjointness axiom in the axioms)
233.     isOpposite=true;
234.   else
235.     isOpposite=false;
236. else
```

Fig. 7. Disjointness test optimization pseudo code

5 Disjunctive Concepts at Nodes

5.1 The Node Matching Problem

Consider the trees depicted in Figure 8a. Notice that the concepts at nodes contain disjunctive concepts of labels. Let us consider matching *fifties or sixties or seventies* with *twenties or thirties or forties*.

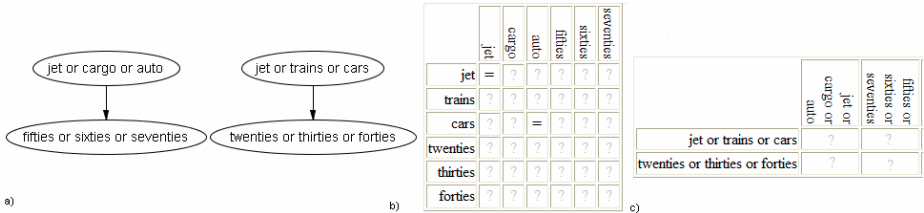


Fig. 8. (a): Two trees. (b): The matrix of relations between concepts of labels in the trees. (c): The matrix of relations between concepts at nodes (matching result)

The relations between atomic concepts of labels in both trees are depicted in Figure 8b. As from the second column of Table 1 *axioms* is:

$$(cars_B \leftrightarrow auto_A) \wedge (jet_A \leftrightarrow jet_B) \quad (15)$$

which can be rewritten as:

$$(\neg cars_B \vee auto_A) \wedge (cars_B \vee \neg auto_A) \wedge (\neg jet_A \vee jet_B) \wedge (jet_A \vee \neg jet_B) \quad (16)$$

As from Step 2 in Section 2 $context_A$ and $context_B$ are:

$$(jet_A \vee cargo_A \vee auto_A) \wedge (fifties_A \vee sixties_A \vee seventies_A) \quad (17)$$

$$(jet_B \vee train_B \vee cars_B) \wedge (twenties_B \vee thirties_B \vee forties_B) \quad (18)$$

The negations of $context_A$ and $context_B$ are:

$$(\neg jet_A \wedge \neg cargo_A \wedge \neg auto_A) \vee (\neg fifties_A \wedge \neg sixties_A \wedge \neg seventies_A) \quad (19)$$

$$(\neg jet_B \wedge \neg train_B \wedge \neg cars_B) \vee (\neg twenties_B \wedge \neg thirties_B \wedge \neg forties_B) \quad (20)$$

Let us consider the formula to be tested for unsatisfiability, as from Table 1. Again, $axioms$ is the same for all the tests. As from Section 4.1, it consists up to $2^{*n_A * n_B}$ clauses with two variables, where n_A and n_B are the number of atomic concepts of labels in the paths to root. In our example n_A and n_B are both equal to 6. The key observation here is that $context_A$ and $context_B$ may contain any number of disjunctions. Some exist because derived from the labels, while others may be obtained by negating $context_A$ or $context_B$ (as from the above example, in the case of less and more generality tests). Thus, for instance, as from Table 1 in case of less generality test we obtain the formula.

$$\begin{aligned} &(\neg cars_B \vee auto_A) \wedge (cars_B \vee \neg auto_A) \wedge (\neg jet_A \vee jet_B) \wedge (jet_A \vee \neg jet_B) \wedge \\ & (jet_A \vee cargo_A \vee auto_A) \wedge (fifties_A \vee sixties_A \vee seventies_A) \wedge \\ & ((\neg jet_B \wedge \neg train_B \wedge \neg cars_B) \vee (\neg twenties_B \wedge \neg thirties_B \wedge \neg forties_B)) \end{aligned} \quad (21)$$

5.2 Optimizations

With disjunctive concepts at nodes, Eq. 1 is a full propositional formula and no hypothesis can be made on its structure. As a consequence its satisfiability must be tested using a standard DPLL SAT solver. Thus for instance CNF conversion of Eq. 21 is

$$\begin{aligned} &(\neg cars_B \vee auto_A) \wedge (cars_B \vee \neg auto_A) \wedge (\neg jet_A \vee jet_B) \wedge (jet_A \vee \neg jet_B) \wedge \\ & (jet_A \vee cargo_A \vee auto_A) \wedge (fifties_A \vee sixties_A \vee seventies_A) \wedge \\ & ((\neg jet_B \vee \neg twenties_B) \wedge (\neg jet_B \vee \neg thirties_B) \wedge (\neg jet_B \vee \neg forties_B) \wedge \\ & (\neg train_B \vee \neg twenties_B) \wedge (\neg train_B \vee \neg thirties_B) \wedge (\neg train_B \vee \neg forties_B) \wedge \\ & (\neg cars_B \vee \neg twenties_B) \wedge (\neg cars_B \vee \neg thirties_B) \wedge (\neg cars_B \vee \neg forties_B)) \end{aligned} \quad (22)$$

In order to avoid the space explosion, which may arise when converting a formula to CNF (see for instance Eq. 22), we apply a set of structure preserving transformations [14, 4]. The main idea is to replace disjunctions occurring in the original formula with newly introduced variables and explicitly state that these variables imply the subformulas they substitute. Consider for instance Eq. 21. We obtain:

$$\begin{aligned}
& (\neg cars_B \vee auto_A) \wedge (cars_B \vee \neg auto_A) \wedge (\neg jet_A \vee jet_B) \wedge (jet_A \vee \neg jet_B) \wedge \\
& (jet_A \vee cargo_A \vee auto_A) \wedge (fifties_A \vee sixties_A \vee seventies_A) \wedge (new_1 \vee new_2) \wedge \\
& (\neg new_1 \vee \neg jet_B \vee \neg train_B \vee \neg car_B) \wedge (\neg new_2 \vee \neg twenties_B \vee \neg thirties_B \vee \neg forties_B)
\end{aligned} \tag{23}$$

Notice that the size of the propositional formula in CNF grows linearly with respect to number of disjunctions in original formula.

To account for this optimization in **nodeMatch** all calls to **convertToCNF** are replaced with calls to **optimizedConvertToCNF**, (see Figure 9):

```

130. formulaInCNF=optimizedConvertToCNF(formula);
...
160. formulaInCNF=optimizedConvertToCNF(formula);
...
250. formulaInCNF=optimizedConvertToCNF(formula);

```

Fig. 9. The CNF conversion optimization pseudo code

6 Evaluation Results

We have implemented the optimizations described above and evaluated the resulting system *S-Match* against the original system and two state of the art matching systems, namely COMA [3] and Similarity Flooding (SF) [12] as implemented in Rondo system [13]. Let us call *S-Match_B* the original version without optimizations. Notice that S-Match, COMA, and SF exploit different matching techniques and differ substantially in the quality of matching results. See [6] for a detailed comparison

Table 2. The structural properties of the trees in the matching problems

	Trees max. depth	# of nodes per tree	# of labels per tree	Average # of labels per node	Concepts at nodes
Cornell-Washington with atomic concepts of labels	10/8	253/220	253/220	1/1	Conjunctive
Handmade trees with disjunctive concepts of labels	10/10	10/10	30/30	3/3	Disjunctive
Looksmart-Yahoo	10/8	140/74	222/101	1,58/1,36	Conjunctive Disjunctive
Yahoo-Standard	3/3	333/115	965/242	2,9/2,1	Conjunctive Disjunctive
Google-Yahoo	11/11	561/665	722/945	1,28/1,42	Conjunctive Disjunctive
Google-Looksmart	11/16	706/1081	1048/1715	1,48/1,63	Conjunctive Disjunctive

among these systems. In this evaluation we have concentrated only on the time performance of the systems. The tests have been performed on a P4 computer with 512 MB of RAM installed. The systems were limited to allocate no more than 512 MB of memory.

The systems have been tested on the six matching problems which can be found at <http://dit.unitn.it/~accord/>. Table 3 reports the properties of these problems.

6.1 Conjunctive Concepts at Nodes

On this problem $S\text{-Match}_B$ works two times faster than COMA. In fact, in this case the DPLL SAT solver of $S\text{-Match}$ runs in polynomial time. $S\text{-Match}$ instead works more than 5 times faster than COMA. However it still runs about 17% slower than SF. This can be explained by noticing that in SF the similarities between the labels of nodes obtained by a simple and fast string matcher, and propagated through a graph structure using a fix point algorithm. This algorithm is very fast and, on these examples, it converges after a few iterations. The drawback of SF, as the last test below shows, is that it requires a much larger amount of memory.

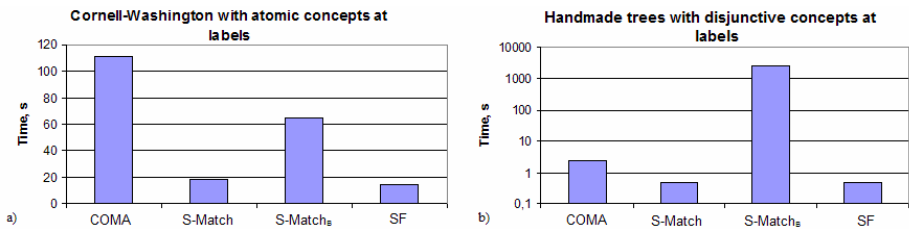


Fig. 10. Execution time of the matching systems

6.2 Disjunctive Concepts at Nodes

Let us consider the test with handmade trees. As from Figure 10b, $S\text{-Match}$ works about 4 orders of magnitude faster than $S\text{-Match}_B$, about 4 times faster than COMA, and as fast as SF. The significant improvement of the optimized algorithm can be explained by considering that $S\text{-Match}_B$ does not control the exponential space explosion on such trees. In fact, the biggest formula in this case consists of about 118000 clauses. The optimization introduced in the Section 5.2 reduces this number to about 20-30 clauses.

We have then considered 4 matching problems involving real world classifications. Three of them, Looksmart-Yahoo, Google-Yahoo, and Google-Looksmart, involve web directories. The forth involves parts of the Yahoo and the Standard catalogues which describe business activities. The results obtained for the Looksmart-Yahoo matching problem are depicted in Figure 11a. In this case the trees contain about 100 nodes each. $S\text{-Match}$ works about 18% faster than $S\text{-Match}_B$ and about 2 % slower than COMA. SF works about 3 times faster. The relatively poor improvement (18%) can be explained by the fact that our optimizations are

implemented in a straightforward way. The higher implementational constants on small trees (like Looksmart-Yahoo) can overcome the order of growth the complexity function.

Figure 11b reports the results obtained for the Yahoo-Standard matching problem. *S-Match* works about 40% faster than *S-Match_B*. It performs 1% faster than COMA and about 5 times slower than SF. The relatively small improvement in this case can be explained by noticing that the maximum depth in both trees is 3 and that the average number of labels at node is about 2. The optimizations can not significantly influence on the system performance.

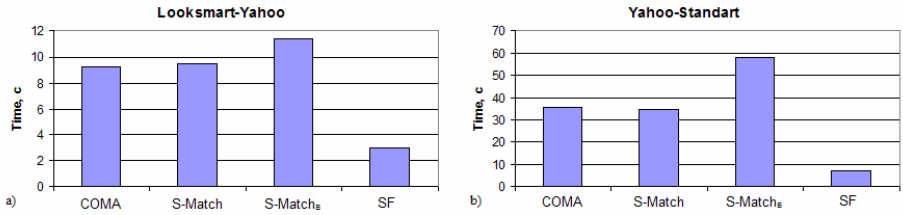


Fig. 11. Execution time of the matching systems

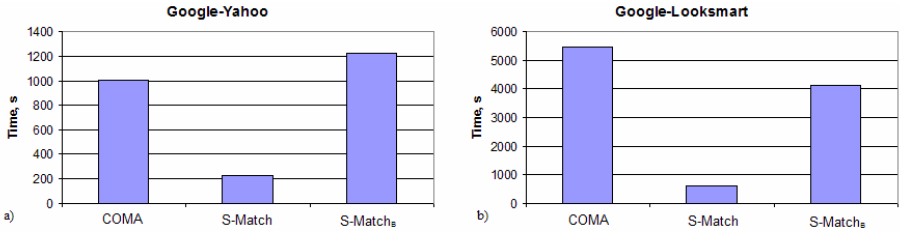


Fig. 12. Execution time of the matching systems

The next two matching problems are much bigger than the previous ones. They contain hundreds and thousands of nodes. On these trees SF went out of memory. Therefore, we provide the results only for the other systems. The results are reported in Figure 12a. *S-Match* is more than 6 times faster than *S-Match_B*. COMA performs about 5 times slower than the optimized version. These results suggest that the optimizations described in this paper are better suited for big schemas. The results of the biggest matching problem, involving Google-Looksmart, are presented in Figure 12b. In this case *S-Match* performs about 9 times faster than COMA, and about 7 times faster than *S-Match_B*.

8 Conclusion

We have presented a structure level semantic matching algorithm and proposed several optimizations to its original version. In particular we have distinguished

between two main classes of problems, namely the problems with *conjunctive* and with *disjunctive* concepts at nodes. For the first class of problems we have presented a modification to the original algorithm which solves the node matching problem in linear time. With *disjunctive* concepts we have presented various techniques, which allow us to avoid the exponential space explosion which arises when converting disjunctive formulas into CNF. We have evaluated *S-Match* against several state of the art matching systems and against the original unoptimized version, *S-Match_B*. The results thorough preliminary are promising. *S-Match* always performs better than *S-Match_B*. Furthermore, in most cases *S-Match* competes well, in terms of time performance, with various state of the art matching systems. Optimizations are most effective on big trees with hundreds and thousands of nodes.

Acknowledgements. This work has been partially supported by the European Knowledge Web network of excellence (IST-2004-507482) and by the research grant COFIN 2003 Giunchiglia 40100657.

References

- [1] P. Bouquet, L. Serafini, S. Zanobini. Semantic Coordination: A new approach and an application. In Proceedings of ISWC 2003.
- [2] M. Davis and H. Putnam. A computing procedure for quantification theory. In *Journal of the ACM*, number 7, pages 201–215, 1960.
- [3] H. Do, E. Rahm. COMA - A system for Flexible Combination of Schema Matching Approaches, In Proceedings of VLDB 2002
- [4] E. Giunchiglia, R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas . In AIIA'99.
- [5] F. Giunchiglia, P. Shvaiko. Semantic Matching. In The Knowledge Engineering Review Journal, 18(3) 2003.
- [6] F. Giunchiglia, P. Shvaiko, M. Yatskevich. S-Match: An algorithm and an implementation of semantic matching. In Proceedings of ESWS'04.
- [7] F. Giunchiglia and M. Yatskevich. Element level semantic matching. In Proceedings of Meaning Coordination and Negotiation workshop at ISWC, 2004.
- [8] D. Le Berre JSAT: The java satisfiability library. <http://cafe.newcastle.edu.au/daniel/JSAT/i>
- [9] D. Le Berre SAT4J: A satisfiability library for Java. <http://www.sat4j.org/>.
- [10] J. Madhavan, P. Bernstein, E. Rahm. Generic Schema Matching with Cupid. VLDB 2001
- [11] B. Magnini, M. Speranza, C. Girardi. A Semantic-based Approach to Interoperability of classification Hierarchies: Evaluation of Linguistic Techniques. In: Proceedings of COLING-2004, Geneva, Switzerland, August 23-27, 2004.ï
- [12] S. Melnik,, H. Garcia-Molina, E. Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm. Proceedings of ICDE, (2002) 117-128.
- [13] S. Melnik, E. Rahm, P. Bernstein: Rondo: A programming platform for generic model management. Proceedings of SIGMOD'03, (2003) 193-204.ï
- [14] D. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. Journal of Symbolic Computation, 2:293-304, 1986
- [15] G. Tsetin. On the complexity proofs in propositional logics. Seminars in Mathematics, 8, 1970

Appendix A. The Pseudo Code of the Optimized S-Match Algorithm

```

1. Node: struct of
2.     int nodeId;
3.     String label;
4.     String cLabel;
5.     String cNode;

6.String[][] treeMatch(Tree of Nodes source, target)
7. Node sourceNode, targetNode;
8. String[][] cLabsMatrix, cNodesMatrix, relMatrix;
9. String axioms, contextA, contextB;
10.int i, j;
11.cLabsMatrix=fillCLabMatrix(source, target);
12.For each sourceNode in source
13. i=getNodeId(sourceNode);
14. contextA=getCnodeFormula (sourceNode);
15. For each targetNode in target
16. j=getNodeId(targetNode);
17. contextB=getCnodeFormula (targetNode);
18. relMatrix=extractRelMatrix(cLabMatrix,
                             sourceNode, targetNode);
19. axioms=mkAxioms(relMatrix);
20. cNodesMatrix[i][j]=nodeMatch(axioms,
                                   contextA, contextB);
21. return cNodesMatrix;

110.String nodeMatch(String axioms, contextA, contextB)
111. if (contextA and contextB are conjunctive)
112.  isLG= fastHornUnsatCheck (contextA, axioms, " $\subseteq$ ")
113.  isMG= fastHornUnsatCheck (contextB, axioms, " $\supseteq$ ")
114. else
120. String formula=And(axioms, contextA, Not(contextB))
130. String formulaInCNF=optimizedConvertToCNF(formula)
140. boolean isLG=isUnsatisfiable(formula)
150. formula=And(axioms, Not(contextA), contextB);
160. formulaInCNF= optimizedConvertToCNF (formula);
170. boolean isMG= isUnsatisfiable(formula);
180. if (isMG && isLG)
190.  return "=";
200.if (isLG)
210.  return " $\subseteq$ ";
220.if (isMG)
230.  return " $\supseteq$ ";
231. If (contextA and contextB are conjunctive)
232.  If (there is disjointness axiom in the axioms)
233.    isOpposite=true;
234.  else

```

```

235.      isOpposite=false;
236. else
240.   formula= And(axioms, contextA, contextB);
250.   formulaInCNF= optimizedConvertToCNF (formula);
260.   boolean isOpposite= isUnsatisfiable(formula);
270. if (isOpposite)
280.   return "⊥";
290. return "Idk";

301. boolean fastHornUnsatCheck(String context, axioms,
                                rel)
302. int m=getNumOfVar(String context);
303. boolean array[m];
304. for each axiom in axioms
305.   if((getAType(axiom)="=") || (getAType(axiom)= rel))
306.     int j=getNumberOfSecondVariable(axiom);
307.     array[j]=true;
308.   for (i=0; i<m; i++)
309.     if (!array[i])
310.       return false;
311. return true;

```

Ontology-Based Policy Specification and Management

Wolfgang Nejdl¹, Daniel Olmedilla¹,
Marianne Winslett², and Charles C. Zhang²

¹ L3S Research Center and University of Hannover, Germany
{nejdl, olmedilla}@l3s.de

² Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
{winslett, cczhang}@cs.uiuc.edu

Abstract. The World Wide Web makes it easy to share information and resources, but offers few ways to limit the manner in which these resources are shared. The specification and automated enforcement of security-related policies offer promise as a way of providing controlled sharing, but few tools are available to assist in policy specification and management, especially in an open system such as the Web, where resource providers and users are often strangers to one another and exact and correct specification of policies will be crucial. In this paper, we propose the use of ontologies to simplify the tasks of policy specification and administration, discuss how to represent policy inheritance and composition based on credential ontologies, formalize these representations and the according constraints in Frame-Logic, and present POLICYTAB, a prototype implementation of our proposed scheme as a Protégé plug-in to support policy specification.

1 Introduction

Open distributed environments like the World Wide Web offer easy sharing of information, but provide few options for the protection of sensitive information and other sensitive resources, such as Web Services. Proposed approaches to controlling access to Web resources include XACML [4], SAML [5], WS-Trust [3] and Liberty-Alliance[1]. All of these approaches to trust management rely on the use of vocabularies that are shared among all the parties involved, and declarative policies that describe who is allowed to do what. Some of these approaches also recognize that trust on the Web and in any other system where resources are shared across organizational boundaries must be *bilateral*.

Specifically, the Semantic Web provides an environment where parties may make connections and interact without being previously known to each other. In many cases, before any meaningful interaction starts, a certain level of trust must be established from scratch. Generally, trust is established through exchange of information between the two parties. Since neither party is known to the other, this trust establishment process should be bi-directional: both parties may have sensitive information that they are reluctant to disclose until the other party has proved to be trustworthy at a certain level. As there are more service providers emerging on the Web every day, and people are performing more

sensitive transactions (e.g., financial and health services) via the Internet, this need for building mutual trust will become more common.

To make controlled sharing of resources easy in such an environment, parties will need software that automates the process of iteratively establishing bilateral trust based on the parties' access control policies, i.e., *trust negotiation* software. Trust negotiation differs from traditional identity-based access control and information release systems mainly in the following aspects:

1. Trust between two strangers is established based on parties' properties, which are proved through disclosure of digital credentials.
2. Every party can define access control and release policies (*policies*, for short) to control outsiders' access to their sensitive resources. These resources can include services accessible over the Internet, documents and other data, roles in role-based access control systems, credentials, policies, and capabilities in capability-based systems. The policies describe what properties a party must demonstrate (e.g., ownership of a driver's license issued by the State of Illinois) in order to gain access to a resource.
3. Two parties establish trust directly without involving trusted third parties, other than credential issuers. Since both parties have policies, trust negotiation is appropriate for deployment in a peer-to-peer architecture such as the Semantic Web, where a client and server are treated equally. Instead of a one-shot authorization and authentication, trust is established incrementally through a sequence of bilateral credential disclosures.

A trust negotiation process is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials $(C_1, \dots, C_k, R)$, where R is the resource to which access was originally requested, such that when credential C_i is disclosed, its policy has been satisfied by credentials disclosed earlier in the sequence or to determine that no such credential disclosure sequence exists.

The use of declarative policies and the automation of the process of satisfying them in the context of such a *trust negotiation process* seem to be the most promising approach to providing controlled access to resources on the Web. However, this approach opens up a new and pressing question: what confidence can we have that our policies are correct? Because the policies will be enforced automatically, errors in their specification or implementation will allow outsiders to gain inappropriate access to our resources, possibly inflicting huge and costly damages. Unfortunately, real-world policies [10] tend to be as complex as any piece of software when written down in detail; getting a policy right is as hard as getting a piece of software correct, and maintaining a large number of them is only harder.

In this paper, we take an ontology-based approach to address this problem. Section 2 discusses the use of ontologies for providing abstraction and structuring for policy specification, and further formalizes these concepts and constraints in Frame-Logic / F-Logic [17]. Section 3 describes our proof-of-concept implementation, POLICYTAB, a Protégé [2] plug-in to support policy specification. We discuss related work in section 4 and give future research directions and conclusions in section 5.

2 Using Ontologies to Ease Policy Specification and Management

Ontology-based structuring and abstraction help maintain complex software, and so do they with complex sets of policies. In the context of the Semantic Web, ontologies provide formal specification of concepts and their interrelationships, and play an essential role in complex web service environments [7], semantics-based search engines [13] and digital libraries [21].

One important purpose of these formal specifications is sharing of knowledge between independent entities. In the context of trust negotiation, we want to share information about credentials and their attributes, which is needed for establishing trust between negotiating parties. Figure 1 shows a simple example ontology for credential IDs.

Each credential class can contain its own attributes; e.g., a Cisco Employee ID credential has three attributes: name, rank and department. Trust negotiation is attributed-based and builds on the assumption that each of these attributes can be protected and disclosed separately. While in some approaches (e.g. with X.509 certificates) credentials and their attributes are signed together as a whole by the credential issuer, in this paper we will rely on cryptographic techniques such as [19] which allow us to disclose credentials with different granularities, hiding attributes not relevant to a given policy.

In trust negotiation, a party's security policies consist of constraints that the other party has to satisfy; e.g. it has to produce a proof that it owns a certain credential, and that one of the credential attributes has to be within a certain range. Assuming a casino requires any customer's age to be over 21 and requires a state ID to testify that, the policy for its `admits` service can be represented as the following logic program, which uses a simplified version of the PEERTRUST [18, 15] policy language:

Casino:

```
allowedInCasino(Requester) ←  
  type(CredentialIdentifier, "State_Id") @ Issuer @ Requester,  
  issuedFor(CredentialIdentifier, Requester) @ Issuer @ Requester,  
  age(CredentialIdentifier, Age) @ Issuer @ Requester,  
  Age > 21.
```

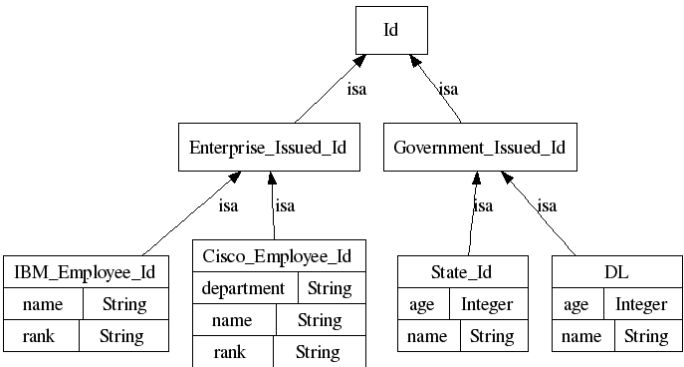


Fig. 1. Simple ID Credential Ontology

In this example, the first two statements in the body of the rule require the requester to prove that he owns a credential of type `State_Id` issued by `Issuer`¹. If the requester proves that he has it (notice that information about attributes has not been disclosed so far, except for the `issuedFor` attribute), the casino asks for the value of the attribute `age` in the presented credential. Then it verifies whether the requester's age is over 21 and, if successful, admits the requester into the casino.

2.1 Sharing Policies for Common Attributes

Often, credentials share common attributes, and these attributes might share the same policies. Figure 1 shows an example of a simple credential hierarchy, where the concrete credential classes used are depicted in the leaves of the hierarchy. The upper part of the hierarchy represents the different abstract classes: the root represents any `Id`, which is partitioned into different subclasses according to the issuer of the credential, distinguished between `Government_Issued` and `Enterprise_Issued` IDs. The leaf nodes represent concrete classes which contain the attributes such as `name`, `age`, and `rank`.

This somewhat degenerated hierarchy however does not yet allow for policy re-use. For this we have to exploit attribute inheritance. In our example, all leaf nodes share the `Name` attribute, which therefore can be moved up to the root class `Id`. We are now able to specify common policies for the `Name` attribute at the `Id` level. Similarly, we will move `Rank` up so that it becomes an attribute of `Enterprise_Issued_Id`, and `Age` an attribute of `Government_Issued_Id`. A subclass automatically inherits its superclass's attributes, which might be local or inherited from the superclass's superclass. In the following, we will use Frame-Logic / F-Logic [17] to represent these constraints. So, in the context of F-Logic, we use *type inheritance* (also *structural inheritance*) to represent this constraint, which is defined as

$$\text{If } I \models p[\text{mthd}@q_1, \dots, q_k \approx > s] \text{ and } I \models r :: p \text{ then } I \models r[\text{mthd}@q_1, \dots, q_k \approx > s]$$

where the symbol $\approx >$ denotes either $\Rightarrow$ or $\Rightarrow>$, I is any F-structure and $r :: p$ represents the fact that “ r is *subclass of* p ”.

This leads to the refined ontology as described in figure 2, where each leaf node has the same set of attributes as in figure 1, but inherits them from higher levels. This makes it possible to specify shared policies for these shared attributes, similar to method inheritance in object oriented programming languages.

2.2 Composing and Overriding Policies

Now, given such credential ontologies, we can specify security policies at different levels. Being able to inherit and compose these security policies simplifies policy maintenance, though of course we have to distinguish between the case where we compose

¹ As an extra hint, in the PEERTRUST language, for a statement such that “ $lit_i @ Authority$ ”, *Authority* specifies the peer who is responsible for evaluating lit_i or has the authority to evaluate lit_i . In addition, *Authority* can be a nested term containing a sequence of authorities, which are then evaluated starting at the outermost layer.

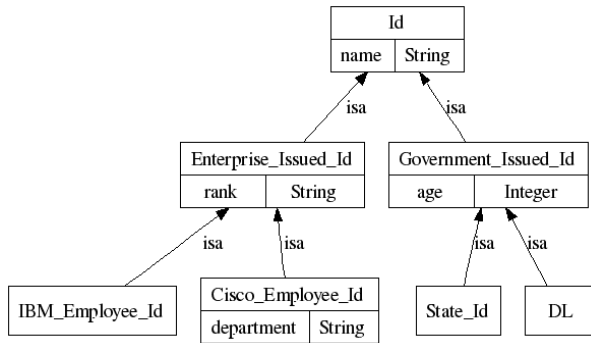


Fig. 2. Refined ID Credential Ontology

inherited and local policies and the case where the local policy specified for an attribute of a specific class overrides the policy inherited from a superclass. In this paper we will describe *mandatory policies* and *default policies*.

To model a policy in F-Logic, we have the following signature declaration for the class `policy`

```

policy [
    name => string,
    value => string,
    type => string
]

```

where `name` is the unique name of the policy, `value` is the text that describes the policy (expressed in a suitable policy language) and `type` describes if the policy is default or mandatory. To express the constraint that `type` can only contain the strings “Default” or “Mandatory” and only one of them at the same time, we define the following integrity constraint

$$false \leftarrow C : policy, C[type \rightarrow V], not\ V = "Default", not\ V = "Mandatory"$$

Moreover, we want to assure that any class in our knowledge base has the possibility to define policies. Therefore we need to declare a meta class called `metaClass` from which all the classes will be an instance.

```

metaClass [
    policySlot ==> policy,
    overallPolicy ==> policy
]

```

This meta class contains a property `policySlot` whose value is a set of policies (possibly empty) attached to the class and a property `overallPolicy` whose value

is the set of policies (possibly empty) of all the policies, directly attached to the class and inherited from superclasses, that apply to this class. The derivation rule

$$C2[overallPolicy \rightarrow P] \leftarrow C2 :: C1, C1[policySlot \rightarrow P] \quad (1)$$

assures that any policy in a direct superclass is inherited to the subclass. We further have

$$C[overallPolicy \rightarrow P] \leftarrow C[policySlot \rightarrow P] \quad (2)$$

to add the policies attached to the the current class ($C :: C$ is not true in F-Logic²).

Finally, in order to assure that any class in the knowledge base (except the policy class defined above) will be an instance of `metaClass` we need the following derivation rule

$$C : metaClass \leftarrow not C = policy, not C : policy \quad (3)$$

Figure 4 depicts the hierarchy of classes and instances in our driver license example.

Mandatory Policies. Mandatory policies are used when we want to mandate that policies of a higher level are always enforced at lower levels. Assume the ontology depicted in figure 3 and that we want to hire an experienced driver to accomplish a certain highly classified and challenging task. Before we show the details of the task to an interested candidate, we want the candidate to present a driver's license, which can be proved to satisfy the following mandatory policies as specified at the different levels:

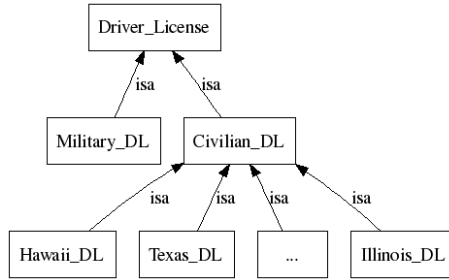


Fig. 3. Driver License Ontology

At the `Driver_License` level, we enforce generic requirements for driver licenses; e.g., a driver license has to be signed by a federally authorized certificate authority and must not have expired.

At the `Civilian_DL` level, we require that the driver license is non-commercial, assuming commercial drivers may have a conflict of interests in the intended task.

At the `Illinois_DL` level, we require that the category of the driver license is not *F*, assuming *F* licenses are for farm vehicles only. At the `Military_DL` level, we

² In the F-Logic notation, the operator $C1 :: C2$ represents “*C1* is subclass of *C2*” while $C1 : C2$ means “*C1* is an instance of *C2*”

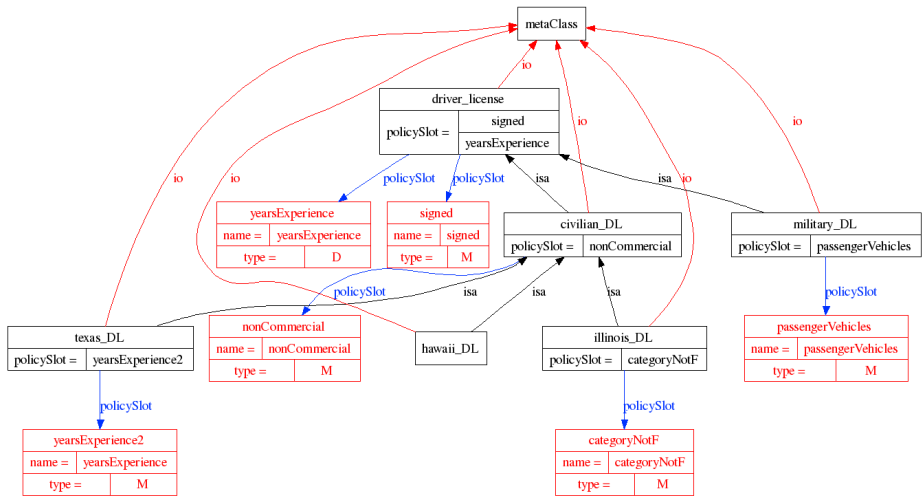


Fig. 4. Driver License Knowledge Base

can specify policies such as “the driver license must be for land passenger vehicles” as opposed to fighter planes or submarines.

So for an Illinois driver, the overall policy is: must hold a valid driver license, as qualified by the policy at the Driver_License level; must hold a non-commercial driver license, as required by the Civilian_DL policy; and the driver license must not be for farm vehicles only. The advantage of using mandatory policies here is twofold: first, shared policies such as the generic driver license requirements are only specified once at a higher level, which means a more compact set of policies; second, it gives a cleaner and more intuitive logical structure to policies, which makes the policies easier to specify and manage.

Default Policies. Let us now assume that all driver licenses include the specification of driving experience, expressed in years of driving. Suppose that a specific task requires the following policy: in most cases, 4 years’ driving experience is required; however, if the driver comes from Texas, he/she needs only 3 years’ experience (assuming it is harder to get a driver’s license in Texas).

To simplify the specification of this policy, we can use the default policy construct. A superclass’s default policy is inherited and enforced by a subclass if and only if the child does not have a corresponding (overriding) policy. In our example, we can specify at the Driver_License level that the driving age has to be at least 4 years; then at the Texas_DL level, specify an overriding policy that the driving age has to be at least 3 years.

It is of interest to note that the same result can be achieved here without using default policies: we can move the shared 4-year mandatory policy down to *every* concrete driver license class except Texas_DL, where we require 3 years. However, the power of policy sharing is lost.

To summarize, on one hand, mandatory policies must be enforced at lower levels in the hierarchy, that is, they can not be overridden. On the other hand, default policies are inheritable, but they can be overridden at lower levels. In order to formalize this, we assume that if two policies have the same value in the property name, the most specific one overrides the other one. Taking that into account, we need to refine equation (1) in a way that overridden policies are not included in the overall policy. The derivation rule would be

$$C2[overallPolicy \rightarrow P] \leftarrow C2 :: C1, C1[policySlot \rightarrow P, \quad (4) \\ not(C2[policySlot \rightarrow P2, P2[name \rightarrow N], P[name \rightarrow N])]$$

Finally, only default policies can be overridden and therefore we need the following integrity constraint to avoid that mandatory policies are overridden

$$false \leftarrow C2 : C1, C1[policySlot \rightarrow P1], C2[policySlot \rightarrow P2], \quad (5) \\ P1[name \rightarrow N, type \rightarrow "Mandatory"], P2[name \rightarrow N]$$

3 POLICYTAB: Making Protégé a Policy Management Tool

To support policy specification as discussed in the previous sections, we have implemented POLICYTAB (available at <http://www.13s.de/peertrust/>, a plug-in for the ontology editor Protégé. We chose Protégé because it is widely used and is extensible by means of plug-ins. The POLICYTAB plug-in adds a new tab to the main window of Protégé (see figure 5 for an example), which consists of the following elements:

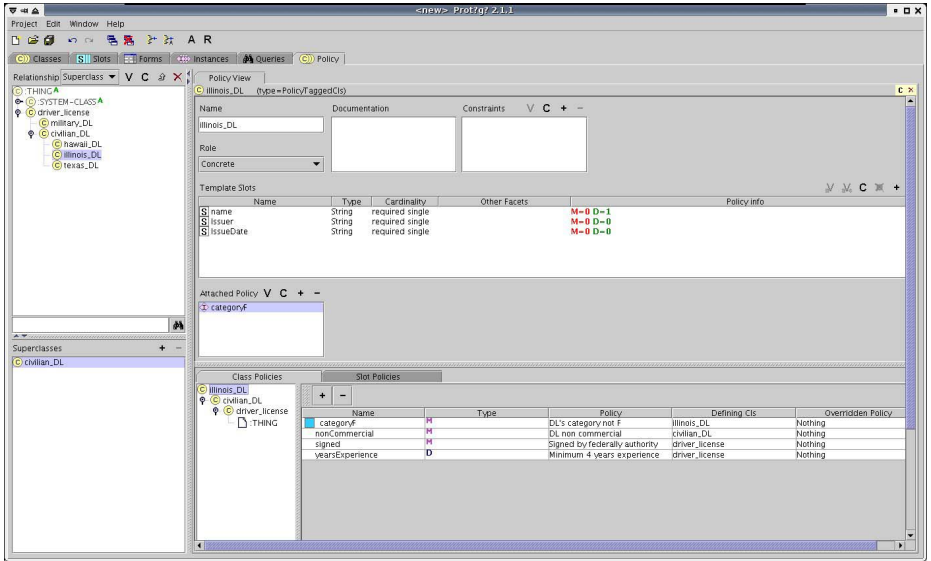


Fig. 5. Screenshot of the POLICYTAB plug-in

- *The Class Relationship Pane.* This panel on the upper left corner of the window displays the existing classes in the knowledge base as a tree.
- *The Superclass Pane.* This panel on the lower left corner shows the superclasses of the class currently selected in the Class Relationship Pane.
- *The PolicyView Form.* This form occupying the upper right part of the window contains the information of the class currently selected in the Class Relationship Pane.
- *The Associated Policies Pane.* This pane at the lower right part of the window displays the policies attached to the current selected class or to the current selected slot.

We describe the PolicyView Form and the Associated Policies Pane in more detail in the following sections.

3.1 The PolicyView Form

The PolicyView Form contains the information related to the currently selected class (see figure 5). Each class has the usual properties available in Protégé

- *Name:* the name of the class
- *Documentation:* extra comments and explanations
- *Role:* describes if the class is concrete or abstract
- *Constraints:* specify constraints to the class
- *Template Slots:* show the different properties of the class

In Protégé, a class's properties are called *slots*. To add a slot to the current class, click the + icon for the `Template Slots` table, a slot creation dialog will pop up (see figure 6), where you can specify the new slot's name, type, etc. POLICYTAB automatically checks name conflicts, and does not permit the specification of a slot with the same name as in one of its superclasses.

3.2 The Associated Policies Pane

The Associated Policies Pane displays the policies that apply to the currently selected class, i.e. the overall set of policies that should be satisfied by a requester in order to get access to the resource represented by the class. A policy's type can be either `mandatory` or `default`. Overriding of policies is done by explicitly selecting which class to be overridden (the combobox `Overridden Policy` shows only overridable policies or `Nothing` as valid values)³. POLICYTAB's automatic overridability checking prevents the user from unintentionally overriding a mandatory policy and hence reduces policy specification errors.

This pane contains two different tabs: `Class Policies` and `Slot Policies` (see figure 7). Class policies are specified to protect the whole class and correspond

³ As to our knowledge there not exists yet an F-Logic inference engine integrated in Protégé, our current implementation “hard-codes” in the plug-in the inference rules presented in the paper.

Issuer (type=PolicyTaggedSlot)

Name

Issuer

Value Type

String

Documentation

The issuer of the Driver License

Cardinality

☒ required

at least

1

☐ multiple

at most

1

Template Value

Default

Domain

driver_license

Minimum

Maximum

Inverse Slot

Attached Policy

protectIssuer

Fig. 6. New Slot Creation

Class Policies

Slot Policies

Illinois_DL

civilian_DL

driver_license

THING

Name	Type	Policy	Defining Cls	Overridden Policy
categoryF	M	DL's category not F	Illinois_DL	Nothing
nonCommercial	M	DL non commercial	civilian_DL	Nothing
signed	M	Signed by federally authority	driver_license	Nothing
yearsExperience	D	Minimum 4 years experience	driver_license	Nothing

Fig. 7. View of overall policy applicable to a class

to the concepts described in section 2. The slot policies are protectors for each specific property, and have a finer-grained protection. A requester has to satisfy the relevant slot policies in addition to the class policies in order to access a certain property of the class. This is crucial in Trust negotiation as the process relies on disclosure of party’s properties, not necessarily whole credentials. The tab Slot Policies displays the policies that apply to the slot currently selected in the PolicyView form. Both class policies and slot policies are inherited by the subclasses in the hierarchy.

Once a class is selected in the Class Relationship Pane, the Associated Policies Pane shows this class’s inheritance hierarchy as well as its class level policies. For each single policy it displays the name, the type (mandatory or default), the string with the policy description, the class where that policy is defined and the class whose corresponding policy is overridden (or Nothing if there isn’t any). Automatic resolution of overriding and inheritance gives the user a clear view of the current class’s effective policies, and showing the original defining class of the inherited policies in addition to the policy tree helps the user understand the policy composition hierarchy and capture the intuitions and implications behind it.

4 Related Work

Recent work in the context of the Semantic Web has focused on how to describe security requirements. KAOs and Rei policy languages [16, 22] investigate the use of ontologies for modeling speech acts, objects, and access types necessary for specifying security policies on the Semantic Web. Hierarchies of annotations to describe capabilities and requirements of providers and requesting agents in the context of Web Services are introduced in [11]. Those annotations are used during the matchmaking process to decide if requester and provider share similar security characteristics and if they are compatible.

Ontologies have also been discussed in the context of digital libraries for concepts and credentials [6]. An approach called “most specific authorization” is used for conflict resolution. It states that policies specified on specific elements prevail over policies specified on more general ones. In this paper we explore complementary uses of ontologies for trust negotiation, through which we target iterative trust establishment between strangers and the dynamic exchange of credentials during an iterative trust negotiation process that can be declaratively expressed and implemented. Work done in [9] defines *abstractions* of credentials and services. Those abstractions allow a service provider to request for example a credit card without specifically asking for each kind of credit card that it accepts. We add to this work in the context of policy specification the concept of *mandatory* and *default* policies.

Ontology-based policy composition and conflict resolving have also been been discussed in previous work. Policy inheritance is done by *implication* in [12], but it does not provide any fine-grained overriding mechanism based on class levels. *Default properties* are discussed in [14], short of generalizing the idea to policies. The approaches closest to our default and mandatory policy constructs are the *weak* and *strong* authorizations in [8], where a strong rule always overrides a weak rule, and SPL in [20], which forces the security administrator to combine policies into a structure that precludes conflicts. Compared to these approaches, we find ours particularly simple and intuitive, while its expressiveness well serves general trust negotiation needs.

5 Conclusions and Future Research Directions

Ontologies can provide important supplemental information to trust negotiation agents both at compile time to simplify policy management and composition. This paper has explored some important benefits of using ontologies.

For compile time usage, ontologies with their possibility of sharing policies for common attributes provide an important way for structuring available policies. In this context we propose two useful strategies to compose and override these policies, building upon the notions of mandatory and default policies, and formalize the constraints corresponding to these kinds of policies using F-Logic. We also present a prototype implementation, POLICYTAB, which shows that the proposed policy specification mechanism is implementable and effective.

Future work will investigate multiple inheritance and resolution of conflicting policies in ontology hierarchies, and whether we need disjunction for composing these poli-

cies. We are also working on a closer integration into our PEERTRUST system, with suitable import/export facilities to and from POLICYTAB. Finally, an interesting research area to consider in the future is policy validation, i.e. whether the final ontologies plus policy rules are consistent and correct with respect to a set of background constraints.

Acknowledgments

The authors thank Rubén Lara for useful discussions and help in the modeling with F-Logic and the anonymous reviewers for their useful comments. The research of Nejdl and Olmedilla was partially supported by the projects ELENA (<http://www.elena-project.org>, IST-2001-37264) and REVERSE (<http://reverse.net>, IST-506779). The research of Winslett was supported by DARPA (N66001-01-1-8908), the National Science Foundation (CCR-0325951, IIS-0331707) and The Regents of the University of California.

References

1. *Liberty Alliance Project*. <http://www.projectliberty.org/about/whitepapers.php>.
2. *The Protégé Ontology Editor and Knowledge Acquisition System*. <http://protege.stanford.edu/>.
3. *Web Services Trust Language (WS-Trust) Specification*. <http://www-106.ibm.com/developerworks/library/specification/ws-trust/>.
4. Xacml 1.0 specification <http://xml.coverpages.org/ni2003-02-11-a.html>.
5. Assertions and protocol for the oasis security assertion markup language (saml); committee specification 01, 2002.
6. N. R. Adam, V. Atluri, E. Bertino, and E. Ferrari. A content-based authorization model for digital libraries. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):296–315, 2002.
7. A. Ankolekar. Daml-s: Semantic markup for web services.
8. E. Bertino, S. Jojodia, and P. Samarati. Supporting multiple access control policies in database systems. In *IEEE Symposium on Security and Privacy*, pages 94–109, Oakland, CA, 1996. IEEE Computer Society Press.
9. P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security*, Athens, Nov. 2000.
10. Cassandra policy for national ehr in england. <http://www.cl.cam.ac.uk/users/mywyb2/publications/ehrpolicy.pdf>.
11. G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
12. W. Emayr, F. Kastner, G. Pernul, S. Preishuber, and A. Tjoa. Authorization and access control in iro-db.
13. M. Erdmann and R. Studer. How to structure and access xml documents with ontologies. *Data and Knowledge Engineering*, 36(3), 2001.
14. R. Fikes, D. McGuinness, J. Rice, G. Frank, Y. Sun, and Z. Qing. Distributed repositories of highly expressive reusable knowledge, 1999.
15. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st First European Semantic Web Symposium*, Heraklion, Greece, May 2004.

16. L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
17. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.
18. W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: automated trust negotiation for peers on the semantic web. In *Workshop on Secure Data Management in a Connected World (SDM'04)*, Toronto, Aug. 2004.
19. P. Persiano and I. Visconti. User privacy issues regarding certificates and the tls protocol. In *Conference on Computer and Communications Security*, Athens, Nov. 2000.
20. C. Ribeiro and P. Guedes. Spl: An access control language for security policies with complex constraints, 1999.
21. S. B. Shum, E. Motta, and J. Domingue. Scholonto: an ontology-based digital library server for research documents and discourse. *Int. J. on Digital Libraries*, 3(3):237–248, 2000.
22. G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei and Ponder. In *2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.

Web Explanations for Semantic Heterogeneity Discovery

Pavel Shvaiko¹, Fausto Giunchiglia¹,
Paulo Pinheiro da Silva², and Deborah L. McGuinness²

¹ University of Trento, Povo, Trento, Italy
{pavel, fausto}@dit.unitn.it

² Stanford University, Stanford, USA
{dlm, pp}@ksl.stanford.edu

Abstract. Managing semantic heterogeneity is a complex task. One solution involves matching like terms to each other. We view *Match* as an operator that takes two graph-like structures (e.g., concept hierarchies or ontologies) and returns a mapping between the nodes of the graphs that correspond semantically to each other. While some state of the art matching systems may produce effective mappings, these mappings may not be intuitively obvious to human users. In order for users to *trust* the mappings, and thus, use them, they need information about them (e.g., they need access to the sources that were used to determine semantic correspondences between terms). In this paper we describe how a matching system can explain its answers using the Inference Web (IW) infrastructure thus making the matching process *transparent*. The proposed solution is based on the assumption that mappings are computed by logical reasoning. There, *S-Match*, a *semantic matching* system, produces proofs and explanations for mappings it has discovered.

1 Introduction

The progress of information and communication technologies, and in particular of the Web, has made available a huge amount of disparate information. The number of different information resources is growing significantly, and therefore, the problem of managing semantic heterogeneity is increasing, see, for instance, [34]. Many solutions to this problem include identifying terms in one information source that match terms in another information source. The applications can be viewed to refer to graph-like structures containing terms and their inter-relationships. These might be database schemas, concept hierarchies, ontologies.

We view *Match* as one of the key operators for enabling semantic applications since it takes two graph-like structures and produces a mapping between the nodes of the graphs that correspond semantically to each other. Matching, however, requires explanations because mappings between terms are not always intuitively obvious to human users. In fact, if Semantic Web users are going to trust the fact that two terms may have the same meaning, then they need to understand the reasons leading a matching system to produce such a result. Expla-

nations are also useful, when matching applications with hundreds or thousands of nodes, since in these cases automatic matching solutions will find a number of plausible mappings, thus some human effort (e.g., database/knowledge base administrators who need to perform some rationalization of the mapping suggestions) is inevitable.

We focus on *semantic matching* as proposed in [11], and implemented within the *S-Match* [12] system. This matching solution is based on the assumption that mappings are computed by logical reasoning. We have extended *S-Match* to use the Inference Web infrastructure [22] and its Proof Markup Language (PML)[5]. Thus, with the help of IW tools and exploiting PML properties, meaningful fragments of the *S-Match* proofs can be loaded on demand. Users can browse an entire proof or they can restrict their view and refer only to specific, relevant parts of proofs. They can ask for provenance information related to proof elements (e.g., the origin of the terms in the proofs, the authors of the ontologies). Therefore, they can make informed decisions about the mappings.

The rest of this paper proceeds as follows. In Section 2, via an example, we discuss semantic matching approach as implemented within the *S-Match* system. In Section 3 we describe the Inference Web infrastructure. Using the example introduced in Section 2, in Section 4, we present how the Inference Web explanations increase user understanding of the *S-Match* answers. Section 5 presents an experimental study that addresses the scaling of the explanation techniques. Section 6 discusses the related work and Section 7 summarizes the contributions of the paper.

2 Semantic Matching

The semantic matching approach is based on two ideas. The first idea is that we calculate mappings between schema/ontology elements by computing *semantic relations* (e.g., equivalence, more general), instead of computing coefficient rating match quality in the $[0,1]$ range, as is the case in other approaches, see, for example, [9, 17, 19]. The second idea is that we determine semantic relations by analyzing the *meaning* (concepts, not labels) which is codified in the elements and the structures of schemas/ontologies. In particular, labels at nodes, written in natural language, are translated into propositional formulas which explicitly codify the label's intended meaning. This allows us to translate the matching problem into a propositional unsatisfiability problem.

We call the *concept of a label* the propositional formula which stands for the set of documents that one would classify under a label it encodes. We call the *concept at a node* the propositional formula which represents the set of documents which one would classify under a node, given that it has a certain label and that it is in a certain position in a tree.

Possible semantic relations that *S-Match* can discover between the concepts of nodes of the two schemas/ontologies are: *equivalence* ($=$); *more general* ($\sqsupseteq$); *less general* ($\sqsubseteq$); *disjointness* ($\perp$). When none of the relations holds, the special *idk* (I dont know) relation is returned. The relations are ordered according to

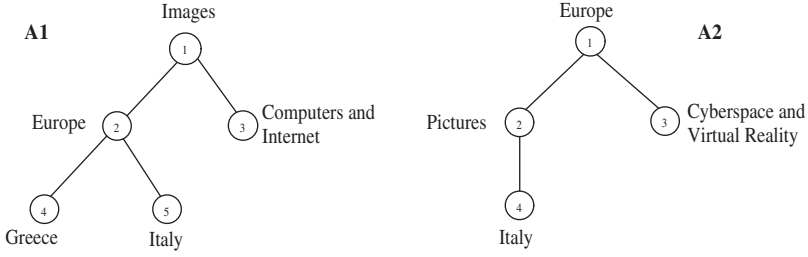


Fig. 1. Simple catalog matching problem

decreasing binding strength, i.e., from the strongest (=) to the weakest (*idk*), with more general and less general relations having equal binding power. The semantics of the above relations are the obvious set-theoretic semantics.

A *mapping element* is a 4-tuple $\langle ID_{ij}, n1_i, n2_j, R \rangle$, $i=1,\dots,N1$; $j=1,\dots,N2$; where ID_{ij} is a unique identifier of the given mapping element; $n1_i$ is the i -th node of the first graph, $N1$ is the number of nodes in the first graph; $n2_j$ is the j -th node of the second graph, $N2$ is the number of nodes in the second graph; and R specifies a semantic relation which may hold between the concepts of nodes $n1_i$ and $n2_j$. *Semantic matching* can then be defined as the following problem: given two graphs $G1$, $G2$ compute the $N1 \times N2$ mapping elements $\langle ID_{ij}, n1_i, n2_j, R' \rangle$, with $n1_i \in G1$, $i=1,\dots,N1$, $n2_j \in G2$, $j=1,\dots,N2$ and R' the strongest semantic relation holding between the concepts of nodes $n1_i, n2_j$. The strongest semantic relation always exists since, when holding together, more general and less general are equivalent to equivalence. *S-Match* is *schema-based*, and therefore, it considers only intentional information, not instance data. In the current version it is limited to the tree-like structures, e.g., taxonomies, or simple XML schemas with attributes.

We concentrate on class matching and motivate the problem by the simple catalog matching example shown in Figure 1. Suppose an agent wants to exchange/search for documents with another agent. The documents of both agents are stored in catalogs according to class hierarchies A1 and A2 respectively. *S-Match* takes as input these hierarchies and computes as output a set of mapping elements in four macro steps. The first two steps represent the pre-processing phase, while the third and the fourth steps correspond to *element level* and *structure level* matching respectively.

Step 1. For all labels L in the two trees, compute *concepts of labels*. We think of labels at nodes as concise descriptions of the data that is stored under the nodes. We compute the meaning of a label at a node by taking as input a label, by analyzing its (real world) semantics, and by returning as output a concept of the label, C_L . For example, when we write $C_{Pictures}$ we mean the concept describing all the documents which are (about) pictures. Notice, that by writing $C_{Pictures}$ we move from the natural language ambiguous label *Pictures* to the concept $C_{Pictures}$, which the given label denotes.

Technically, concepts at labels are encoded as propositional logical formulas, where atomic formulas are WordNet [25] *senses* (possible meanings) of single words, and complex formulas are obtained by combining atomic concepts using the connectives of set theory and set-theoretic semantics. For example, $C_{Pictures} = \langle picture, senses_{WN\#11} \rangle$, where $senses_{WN\#11}$ is taken to be disjunction of the eleven senses that WordNet attaches to *pictures*. The process of extraction of logical formulas from natural language labels is described in detail in [20].

Step 2. For all nodes N in the two trees, compute *concepts of nodes*.

In this step we analyze the meaning of the positions that the labels at nodes have in a tree. By doing this we *extend* concepts of labels to *concepts of nodes*, C_N . This is required to capture the knowledge residing in the structure of a graph, namely the context in which the given concept at label occurs. For example, in A2, when we write C_2 we mean the concept describing all the documents which are (about) pictures and which are also about Europe. Thus, in Figure 1, following the classification link semantics, which is an *access criterion* [16], the logical formula for a concept at node is defined as a conjunction of concepts of labels located above the given node, including the node itself, for example, in A2, $C_2 = C_{Europe} \sqcap C_{Pictures}$.

Step 3. For all pairs of labels in the two trees, compute *relations among concepts of labels*. Relations between concepts of labels are computed with the help of a library of element level semantic matchers [13]. These matchers take as input two concepts of labels and produce as output a semantic relation between them. For example, in Figure 1, C_{Images} can be found equivalent to $C_{Pictures}$. In fact, according to WordNet, *images* and *pictures* are synonyms. Notice that in WordNet *pictures* has 11 senses and *images* has 8 senses. We use some sense filtering techniques to discard the irrelevant senses for the given context, see [20] for details.

Step 4. For all pairs of nodes in the two trees, compute *relations among concepts of nodes*. *S-Match* decomposes the tree matching problem into the set of node matching problems. Then, each node matching problem, namely pairs of nodes with possible relations between them, is translated into a propositional formula. The semantic relations are translated into propositional connectives as follows: equivalence into equivalence, more general and less general into implication, and disjointness into negation of the conjunction. As from [11], we have to prove that the following formula:

$$Axioms \longrightarrow rel(C1_i, C2_j) \quad (1)$$

is valid, namely that it is *true* for all the truth assignments of all the propositional variables occurring in it. In (1), *Axioms* is the conjunction of all the relations between concepts of labels computed in step 3. $C1_i$ is the propositional formula encoding concept at node i in tree 1, $C2_j$ is the propositional formula encoding concept at node j in tree 2, *rel* is the semantic relation that we want to prove holding between $C1_i$ and $C2_j$.

In the Inference Web, proof and explanation documents are formatted in PML and are composed of PML *node sets*. Each node set represents a step in a proof whose conclusion is justified by any of a set of inference steps associated with a node set. Also, node sets are subclasses of the W3C's OWL Class [33] and they are the building blocks of OWL documents describing proofs and explanations for application answers published on the Web.

The *IW Browser* is used to present proofs and explanations. Exploiting PML properties, meaningful fragments of the *S-Match* proofs can be loaded on demand. Users can browse an entire proof or they can limit their view and refer only to specific, relevant parts of proofs since each node set has its own URI that can be used as an entry point for proofs and proof fragments.

4 Producing Explanations

A default explanation of mappings the *S-Match* system produces is a short, natural language, high-level explanation without any technical details. It is designed to be intuitive and understandable by ordinary users.

Let us recall the catalog matching example. Suppose that agent A2 is interested in knowing why *S-Match* suggested a set of documents stored under the node with label *Europe* in A1 as the result to the query - "*find european pictures*". A default explanation is presented in Figure 3.

From the explanation in Figure 3, users may learn that *Images* in A1 and *Pictures* in A2 are equivalent words, i.e., they can be interchanged, in the context of the query. Also, users may learn that *Europe* in A1 denotes the same concept as *Europe* (*European*) in A2. Therefore, they can conclude that *Images of Europe* means the same thing as *European Pictures*. Future work includes optional pruning of statements containing information that two concepts are identical.

However, users may not be satisfied with this level of explanation. Let us therefore discuss how they can investigate the details of the matching process by exploiting more verbose explanations. We have implemented two kinds of verbose explanations: background knowledge explanations and logical reasoning explanations. Let us consider them in turn.

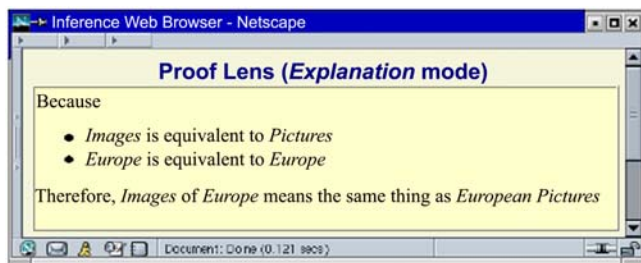


Fig. 3. An explanation in English

4.1 Explaining Background Knowledge

Suppose that the agent wants to see the sources of background knowledge used in order to determine the mapping. For example, which applications, publications, other sources, have been used to determine that *Images* is equivalent to *Pictures*. Figure 4 presents the source metadata for the default explanation of Figure 3.

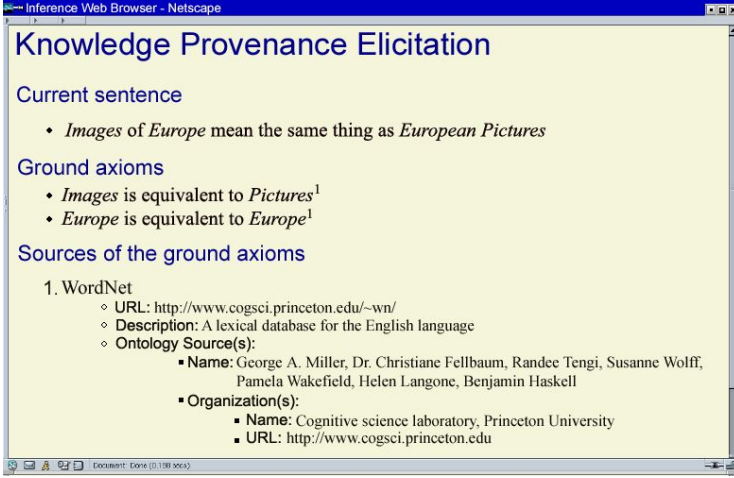


Fig. 4. Source metadata information

In this case, both (all) the ground sentences used in the *S-Match* proof came from WordNet. Using WordNet, *S-Match* learned that the first sense of the word pictures is a synonym to the second sense of the word images. Therefore, *S-Match* can conclude that these two words are equivalent words in the context of the answer. The meta information about WordNet from the IW Registry is also presented in Figure 4 as *sources of the ground axioms*. Further examples of explanations include: providing meta information about the *S-Match* library of element-level matchers [13], i.e., those which are based not only on WordNet, the order in which the matchers are used, and so on.

4.2 Explaining Logical Reasoning

A more complex explanation may be required if users are not familiar with or do not trust inference engine(s) embedded in a matching system. As the Web starts to rely more on information manipulations (instead of simply information retrieval), explanations of embedded manipulation/inference engines become more important. In the current version of *S-Match*, a propositional satisfiability engine is used, more precisely, the Davis-Putnam-Longemann-Loveland (DPLL) procedure [6, 7] as implemented in JSAT [2].

The task of a SAT engine is to find an assignment $\mu \in \{\top, \perp\}$ to atoms of a propositional formula φ such that φ evaluates to *true*. φ is *satisfiable* iff $\mu \models \varphi$

for some μ . If μ does not exist, φ is *unsatisfiable*. A *literal* is a propositional atom, or its negation. A *clause* is a disjunction of one or more literals. φ is said to be in conjunctive normal form (CNF) iff it is a conjunction of disjunctions of literals. The basic DPLL procedure recursively implements the three rules: *unit resolution*, *pure literal* and *split* [6, 7].

Let l be a literal and φ a propositional formula in CNF. A clause is called a *unit clause* iff it has exactly one unassigned literal. *Unit resolution* is an application of *resolution* to a unit clause.

$$\text{unit resolution} : \frac{\varphi \wedge \{l\}}{\varphi[l \mid \top]}$$

l is called a *pure literal* in φ iff it occurs in φ only positively or negatively. *Pure literal* removes all clauses in which pure literals occur.

$$\text{pure literal} : \frac{\varphi}{\varphi[l \mid \top]}$$

Split rule performs branching first on truth values of literals then on their false values, iff the above two rules (deterministic choices) cannot be applied.

$$\text{split} : \frac{\varphi}{\varphi[l \mid \top] \quad \varphi[l \mid \perp]}$$

Usually performance of SAT engines is not a concern for producing proofs. Thus, we have modified the JSAT DPLL procedure and enabled it to generate proofs. Next, we discuss the IW proofs and explanations of the *unit resolution* rule in detail. In the current version, the *pure literal* and *split* rules are explained in the same manner as the *unit resolution* rule.

Unit Resolution Rule. Let us consider the propositional formula standing for the problem of testing if the concept at node 2 in A1 is less general than the concept at node 2 in A2. In the following, to simplify the presentation we use a label as a placeholder of a concept the given label denotes. The propositional formula encoding the above stated matching problem is as follows:

$$\begin{aligned} & ((\text{Images} \leftrightarrow \text{Pictures}) \wedge (\text{Europe} \leftrightarrow \text{Europe})) \wedge \neg \\ & ((\text{Europe} \wedge \text{Images}) \rightarrow (\text{Pictures} \wedge \text{Europe})) \end{aligned} \quad (3)$$

An intuitive reading of (3) is "is there any situation such that the concept *Images of Europe* is less general than the concept *European Pictures* assuming that *Images* and *Pictures* denote the same concept?". The IW proof of the fact that this is not the case is shown in Figure 5. Notice that, since the DPLL procedure of JSAT handles only CNF formulas, in Figure 5, we show the CNF equivalent of formula (3).

From the explanation in Figure 5, users may learn that the IW proof of the fact that the concept at node 2 in A1 is less general than the concept at node 2 in A2 requires 4 steps and at each proof step (excepting the first one, which is a problem statement) the *unit resolution* rule is applied. Also, users may learn

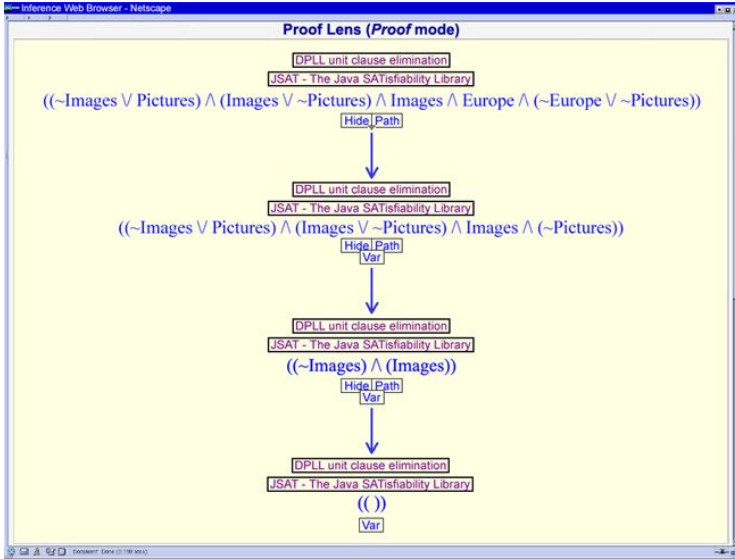


Fig. 5. A graphical explanation of the unit clause rule

the assumptions that are made by JSAT. For example, at the second step the DPLL procedure of JSAT assigns the truth value to (all instances of) the atom *Europe*, therefore making an assumption that there is a model, where what an agent says about *Europe* is always true. According to the *unit resolution* rule, then the atom *Europe* should be stroked out from the input sentence (and, hence it does not appear in the sentence of the step 2).

The explanation of Figure 5 represents some technical details (only the less generality test) of the default explanation in Figure 3. This type of explanations is the most verbose. It assumes (even if the graphical representation of a decision tree is quite intuitive) that the matching system users have some background knowledge in logics and SAT. However, if they do not have it, they have a possibility to learn it by following the publications mentioned in the source metadata information of the DPLL *unit resolution* rule and JSAT (by clicking the *DPLL unit clause elimination* and the *JSAT-The Java SATisfiability Library* buttons respectively).

Two further notes are to be made with respect to the *split* rule. The first is that, it is applied when we need to reason by case distinction, for example, when matching *C1₃* and *C2₃*. The second note is that, in the case of a satisfiable result, only a path of a decision tree standing for a successful assignment is represented. In the case of an unsatisfiable result a full decision tree is reported.

5 Experimental Study

The main goal of the experiments being conducted is to obtain a vision of how the *S-Match* explanations potentially scale to the requirements of the Semantic Web, providing meaningful and adjustable answers in real time.

The semantic (node) matching problem is a CO-NP complete problem, since it is reduced to the validity problem (a formula is valid iff its negation is unsatisfiable) for the propositional calculus. Resolving this class of problems requires exponential time and exponentially long proof logs. However, in all the examples we have done so far proofs are not too long and seem of length polynomial in the length of the input clause. As a matter of fact, [14] shows, that when we have conjunctive concepts at nodes (e.g., $Images \wedge Europe$), these matching tasks can be resolved by the basic DPLL procedure in polynomial time; while when we have full proposition concepts at nodes (e.g., $Images \wedge (Computers \vee Internet)$), the length of the original formula can be exponentially reduced by structure preserving transformations.

In our experiments we have used three test cases: the simple catalog matching problem, presented in the paper, one example from academic and one example from business domains. The business example describes two company profiles: a standard one (mini) and Yahoo Finance (mini). The academic example describes courses taught at Cornell University (mini) and at the University of Washington (mini). Table 1 provides some indicators of the complexity of the test cases¹.

Table 1. Some indicators of the complexity of the test cases

	Images vs. Europe	Yahoo(mini) vs. Standard(mini)	Cornell(mini) vs. Washington(mini)
#nodes	4/5	10/16	34/39
max depth	2/2	2/2	3/3
#leaf nodes	2/2	7/13	28/31

We focus on indicators characterizing explanations of mappings. The analysis of the quality of mappings is beyond scope of this paper². In the experimental study we have used the following indicators:

- Number of mapping elements determined by *S-Match* for a pair of schemas/ontologies. As follows from the definition of semantic matching, this number should be $N1 \times N2$, where $N1$ is the number of nodes in the first schema/ontology, $N2$ is the number of nodes in the second schema/ontology.
- Number of steps in a proof of a single mapping element. This indicator represents the number of PML node sets are to be created in the proof.
- Time needed to produce a proof of a single mapping element. This indicator estimates how fast the modified JSAT in producing IW proofs for a particular task.
- Time needed to produce a proof of all mappings determined by *S-Match* for a pair of schemas/ontologies.

¹ Source files and description of the test cases can be found at <http://www.dit.unitn.it/~accord/>, experiments section.

² Analysis of the quality of mappings produced by *S-Match* and a comparative evaluation against state of the art systems, such as COMA [17], Cupid [19], and Rondo [24] can be found in [12].

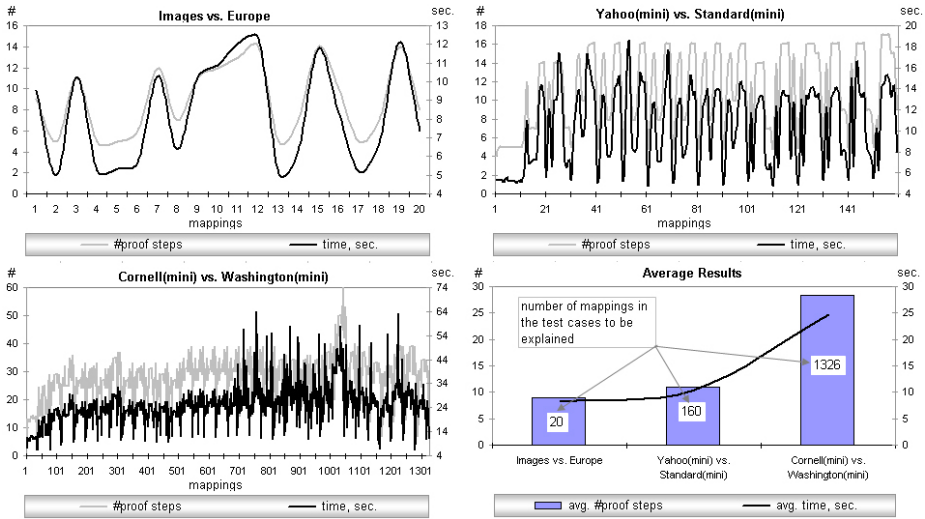


Fig. 6. Experimental Results

In order to conduct tests in a real environment, we used the IW web service of KSL at Stanford University (on a P4-2.8GHz, 1.5Gb of RAM, Linux, Tomcat web server) to generate proofs in PML, while the modified JSAT version was run at the University of Trento (on a P4-1.7GHz, 256 MB of RAM, Windows XP). All the tests were performed without any optimizations: for each single task submitted to JSAT, the IW web service was invoked, no compression methods were used while transferring files, etc.

Figure 6 reports on the results of the experimental study. In particular, for each mapping element of the three test cases, it represents the number of proof steps required and the time needed to generate proofs in PML. Notice, that the proof time indicator in Figure 6 takes into account the time needed by the modified version of JSAT to produce proof information, connection time to the IW web service, time for producing and posting PML documents.

An observation of the spikes starting from the mapping #700 in the time line of the Cornell vs. Washington test case is an example of how Internet connection increases the proof time. The average proof length and proof time for a single mapping element in the test cases of Figure 6 constitute 16 steps and 14 seconds. Time needed to produce proofs of all mapping elements in each test case is *2.7min.* - 20 mappings; *27.7min.* - 160 mappings; and *546.2min.* - 1326 mappings respectively. Notice that the modified JSAT version produces proof information on a single mapping element requiring, in the average, less than 1 millisecond, therefore producing proof information for all mappings, for instance, in the case of 1326 mappings, would require less than 1 minute. Moreover, it is hard to imagine that (ordinary) users will be willing to browse explanations of thousands and even hundreds of mappings. However, one dozen seems to be a reasonable number of mappings to be looked through for a short period of time. Also, as

[12] indicates, *S-Match* mappings quality indicators (e.g., precision, recall), on average are above 80%, therefore, may be that users will not need explanations for a large number of mappings.

Results of the experimental study look promising, however there are proof time issues to be addressed. For example, if a user needs explanations aimed at proof generation and manipulation need to be added. Future work also includes further experiments with more complex test cases. However, the experimental study we have conducted gives a preliminary vision that the explanation techniques proposed potentially scale to requirements of the Semantic Web, providing meaningful and adjustable answers in real time.

6 Discussion

A line of semi-automated schema/ontology matching systems exists, see for instance [4, 9, 10, 17, 19, 24, 27]. Good surveys are provided in [30–32]. To the best of our knowledge, only the iMap system [8] generates explanations of its matching process. However, it substantially differs from *S-Match*, in the type of the result it returns and in the matching approach. In particular, iMap returns an affinity coefficient in the $[0,1]$ range, it does analyze term meaning, and it does not exploit any inference engines. It is based on a combination of constraint/instance-based matching techniques, called *searchers*. Explanations of mappings in the iMap system are based on the idea of a *dependency graph*, which traces the searchers (memorizing relevant slices of the graph) used to determine a particular mapping. Finally, exploiting the dependency graph, explanations are presented to the user in the English format. Although, the meaning of the affinity coefficient returned remains obscure. Additionally, it becomes more obscure as more operations (e.g., use of particular thresholds or weights) are made on these affinity measures.

The DPLL procedure discussed in the paper constitutes a basic (without heuristics and optimizations) propositional satisfiability search procedure of the state of the art SAT engines, such as Chaff [26], etc. Thus, our approach for producing explanations remains valid also for efficient semantic matching.

Recently there has been some work on verifying SAT solvers, in particular on checking the correctness of unsatisfiability proofs by representing the proof as a chronologically ordered set of conflict clauses [15] and using independent resolution-based checking procedures [36]. The major drawback from the IW perspective is that the above mentioned approaches do not provide proofs as independent (portable) objects, which can be checked by a trusted theorem prover. Another problem is that typical traces of DPLL processing are not logical proofs (e.g., they cannot be translated into natural deduction proofs). One approach describes "equivalent" inferences [3, 21] for use in explaining answers as a correct although potentially alternative deductive path. A direct solution to the above problem is provided in [1].

Also, an emergent and challenging research direction in the SAT community concerns *unsatisfiability cores*, which is a task of extracting a (optionally mini-

mal) subset of clauses of the original formula such that the conjunction of these clauses is still unsatisfiable, see, for example [28, 35]. Typically this subset of clauses is much smaller than the original formula. Although, extracting unsatisfiable cores requires producing another trace file representing a decision tree of an unsatisfiable proof. This direction seems promising with respect to the work on explanations of answers from *S-Match*, since by using unsatisfiability cores, proof logs can be significantly reduced. Moreover, minimal unsatisfiability subformulas should allow for localizing a minimal number of axioms implying a particular semantic relation between the nodes under consideration. This approach focuses a user's attention precisely on a reason why this type of a relation holds.

As the use of matching systems for managing semantic heterogeneity grows, it becomes very important to produce explanations of them in order to make the Semantic Web *transparent* and *trustable*. Some technical details of our solution are:

- We use the Proof Mark-up Language for representing *S-Match* proofs, thus facilitating interoperability;
- We use meaningful terms rather than numbers in the DIMACS format, thus facilitating understandability;
- We use the IW tools, thus facilitating customizable, interactive proof and explanation presentation and abstraction;
- Our solution is potentially scalable to the Semantic Web requirements.

7 Conclusions

In this paper, by extending *S-Match* to use the Inference Web infrastructure, we have demonstrated our approach for explaining answers from matching systems exploiting background ontological information and reasoning engines. The explanations can be presented in different styles allowing users to understand the mappings and consequently to make informed decisions about them. The paper also demonstrates that *S-Match* users can leverage the Inference Web tools, for example, for sharing, combining, browsing proofs, and supporting proof meta-information including background knowledge. We also have presented DPLL-based IW explanations of the SAT engine used in the context of *S-Match* tasks. We have tested our approach of explaining *S-Match* answers. The results look promising and demonstrate their potential to scale to the requirements of Semantic Web.

Future work proceeds in at least two directions. Using explanations, a matching system can provide users with meaningful prompts and suggestions on further steps towards the production of a sound and complete result. Having understood the mappings returned by a matching system, users can deliberately edit them manually, therefore providing the feedback to the system. Thus, the first direction includes developing an *environment*, which efficiently exploits the IW proofs and explanations presented in the paper, in order to make the *S-Match* matching process (fully-fledged) *interactive* and *iterative*, involving user in the critical points where his/her input is maximally useful.

The second direction includes (i) improving the *S-Match* proofs and explanations by using *abstraction* techniques more extensively; (ii) conducting a user satisfaction study of the explanations; and (iii) extending explanations to other SAT engines as well as to other non-SAT DPLL-based inference engines, e.g., DLP, FaCT [18], and Pellet [29].

Acknowledgements. This work has been partly supported by the European Knowledge Web network of excellence (IST-2004-507482), by the research grant COFIN 2003 Giunchiglia 40100657, and by research grants from DARPA's DAML and PAL programs.

References

1. C. Barrett and S. Berezin. A proof-producing boolean search engine. In *Proceedings of PDPAR*, 2003.
2. D. Le Berre. JSAT: The java satisfiability library. <http://cafe.newcastle.edu.au/daniel/JSAT/>, 2001.
3. A. Borgida, E. Franconi, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Explaining ALC subsumption. In *Proceedings of Description Logics workshop*, 1999.
4. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In *Proceedings of ISWC*, pages 130–145, 2003.
5. P. Pinheiro da Silva, D. L. McGuinness, and R. Fikes. A proof markup language for semantic web services. Technical report, KSL, Stanford University, 2004.
6. M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, (5(7)), 1962.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, (7):201–215, 1960.
8. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *Proceedings of SIGMOD*, pages 383 – 394, 2004.
9. M. Ehrig and S. Staab. QOM: Quick ontology mapping. In *Proceedings of ISWC*, pages 683–697, 2004.
10. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of ECAI*, pages 333–337, 2004.
11. F. Giunchiglia and P. Shvaiko. Semantic matching. *KER Journal*, (18(3)):265–280, 2003.
12. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS*, pages 61–75, 2004.
13. F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Proceedings of Meaning Coordination and Negotiation workshop at ISWC*, 2004.
14. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient semantic matching. In *Proceedings of ESWC*, 2005.
15. E. Goldberg and Y. Novikov. Verication of proofs of unsatisfiability for CNF formulas. In *Proceedings of DATE*, 2003.
16. N. Guarino. The role of ontologies for the Semantic Web (and beyond). Technical report, Laboratory for Applied Ontology, Institute for Cognitive Sciences and Technology (ISTC-CNR), 2004.

17. H.H.Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, pages 610–621, 2001.
18. I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Automated Reasoning with Analytic Tableaux and Related Methods: Tableaux*, pages 27–30, 1998.
19. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of VLDB*, pages 49–58, 2001.
20. B. Magnini, L. Serafini, and M. Speranza. Making explicit the semantics hidden in schema models. In *Proceedings of workshop on Human Language Technology for the Semantic Web and Web Services at ISWC*, 2003.
21. D. L. McGuinness and A. Borgida. Explaining subsumption in description logics. In *Proceedings of IJCAI*, pages 816–821, 1995.
22. D. L. McGuinness and P. Pinheiro da Silva. Infrastructure for web explanations. In *Proceedings of ISWC*, pages 113–129, 2003.
23. D. L. McGuinness and Pinheiro da Silva P. Registry-based support for information integration. In *Proceedings of IJCAI Workshop on Information Integration on the Web*, 2003.
24. S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of SIGMOD*, pages 193–204, 2003.
25. A.G. Miller. WordNet: A lexical database for english. *Communications of the ACM*, (38(11)):39–41, 1995.
26. M. Moskewicz, C. Madigan, Y. Zhaod, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC*, 2001.
27. N. Noy and M. A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of IJCAI workshop on Ontologies and Information Sharing*, pages 63–70, 2001.
28. Y. Oh, M. N. Mneimneh, Z. S. Andraus, K. A. Sakallah, and I. L. Markov. AMUSE: A minimally-unsatisfiable subformula extractor. In *Proceedings of DAC*, pages 518–523, 2004.
29. B. Parsia, E. Sirin, M. Grove, and R. Alford. Pellet OWL reasoner. <http://www.mindswap.org/2003/pellet/index.shtml>.
30. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, (10(4)):334–350, 2001.
31. P. Shvaiko. A classification of schema-based matching approaches. In *Proceedings of Meaning Coordination and Negotiation workshop at ISWC*, 2004.
32. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. Technical report, DIT-04-087, University of Trento, 2004.
33. M.K. Smith, C. Welty, and D.L. McGuinness. OWL web ontology language guide. Technical report, World Wide Web Consortium (W3C), <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>, February 10 2004.
34. H. Wache, T. Voegelé, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huebner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of IJCAI workshop on Ontologies and Information Sharing*, pages 108–117, 2001.
35. L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formulas. In *Proceedings of SAT*, 2003.
36. L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of DATE*, 2003.

Approximating Description Logic Classification for Semantic Web Reasoning

Perry Groot¹, Heiner Stuckenschmidt², and Holger Wache²

¹ Radboud University Nijmegen, Toernooiveld 1,
6500GL Nijmegen, The Netherlands
Perry.Groot@science.ru.nl

² Vrije Universiteit Amsterdam, de Boelelaan 1081a,
1081HV Amsterdam, The Netherlands
{holger, heiner}@cs.vu.nl

Abstract. In many application scenarios, the use of the Web ontology language OWL is hampered by the complexity of the underlying logic that makes reasoning in OWL intractable in the worst case. In this paper, we address the question whether approximation techniques known from the knowledge representation literature can help to simplify OWL reasoning. In particular, we carry out experiments with approximate deduction techniques on the problem of classifying new concept expressions into an existing OWL ontology using existing Ontologies on the web. Our experiments show that a direct application of approximate deduction techniques as proposed in the literature in most cases does not lead to an improvement and that these methods also suffer from some fundamental problems.

1 Introduction and Motivation

A strength of the current proposals for the foundational languages of the Semantic Web is that they are all based on formal logic. This makes it possible to formally reason about information and derive implicit knowledge. However, this reliance on logics is not only a strength but also a weakness. Traditionally, logic has always aimed at modelling idealised forms of reasoning under idealised circumstances. Clearly, this is not what is required under the practical circumstances of the Semantic Web. Instead, the following are all needed:

- reasoning under time-pressure
- reasoning with other limited resources besides time
- reasoning that is not ‘perfect’ but instead ‘good enough’ for given tasks under given circumstances

It is tempting to conclude that symbolic, formal logic fails on all these counts, and to abandon that paradigm. Our aim is to keep the advantages of formal logic in terms of definitional rigour and reasoning possibilities, but at the same time address the needs of the Semantic Web.

Research in the past few years has developed methods with the above properties while staying within the framework of symbolic, formal logic. However, many of those previously developed methods have never been considered in the context of the Semantic Web. Some of them have only been considered for some very simple underlying description languages [1]. As the languages proposed for modelling Ontologies in the Semantic Web are becoming more and more complex, it is an open question whether those approximation methods are able to meet the practical demands of the Semantic Web. In this article, we look at approximation methods for Description Logics (DLs), which are closely related to some of the currently proposed Semantic Web languages, e.g., OWL.

The remainder of the article is structured as follows. Section 2 gives an overview of various approximation approaches and techniques useful in the context of the Semantic Web. Thereafter, the article focuses on the investigation of a particular approximation technique in the context of a particular reasoning task. Section 3 describes the approximation approach used. Section 4 describes the reasoning task focused on. Section 5 gives experimental results of the approximation method applied to the classification of concepts in a number of Ontologies. Section 6 gives conclusions, discusses the results and the applicability of the analysed approximation approach to the Semantic Web.

2 Approximation Approaches

A typical architecture for a KR system based on DLs can be sketched as in Figure 1 [2], which contains three components that can be approximated to obtain a simplified system that is more robust and more scalable. These components are: (1) the underlying description language, (2) the knowledge base, and (3) the reasoner. The knowledge base itself comprises two components (TBox and ABox), which can also be approximated as a whole or separately. Some general approximation techniques that can be applied to one or more of these components are the following:

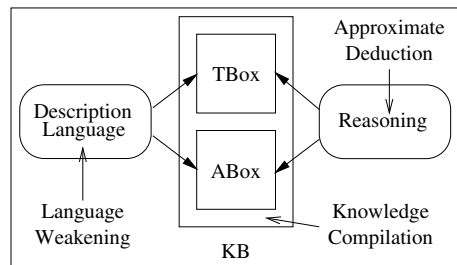


Fig. 1. Architecture of a KR system based on Description Logic together with possible approximation approaches

Language Weakening: The idea of language weakening is based on the well-known trade-off between the expressiveness and the reasoning complexity of a logical language. By weakening the logical language in which a theory is encoded, we are able to trade the completeness of reasoning against run-time. For example, [3] shows how hierarchical knowledge bases can be used to reason approximately with disjunctive information. The logic that underlies OWL Full for example is known to be intractable, reasoners can use a slightly weaker logic (e.g., OWL Lite) that still allows the computation of certain consequences. This idea can be further extended by starting with a very simple language and iterating over logics of increasing strength supplementing previously derived facts.

Knowledge Compilation: In order to avoid complexity at run-time, knowledge compilation aims at pre-processing the ontology off-line such that on-line reasoning becomes faster. For example, this can be achieved by explicating hidden knowledge. Derived facts are added to the original theory as axioms, avoiding the need to deduce them again. In the case of ontological reasoning, implicit subsumption and membership relations are good candidates for compilation. For example, implicit subsumption relations in an OWL ontology could be identified using a DL reasoner, the resulting more complete hierarchy could be encoded e.g., in RDF schema and used by systems that do not have the ability to perform complex reasoning. This example can be considered to be a transformation of the DL language. When one transforms an ontology into a less expressive DL language [4, 5], this often results in an approximation of the original ontology.

Approximate Deduction: Instead of modifying the logical language, approximations can also be achieved by weakening the notion of logical consequence [1, 6]. The approximated consequences are usually characterised as sound but incomplete, or complete but unsound. Only [1] has made some effort in the context of DLs.

The approximation method focused on in this article belongs to the last category, however there is not always a clear classification of one method to the three categories defined above. In the following section we discuss in more detail what is meant by approximating DLs in the remainder of this paper.

3 Approximating Description Logics

The elements of a DL are concept expressions and determining their satisfiability is the most basic task. Other reasoning services (e.g., subsumption, classification, instance retrieval) can often be restated in terms of satisfiability checking [2]. With approximation in DLs, we mean determining the satisfiability of a concept expression through some other means than computing the satisfiability of the concept expression itself. This use of approximation differs with other work on approximating DLs [4, 5] in which a concept expression is translated to another concept expression, defined in a second typically less expressive DL.

In our approach (originally proposed in [1]), in a DL only other, somehow ‘related’, concept expressions can be used that are in some way ‘simpler’ when determining their satisfiability. For example, a concept expression can be related to another concept expression through its subsumption relation, and a concept expression can be made simpler by either forgetting some of its subconcepts or by replacing some of its subconcepts with simpler concepts. In particular, there are two ways that a concept expression C can be approximated by a related simpler concept expression D . Either the concept expression C is approximated by a weaker concept expression D (i.e., less specific, $C \sqsubseteq D$) or by a stronger concept expression D (i.e., more specific, $D \sqsubseteq C$). When $C \sqsubseteq D$, unsatisfiability of D implies unsatisfiability of C . When $D \sqsubseteq C$, satisfiability of D implies satisfiability of C . Note that this is similar to set theory. For two sets C, D , when $C \subseteq D$ holds, emptiness of D implies emptiness of C , and when $D \subseteq C$ holds, non-emptiness of D implies non-emptiness of C .

In [1] Cadoli and Schaerf propose a syntactic manipulation of concept expressions that simplifies the task of checking their satisfiability. The method generates two sequences of approximations, one sequence containing weaker concepts and one sequence containing stronger concepts. The sequences of approximations are obtained by substituting a substring D in a concept expression C by a simpler concept.

More precisely, for every substring D they define the depth of D to be ‘the number of universal quantifiers occurring in C and having D in its scope’ [1]. The scope of $\forall R.\phi$ is ϕ which can be any concept term containing D . Using the definition of depth a sequence of weaker approximated concepts can be defined, denoted by $C_i^\top$, by replacing every existentially quantified subconcept, i.e., $\exists R.\phi$ where ϕ is any concept term, of depth greater or equal than i by $\top$. Analogously, a sequence of stronger approximated concepts can be defined, denoted by $C_i^\perp$, by replacing every existentially quantified subconcept of depth greater or equal than i by $\perp$. The concept expressions are assumed to be in negated normal form (NNF) before approximating them. These definitions lead to the following result:

Theorem 1. *For each i , if $C_i^\top$ is unsatisfiable then $C_j^\top$ is unsatisfiable for all $j \geq i$, hence C is unsatisfiable. For each i , if $C_i^\perp$ is satisfiable then $C_j^\perp$ is satisfiable for all $j \geq i$, hence C is satisfiable.*

These definitions are illustrated by the following concept expression in NNF taken from the Wine ontology¹

$$\text{Merlot} \equiv \text{Wine} \sqcap (\leq 1 \text{ madeFromGrape}.\top) \sqcap \exists \text{madeFromGrape}.\{\text{MerlotGrape}\},$$

which states that a Merlot wine is a wine that is made from the Merlot grape and no other grape. This concept expression contains no $\forall$ -quantifiers. Therefore the depth of the only existentially quantified subconcept ‘ $\exists \text{madeFromGrape}.\{\text{MerlotGrape}\}$ ’ is 0. Substituting either $\top$ or $\perp$ leads to the following approximations for level 0:

$$\text{Merlot}_0^\top \equiv \text{Wine} \sqcap (\leq 1 \text{ madeFromGrape}.\top) \sqcap \top,$$

$$\text{Merlot}_0^\perp \equiv \text{Wine} \sqcap (\leq 1 \text{ madeFromGrape}.\top) \sqcap \perp.$$

No subconcepts of level 1 appear in the concept expression for Merlot. Therefore, $\text{Merlot}_1^\top$ and $\text{Merlot}_1^\perp$ are equivalent to Merlot. The nesting of existential and universal quantifiers is an important measure of the complexity of satisfiability checking when considered from a worst case complexity perspective [8]. This is a motivation for Cadoli and Schaerf to make their specific substitution choices. Furthermore, they are able to show a relation between $C_i^\top$ - and $C_i^\perp$ -approximation and their multi-valued logic based on S -1- and S -3-interpretations [1]. Therefore, properties obtained for S -1- and S -3-approximation also hold for $C_i^\top$ - and $C_i^\perp$ -approximation. These properties include the following: (1) Semantically well founded, i.e., there is a relation with a logic that can be used to give meaning to approximate answers; (2) Computationally attractive, i.e., approximate answers are cheaper to compute than the original problem; (3) Duality, i.e., both sound but incomplete and complete but unsound approximations can be

¹ A wine and food ontology which forms part of the OWL test suite [7].

constructed; (4) Improvable, i.e., approximate answers can be improved while reusing previous computations; (5) Flexible, i.e., the method can be applied to various problem domains. These properties were identified by Cadoli and Schaerf to be necessary for any approximation method.

Although the proposed method by Cadoli and Schaerf [1] satisfies the needs of the Semantic Web identified in Section 1 in theory, little is known about the applicability of their method to practical problem solving. Few results have been obtained for S -1- and S -3-approximation when applied to propositional logic [9, 10, 11], but no results are currently known to the authors when their proposed method is applied to DLs. Current work focuses on empirical validation of their proposed method. Furthermore, DLs have changed considerably in the last decade. Cadoli and Schaerf proposed their method for approximating the language $\mathcal{AL}\mathcal{E}$ (they also give an extension for $\mathcal{AL}\mathcal{C}$), but $\mathcal{AL}\mathcal{E}$ has a much weaker expressivity than the languages that are currently proposed for ontology modeling on the Semantic Web such as OWL. The applicability of their method to a more expressive language like OWL is an open question. Current work takes the method of Cadoli and Schaerf as a basis and focuses on extending it to more expressive DLs.

4 Approximating Classification

The problem of classification is to arrange a complex concept expression into the subsumption hierarchy of a given TBox. We choose this task for two reasons. First, the worst-case complexity of a classification algorithm for expressive representation languages like OWL-DL is known to be intractable. Efficient alternatives have only been proposed for subsets of DLs [12].

Second, classification is a very important part of many other reasoning services and applications. For example, classification is used to generate the subsumption hierarchy of the concept descriptions in an ontology. Furthermore, classification is used in the task of retrieving instances. From a theoretical point of view, checking whether an instance i is member of a concept Q can be done by proving the unsatisfiability of $\neg Q(i)$. Doing this for all existing instances, however, is intractable. Therefore, most DL systems use a process that reduces the number of instance checks. It is assumed that the ontology is classified and all instances are assigned to the most specific concept they belong to. Instance retrieval is then done by first classifying the query concept Q in the subsumption hierarchy and then selecting the instances of all successors of Q and of all direct predecessors of Q that pass the membership test in Q . We conclude that there is a lot of potential for approximating the classification task.

In the following, we first describe the process of classification in DL systems. Afterwards we explain how the approximation technique introduced in Section 3 can be used to approximate (part of) this problem.

For classifying a concept expression Q into the concept hierarchy (Algorithm 1) a number of subsumption tests are required for comparing the query concept with other concepts C_i in the hierarchy. As the classification hierarchy is assumed to be known, the number of subsumption tests can be reduced by starting at the highest level of the hierarchy and to move down to the children of a concept only if the subsumption test is positive. The most specific concepts w.r.t. the subsumption hierarchy which passed the

Algorithm 1 classification**Require:** A classified concept hierarchy with root *Root***Require:** A query concept *Q*VISITED := $\emptyset$ RESULT := $\emptyset$ GOALS := $\{\top\}$ **while** Goals $\neq \emptyset$ **do** $C \in \text{Goals where } \{\text{direct parents of } C\} \subseteq \text{Visited}$ GOALS := Goals $\setminus \{C\}$ VISITED := Visited $\cup \{C\}$ **if** subsumed-by(*Q*,*C*) **then** GOALS := Goals $\cup \{\text{direct children of } C\}$ RESULT := (Result $\cup \{C\}$) $\setminus \{\text{all ancestors of } C\}$ **end if** **end while** **if** |Result| = 1 $\wedge$ subsumed-by(*C*,*Q*) **then**

EQUAL := 'yes'

else

EQUAL := 'no'

end if **return** Equal, Result

subsumption test are collected for the results. At the end of the algorithm, we check if the result is subsumed by *Q* as this implies that both are equal.

Algorithm 1 contains more than one step that can be approximated. For example, the subsumption tests, represented by subsumed-by(*X*, *Y*) in the algorithm, can be approximated using the method of Cadoli and Schaerf.

The subsumption test $Q \sqsubseteq C$ can be reformulated into the unsatisfiability test of $Q \sqcap \neg C$ (Algorithm 2). The idea is to replace standard subsumption checks by a series of approximate checks of increasing exactness. In particular, we use weaker approximations $C_i^\top$ in the $C^\top$ -subsumption algorithm (Algorithm 3) and stronger approximations $C_i^\perp$ in the $C^\perp$ -subsumption algorithm (Algorithm 4). (Note the difference between Algorithm 2 and Algorithms 3 and 4.) The approximations are easily constructed in a linear way. When the approximation at a certain level *I* does not lead to a conclusion (based on Theorem 1) the level *I* is increased by one. This is repeated until the original concept expression is obtained, i.e., the exact subsumption test has to be performed. Algorithm 5 integrates both approximations in one procedure. The approximate versions,

Algorithm 2 subsumption**Require:** A complex concept expression *C***Require:** A Query *Q*CURRENT := $Q \sqcap \neg C$

RESULT := unsatisfiable(Current)

return Result

Algorithm 3 $C^\top$ -subsumption

Require: A complex concept expression C **Require:** A Query Q $I := 0$ **repeat** $\text{CURRENT} := (Q \sqcap \neg C)_I^\top$ $\text{RESULT} := \text{unsatisfiable}(\text{Current})$ **if** Result = 'true' **then****break****end if** $I := I+1$ **until** Current = $Q \sqcap \neg C$ **return** Result

Algorithm 4 $C^\perp$ -subsumption

Require: A complex concept expression C **Require:** A Query Q $I := 0$ **repeat** $\text{CURRENT} := (Q \sqcap \neg C)_I^\perp$ $\text{RESULT} := \text{unsatisfiable}(\text{Current})$ **if** Result = 'false' **then****break****end if** $I := I+1$ **until** Current = $Q \sqcap \neg C$ **return** Result

Algorithm 5 $C_I^\perp$ - $C_I^\top$ -subsumption

Require: A complex concept expression C **Require:** A Query Q $I := 0$ **repeat** $\text{CURRENT} := (Q \sqcap \neg C)_I^\perp$ $\text{RESULT} := \text{unsatisfiable}(\text{Current})$ **if** Result = 'false' **then****break****end if** $\text{CURRENT} := (Q \sqcap \neg C)_I^\top$ $\text{RESULT} := \text{unsatisfiable}(\text{Current})$ **if** Result = 'true' **then****break****end if** $I := I+1$ **until** Current = $Q \sqcap \neg C$ **return** Result

i.e., $C^\top$ -subsumption, $C^\perp$ -subsumption, and $C_I^\perp$ - $C_I^\top$ -subsumption will replace the method `subsumed-by` in Algorithm 1 in the forthcoming experiments.

While these approximate versions can in principle be applied to all occurrences of subsumption tests, we restricted the use of approximations to the first part of the algorithm where the query concept is classified into the hierarchy.

Each DL reasoner (e.g., Fact [13], Racer [14, 15]) implements the classification functionality internally. In order to obtain comparable statements about approximate classification, independently from the implementation of a particular DL reasoner, which may use highly optimised heuristics, we implement our own and independent classification method. The classification procedure was built on top of an arbitrary DL reasoner according to Algorithm 1, which can call the various approximation forms stated in Algorithms 3, 4, and 5. The satisfiability tests are propagated to the DL reasoner through the DIG interface [16] as depicted in Figure 2.

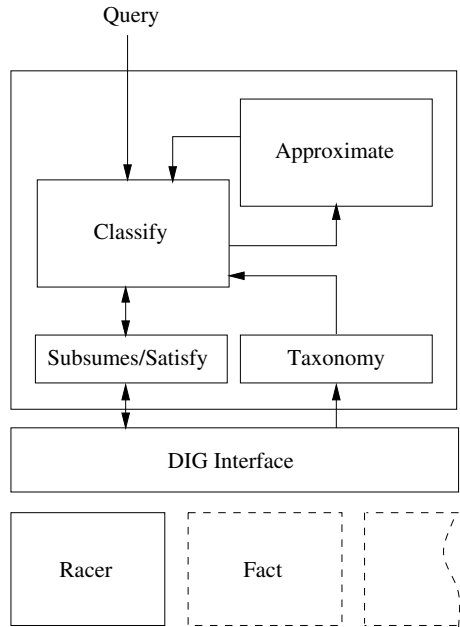


Fig. 2. Architecture of experimental setup

5 Experiments

The main question focused on in the experiments is which form of approximation, i.e., $C_I^\top$, $C_I^\perp$, or their combination, can be used to reduce the complexity of the classification task. The focus of the experiments will not be on the overall computation time, but on the number of operations needed. The goal of approximation is to replace costly reasoning operations by a (small) number of cheaper approximate reasoning operations. The suitability of the method of Cadoli and Schaerf therefore depends on the number of classical reasoning operations that can be replaced by their approximate counterparts without changing the result of the computation.

In the experiments queries are generated automatically. The system randomly selects a number of concept descriptions from the loaded ontology. These definitions are used as queries and are reclassified into the subsumption hierarchy. Note that the queries are first randomly selected, then they are used in the experiments with all forms of approximation.

The first experiments were made with the TAMBIS ontology in which we (re)classified 16 unfoldable concept definitions.² Only the approximation method

² A biochemistry ontology developed in the TAMBIS project [17].

Table 1. Subsumption tests for the reclassification of 16 concepts in TAMBIS

	normal		$C_i^\perp$			$C_i^\top$			$C_i^\perp \& C_i^\top$		
		true	false		true	false		true	false		true
Tambis (16)			$C_0^\perp$	157	32	$C_0^\top$	8	181	$C_0^\perp$	157	32
			$C_1^\perp$	0	0	$C_1^\top$	0	0	$C_0^\top$	8	149
	N	24	279	N	24	247	N	16	279	N	16

originally suggested by Cadoli and Schaerf [1] for $\mathcal{AL}\mathcal{E}$ (described in Section 3) was used.

The results of the first experiments are shown in Table 1, which is divided into four columns. Each column reports the number of subsumption tests when using a certain form of approximation. The first column reports results for the experiment with normal classification (i.e., without approximation), the second column for $C_i^\perp$ -approximation, the third column for $C_i^\top$ -approximation, and the fourth column for a combination of $C_i^\perp$ - and $C_i^\top$ -approximation.

Each column of Table 1 is divided into a number of smaller rows and columns. The rows represent the level of the approximation used, where N denotes normal subsumption testing, i.e., without approximation. The columns represent whether the subsumption test resulted in true or false.³ This distinction is important, because Theorem 1 tells us that only one of those two results will immediately lead to a reduction in complexity, while for the other result approximation has to continue at the next level. This continues until no more approximation steps can be done.

The first column shows that for the reclassification of 16 concepts in the TAMBIS ontology, 24 true subsumption tests and 279 false subsumption tests were needed.

The second column shows that $C_i^\perp$ -approximation leads to a change in normal subsumption tests. Compared to the normal case, the number of false subsumption tests are reduced from 279 to 247. However, the 24 true subsumption tests are not reduced. Note that 32 (279 - 247) false subsumption tests are replaced by 157 true $C_0^\perp$ -subsumption tests and 32 false $C_0^\perp$ -subsumption tests.⁴

The third column shows that $C_i^\top$ -approximation also leads to a change in normal subsumption tests, but quite different when compared to $C_i^\perp$ -approximation. With $C_i^\top$ -approximation we reduce the true subsumption tests from 24 to 16. However, the 279 false subsumption tests are not reduced. Note that 8 (24 - 16) true subsumption tests are replaced by 8 true $C_0^\top$ -subsumption tests and 181 false $C_0^\top$ -subsumption tests. Analogously to $C_i^\perp$ -approximation, no $C_i^\top$ -approximation was used when this would not lead to a change in the subsumption expression.

The fourth column shows the combination of $C_i^\perp$ - and $C_i^\top$ -approximation by using the approximation sequence $C_0^\perp, C_0^\top, C_1^\perp, C_1^\top, \dots, C_{n-1}^\perp, C_{n-1}^\top, normal$. This combination

³ We will use the shorthand ‘true subsumption test’ and ‘false subsumption test’ to indicate these two distinct results.

⁴ Note that the numbers do not add up. The reason for this is that approximation is not used when there is no change in the subsumption expression after approximation, i.e., when $C_i^\perp = C$ the DL reasoner is not called and no subsumption check for $C_i^\perp$ is performed.

leads to a reduction of normal subsumption tests, which is the combination of the reductions found when using $C_i^\perp$ - or $C_i^\top$ -approximation by itself. The true subsumption tests are reduced from 24 to 16 and the false subsumption tests are reduced from 279 to 247. Note that the reduction of 8 (24 - 16) true subsumption tests and 32 (279 - 247) are now replaced by 157 true $C_0^\perp$ -subsumption tests, 32 false $C_0^\perp$ -subsumption tests, 8 true $C_0^\top$ -subsumption tests, and 149 false $C_0^\top$ -subsumption tests.

5.1 Analysis of $C_i^\perp$ -/ $C_i^\top$ -Approximation

The approximation of concept classification in the TAMBIS ontology using the method of Cadoli and Schaerf reveals at least four points of interest. First, Table 1 shows that using $C_i^\perp$ -approximation can only lead to a reduction of the false subsumption tests and $C_i^\top$ -approximation can only lead to a reduction of the true subsumption tests. These results could be expected as they follow from Theorem 1 and are reflected by Algorithm 3, 4, and 5. Using Theorem 1 we have the following reasoning steps for $C_i^\perp$ -approximation:

$$\begin{aligned} \text{Query} \not\models \text{Concept} &\Leftrightarrow (\text{Query} \sqcap \neg \text{Concept}) \text{ is satisfiable} \\ &\Leftarrow (\text{Query} \sqcap \neg \text{Concept})_i^\perp \text{ is satisfiable.} \end{aligned}$$

Hence, when $(\text{Query} \sqcap \neg \text{Concept})_i^\perp$ is *not satisfiable*, nothing can be concluded and approximation can not lead to any gain.

Using Theorem 1 we have the following reasoning steps for $C_i^\top$ -approximation:

$$\begin{aligned} \text{Query} \sqsubseteq \text{Concept} &\Leftrightarrow (\text{Query} \sqcap \neg \text{Concept}) \text{ is not satisfiable} \\ &\Leftarrow (\text{Query} \sqcap \neg \text{Concept})_i^\top \text{ is not satisfiable.} \end{aligned}$$

Hence, when $(\text{Query} \sqcap \neg \text{Concept})_i^\top$ is *satisfiable*, nothing can be concluded and approximation can not lead to any gain.

Second, no approximations are used on a level higher than zero. This is a direct consequence of the TAMBIS ontology containing no nested concept definitions. Further on, we show this to be the case for most ontologies found in practice.

Third, both $C_i^\perp$ - and $C_i^\top$ -approximation are not applied in all subsumption tests that are theoretically possible. With normal classification 303 (24 + 279) subsumption tests are needed. However, with $C_i^\perp$ -approximation in only 189 (157 + 32) cases approximation was actually used. In the remaining 114 (303 - 189) cases approximation had no effect on the concept definitions, i.e., $C_i^\perp = C$, and no test was therefore performed. Hence, in 38% of the subsumption tests, approximation was not used. Similar observations hold for $C_i^\top$ -approximation. This observation indicates that $C_i^\perp$ -/ $C_i^\top$ -approximation is not very useful (at least for the TAMBIS ontology) for approximating classification in an ontology.

Fourth, apart from the successful reduction of normal subsumption tests, we must also consider the cost for obtaining the reduction. For example, with $C_i^\perp$ -approximation we obtained a reduction in 32 false subsumption tests, i.e., 32 normal false subsumption tests could be replaced by 32 cheaper false $C_0^\perp$ -subsumption tests, however it also

cost an extra 157 true $C_0^\perp$ -subsumption tests that did not lead to any reduction. As nothing can be deduced from these 157 true $C_0^\perp$ -subsumption tests, these computations are wasted and reduce the gain obtained with the 32 reduced false subsumption tests considerably. Obviously, these unnecessary true $C_0^\perp$ -subsumption tests should be minimised. No final verdict can be made however, because it all depends on the computation time needed to compute the normal subsumption tests and $C_0^\perp$ -subsumption tests, but 157 seems rather high. Similar observations hold for $C_i^\top$ -approximation.

Analysing the high amount of unnecessary subsumption tests, we discovered a phenomenon, which we call *term collapsing*. We illustrate term collapsing through an example taken from the Wine ontology. Suppose that during a classification the subsumption test $\text{Query} \sqsubseteq \text{WhiteNonSweetWine}$ is generated. The definition for WhiteNonSweetWine is:

$$\text{Wine} \sqcap \exists \text{hasColor}.\{\text{White}\} \sqcap \forall \text{hasSugar}.\{\text{OffDry}, \text{Dry}\}.$$

The subsumption query is first transformed into a satisfiability test, i.e., $\text{Query} \sqsubseteq \text{WhiteNonSweetWine} \Leftrightarrow \text{Query} \sqcap \neg \text{WhiteNonSweetWine}$ is *unsatisfiable*, because $C_i^\perp$ -/ $C_i^\top$ -approximation is defined in terms of satisfiability checking. The definition of $\neg \text{WhiteNonSweetWine}$ is

$$\begin{aligned} &\equiv \neg(\text{Wine} \sqcap \exists \text{hasColor}.\{\text{White}\} \sqcap \forall \text{hasSugar}.\{\text{OffDry}, \text{Dry}\}) \\ &\equiv \neg \text{Wine} \sqcup \forall \text{hasColor}.\neg\{\text{White}\} \sqcup \exists \text{hasSugar}.\neg\{\text{OffDry}, \text{Dry}\}. \end{aligned}$$

and therefore the approximation $(\neg \text{WhiteNonSweetWine})_0^\top$ is

$$\begin{aligned} &\equiv (\neg \text{Wine} \sqcup \forall \text{hasColor}.\neg\{\text{White}\} \sqcup \exists \text{hasSugar}.\neg\{\text{OffDry}, \text{Dry}\})_0^\top \\ &\equiv (\neg \text{Wine})_0^\top \sqcup (\forall \text{hasColor}.\neg\{\text{White}\})_0^\top \sqcup (\exists \text{hasSugar}.\neg\{\text{OffDry}, \text{Dry}\})_0^\top \\ &\equiv \neg \text{Wine} \sqcup \forall \text{hasColor}.\neg\{\text{White}\} \sqcup \top \\ &\equiv \top. \end{aligned}$$

Therefore, approximating the expression $\text{Query} \sqcap \neg \text{WhiteNonSweetWine}$ results in checking unsatisfiability of $\text{Query}_0^\top$, i.e., $(\text{Query} \sqcap \neg \text{WhiteNonSweetWine})_0^\top \Leftrightarrow \text{Query}_0^\top \sqcap \top \Leftrightarrow \text{Query}_0^\top$ is *unsatisfiable*. This test most likely fails, because in a consistent ontology Query will be satisfiable and as Query is more specific than $\text{Query}_0^\top$, i.e., $\text{Query} \sqsubseteq \text{Query}_0^\top$, the latter will be satisfiable.

Analogously, applying $C_i^\perp$ -approximation may result in a collapse of the Query to $\perp$. This occurs whenever Query contains a conjunction with at least one $\exists$ -quantifier. In this case, the entire subsumption test is collapsed into checking the satisfiability of $\perp$. As $\perp$ can never be satisfied, this results in an unnecessary subsumption test.

For the TAMBIS ontology we counted the numbers of occurrences of term collapsing in approximated concept expressions. With $C_i^\top$ -approximation 65 terms out of 181 collapsed. In other words, 35.9% of the approximated false subsumption tests are obviously not needed and should be avoided. With $C_i^\perp$ -approximation it is even more severe drastic: 157 terms out of 157 collapsed. With $C_i^\perp$ - and $C_i^\top$ -approximation 190 terms out of 306 collapsed. In other words 62.1% of the approximated subsumption tests are not needed.

An additional linear time test could be added to the approximation algorithm to detect term collapsing. However, an optimised DL reasoner with lazy evaluation would perform this simple test in a similar way. Experiments indeed show that the DL reasoner quickly detects term collapsing.

Summarising, using the proposed approximation method by Cadoli and Schaerf [1] on query classification in the TAMBIS ontology leads to many collapsing terms. Furthermore, in only a few cases expensive subsumption tests are replaced by cheaper approximated subsumption tests. These results indicate that their approximation method does not fit practical situations well. A different approximation method may provide better approximation.

5.2 Further Experiments

Although practical results of $C_i^\perp$ -/ $C_i^\top$ -approximation are somewhat disappointing for the TAMBIS ontology, similar experiments were made with other ontologies. Table 2 summarises the results of $C_i^\perp$ -/ $C_i^\top$ -approximation applied to the reclassification of 10 unfoldable concepts in five other Ontologies.

Table 2. Number of subsumption tests for reclassification in five ontologies

		normal		$C_i^\perp$		$C_i^\top$		$C_i^\perp \& C_i^\top$	
		true	false	true	false	true	false	true	false
Dolce (10)	$C_0^\perp$	-	-	0	0	-	-	0	0
	$C_0^\top$	-	-	-	-	0	0	0	0
	normal	10	113	10	113	10	113	10	113
Galen (10)	$C_0^\perp$	-	-	0	0	-	-	0	0
	$C_0^\top$	-	-	-	-	0	0	0	0
	normal	10	12190	10	12190	10	12190	10	12190
Monet (10)	$C_0^\perp$	-	-	0	0	-	-	0	0
	$C_0^\top$	-	-	-	-	0	0	0	0
	normal	20	656	20	656	20	656	20	656
MadCow (10)	$C_0^\perp$	-	-	145	0	-	-	145	0
	$C_0^\top$	-	-	-	-	5	140	5	140
	normal	66	152	66	152	61	152	61	152
Wine (10)	$C_0^\perp$	-	-	228	1	-	-	228	1
	$C_0^\top$	-	-	-	-	6	223	6	222
	normal	33	252	33	251	27	252	27	251

For the first three Ontologies in Table 2, the DOLCE⁵, Galen⁶, and Monet ontology⁷, $C_i^\perp$ - or $C_i^\top$ -approximation has no effect. In these three Ontologies, $C_i^\perp$ -/ $C_i^\top$ -approximation does not change any concept expression and therefore no reduction in

⁵ An ontology for linguistic and cognitive engineering [18].

⁶ A medical terminology developed in the Galen project [19].

⁷ An ontology for mathematical web services [20].

normal subsumption tests can be obtained. An analysis of these three Ontologies shows that the Ontologies use some roles and/or attributes, but the $\exists$ - and/or $\forall$ -quantifiers are very rarely used. For example, the Monet ontology contains 2037 concepts, 34 roles, and 10 attributes. The $\exists$ -constructor is only used in 13 definitions (0.64% of all concept definitions). The $\forall$ -constructor is only used in 11 cases (0.54% of all concept definitions). Therefore no quantifiers were present in the ten randomly selected queries. As quantifiers are so rare, $C_i^\perp$ -/ $C_i^\top$ -approximation seems to be useless for those Ontologies.

The next two Ontologies in Table 2, MadCow⁸ and Wine, are somewhat artificial because they are developed for demonstrating the expressive power of DLs. $C_i^\perp$ -/ $C_i^\top$ -approximation was applied to classification in both Ontologies, but this leads to almost no reduction of normal subsumption tests. In the Madcow ontology only 5 true subsumption tests are reduced and in the Wine ontology only 7 subsumption tests are reduced (6 true subsumption tests + 1 false subsumption test). Many more subsumption tests are not reduced. In many cases approximating subsumption tests leads to term collapsing and useless subsumption tests.

6 Conclusions

We argued that the idea of approximate logical reasoning matches the requirements of the Semantic Web in terms of robustness against errors and the ability to cope with limited resources better than conventional reasoning methods. At the same time, approximate logical inference avoids the problems of many numerical approaches for approximate reasoning like the proper interpretation of the numeric values assigned to statements and the problem of acquiring these numbers. We tested a concrete method for approximate logical reasoning in DLs against these claims by applying it to the classification problem on a number of Ontologies. In particular, subsumptions were approximated by sequences of weaker and stronger subsumptions. We showed that in principle both approximations can contribute to the efficiency of query classification.

The main result, however, is that the use of the approximation method for DLs proposed by Cadoli and Schaerf is problematic for two reasons:

- A problematic side effect of using the approximation method is the collapsing of concept expressions leading to many unnecessary approximation steps. This happens either when terms of a disjunction are replaced by $\top$ or terms of a conjunction are replaced by $\perp$. The former case happens when $C_i^\top$ is used on a concept that contains a universal quantifier at the top level of the definition. The latter happens when $C_i^\perp$ is used on a concept with an existential quantifier at the top level of the definition. This feature of the approach is quite problematic as it excludes an important class of query concepts from the method, namely translations of conjunctive queries which are mostly translated using nested existential quantifications [22].
- The experiments show that only in some cases the method is able to successfully replace subsumption tests by cheaper approximations. In many cases like DOLCE, Galen, and Monet no test could successfully be approximated. This observation

⁸ Ontology about mad cows, part of the OWL Reasoning Examples [21].

can be explained by the fact that the approximation method only works on nested expressions that are existentially quantified. Many existing Ontologies, however, do not contain concept expressions with nested expressions. The average ontology on the Semantic Web uses quite simple concept expressions that, if at all, are of depth one. The approximation method by Cadoli and Schaerf was designed based on theoretical considerations to reduce worst case complexity of the subsumption problem, but it does not take practical considerations like the nature of definitions that are likely to be found in Ontologies into account.

We conclude that the use of this specific method of approximating subsumption is often not suited for Semantic Web reasoning. Nevertheless, we believe in the general idea of approximate logical reasoning. The goal is to find an approximation strategy that takes the specifics of Ontologies into account. A particular problem with the current approach is the reliance on alternations of $\forall$ - and $\exists$ -quantifiers. A straightforward way to modify this approach is to find alternative strategies for selecting subexpressions that are to be replaced by $\top$, $\perp$, or other simpler subconcepts. A good candidate, that will be explored in future work, can use domain knowledge to determine the subset of the vocabulary to be replaced. We could for example first exclude very specific terms and then gradually add more specific ones. This and other options for approximating Semantic Web reasoning will be studied in future research.

References

1. Schaerf, M., Cadoli, M.: Tractable reasoning via approximation. *Artificial Intelligence* **74** (1995) 249–310
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press (2003)
3. Borgida, A., Etherington, D.W.: Hierarchical knowledge bases and efficient disjunctive reasoning. In Brachman, R.J., Levesque, H.J., Reiter, R., eds.: *KR'89: Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, San Mateo, California (1989) 33–43
4. Baader, F., Küsters, R., Molitor, R.: Rewriting concepts using terminologies. In Cohn, A.G., Giunchiglia, F., Selman, B., eds.: *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, San Francisco, Morgan Kaufman (2000) 297–308
5. Brandt, S., Küsters, R., Turhan, A.Y.: Approximation and difference in description logics. In Fensel, D., Giunchiglia, F., McGuinness, D., Williams, M.A., eds.: *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, San Francisco, CA, Morgan Kaufman (2002) 203–214
6. McAllester, D.: Truth maintenance. In: *Proceedings of AAAI'90*, Morgan Kaufmann (1990) 1109–1116
7. OWL test suite. <http://www.w3.org/TR/owl-test/>.
8. Donini, F., Hollunder, B., Lenzerini, M., Spaccamela, A.M., Nardi, D., Nutt, W.: The complexity of existential quantification in concept languages. *Artificial Intelligence* **53** (1992) 309–327

9. Groot, P., ten Teije, A., van Harmelen, F.: Towards a Structured Analysis of Approximate Problem Solving: a Case Study in Classification. In Dubois, D., Welty, C., Williams, M., eds.: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004)*, Whistler, BC, Canada, AAAI Press (2004) 399–406
10. ten Teije, A., van Harmelen, F.: Exploiting domain knowledge for approximate diagnosis. In Pollack, M., ed.: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*. Volume 1., Nagoya, Japan, Morgan Kaufmann (1997) 454–459
11. ten Teije, A., van Harmelen, F.: Computing approximate diagnoses by using approximate entailment. In Aiello, G., Doyle, J., eds.: *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, Boston, Massachusetts, Morgan Kaufman (1996)
12. Grosz, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, ACM (2003) 48–57
13. Horrocks, I.: The FaCT system. In: *Proceedings of the second International Conference on Analytic Tableaux and Related Methods*. Volume 1397 of *Lecture Notes in Artificial Intelligence*., Springer (1998) 307–312
14. Haarslev, V., Möller, R.: Race system description. In: *Proceedings of the 1999 Description Logic Workshop (DL'99)*. *CEUR Electronic Workshop Proceedings* (1999) 130–132
15. Haarslev, V., Möller, R.: Racer system description. In: *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*. Volume 2083 of *Lecture Notes in Artificial Intelligence*., Springer (2001) 701–705
16. Bechhofer, S., Möller, R., Crowther, P.: The dig description logic interface. In: *Proceedings of DL2003 International Workshop on Description Logics*, Rome (2003)
17. Baker, P., Brass, A., Bechhofer, S., Goble, C., Paton, N., Stevens, R.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. An Overview. In: *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB'98)*, Menlow Park, California, AAAI Press (1998) 25–34
18. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: *Ontology Library*. Wonder-Web Deliverable D18. Laboratory For Applied Ontology - ISTC-CNR. (20003)
19. Rector, A.L., Nowlan, W.A., Glowinski, A.: Goals for concept representation in the galen project. In: *Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care (SCAMC-93)*, Washington DC, USA (1993) 414–418
20. Caprotti, O., Dewar, M., Turi, D.: Mathematical service matching using description logic and owl. In: *To appear in Proceedings 3rd Int'l Conference on Mathematical Knowledge Management (MKM'04)*. Volume 3119 of *Lecture Notes in Computer Science*., Springer-Verlag (2004)
21. Bechhofer, S.: *OWL Reasoning Examples*. University of Manchester (2003) <http://owl.man.ac.uk/2003/why/latest/>.
22. Horrocks, I., Tessaris, S.: A conjunctive query language for description logic aboxes. In: *National conference on artificial intelligence (AAAI 2000)*. (2000) 399–404

AIS and Semantic Query

Rana Kashif Ali and Steve Cayzer

¹ The University of Birmingham, Birmingham B15 2TT, UK

² HP Laboratories, Bristol, UK

Abstract. The semantic web has created various exciting opportunities to explore. Here we present a nature inspired solution to one such opportunity; that of semantic queries for information retrieval. We take our inspiration from the human immune system and develop an analogy between antibodies and queries. Successful antibodies are those that are activated by an infection. These antibodies are stimulated to clone, but imperfectly, giving rise to a multitude of similar antibodies that are better suited to tackle the infection. Analogously, queries producing relevant results can be cloned to give rise to various similar queries, each of which may be an improvement on the original query. The semantic web, being concept based, has a set of rules for creating expressive yet standardised queries with clear semantics guiding their modification. This paper discusses the implementation and evaluation of such an immune based information retrieval technique for the semantic web. Two query mutation operators; *RandomMutationOperator* and *ConstrainedMutationOperator* are proposed and compared in terms of their *precision*, *recall* and *convergence*. We have found the presented approach to be viable, and we discuss the potential for further improvements.

1 Introduction

The presented work combines disparate areas of research namely, semantic web, Artificial Immune Systems (AIS), Query Expansion (QE) and Information Retrieval (IR). In this section, we introduce these concepts before outlining the structure of the remaining paper.

The Semantic Web is an extension of the current World Wide Web (WWW) in which resources are connected semantically rather than through hyperlinks. This semantic connectivity is achieved by making metadata about resources available for machine processing. Metadata is typically written in Resource Description Format (RDF: [4]) to conform to a model, or ontology. Both the metadata and ontologies are available to all. Different frameworks exist to manipulate the RDF metadata, for example Jena [3], which supports the Resource Description Query Language (RDQL; [5]) for querying metadata.

Computational models and problem solving approaches inspired from the Biological Immune System (BIS) are called AIS [7]. The presented work focuses on the *clone-and-refine* paradigm of the BIS, according to which, a body exposed to antigen produces various antibodies, some of which (those showing a higher affinity

to the antigen) are more suitable to overcome the infection. These antibodies undergo affinity-related cloning and mutation to produce novel, but similar, antibodies, some of which might be an improvement over the original antibody and can better tackle the infection. However, some antibodies may be self-reactive and hence must be destroyed or they will cause an autoimmune reaction. We bring this idea into the realm of query expansion by establishing an analogy between antibodies and queries, refining the search process with clonal expansion, mutation and screening of self-reactive queries.

The next section gives an overview of the related work, followed by the details of the AIS and query expansion. In section 4 we describe the experimental plan and the obtained results. In the final section we discuss future directions and present our conclusions.

2 Related Work

AIS is a relatively new area of research with a diversity of applications such as data mining, computer security and robotics. A full survey can be found in Jon Timmis' and Leandro de Castro's book [7], but here we describe work relevant to our application.

The notion of AIS for semantic queries was first proposed by Lee et al [1]. They show, using the Gene Ontology (GO) as an example, how data can be retrieved based on the principles of immunity by expanding queries. Their work does not involve a concrete implementation but does provide a useful conceptual framing for our work. We have applied the idea to a new domain, filled in some details and provided a real application that we evaluate.

Efthimiadis's work [2] provides a sound foundation of traditional query expansion, drawing a distinction between manual, automated and interactive approaches. However the methods he describe are predominantly keyword based - that is, not semantic. Thus, we aim to demonstrate the feasibility of using semantic queries within a principled query expansion framework.

Our work is grounded using real semantic web data. The Semantic Web Environmental Directory, SWED [6], provides a decentralised, RDF-backed portal for storing the details of environmental organisations in the UK. SWED provides a novel 'facet browse' mechanism that enables users to navigate to the organisations of interest using conjunctive combinations of metadata attributes (for example, "Not for Profit organisations based in Bristol that are concerned with animal welfare"). Our semantic query mechanism facilitates a different approach, akin to a semantic "More Like This" utility.

3 AIS for Semantic Query Expansion

A web based semantic search utility was developed with the AIS infrastructure embedded in it. A high level view of the utility is shown in Figure 1. It is also important at this stage to establish the mapping between the AIS and BIS. In our AIS

we regard the irrelevant results as self and relevant results as non-self. Antibodies are semantic queries and antigens are a collection of the non-self (relevant results). Finally, mutation is equivalent to query expansion. Thus, mutation of a query may result in queries that are better suited to answer a particular search criterion. On the other hand, mutation may result in queries that return irrelevant results; these are deemed self-reactive and hence are destroyed.

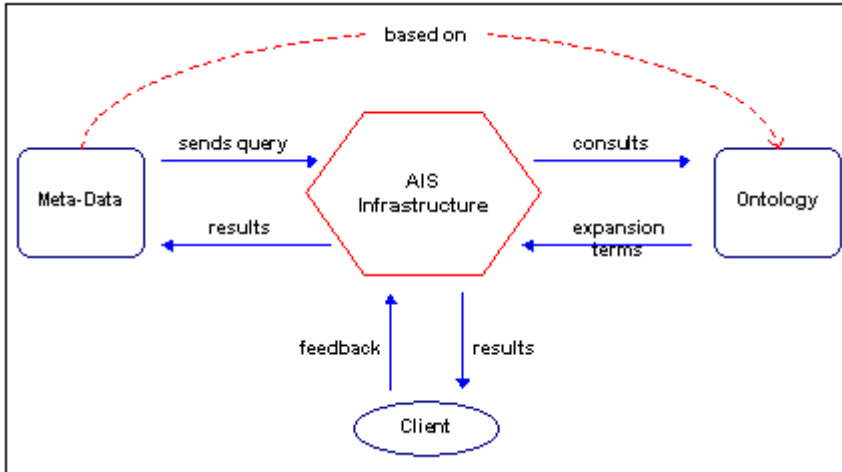


Fig. 1. AIS infrastructure and flow of information

3.1 User Interface

The interface to the search process is designed so as to let the user know how the query expansion is being done. Initially, the user chooses a particular organisation of interest, the details of which appear as shown in figure 2. This is a single record drawn from the SWED dataset that will seed the semantic queries. The only feedback at this stage involves the user clicking on the 'FIND SIMILAR' link. Upon the feedback the AIS initialises, fetches relevant organisations and presents them grouped by queries (figure 3). This new interface lets the user build sets of self and non-self by specifying results as either relevant (non-self) or irrelevant (self). These sets act as an evaluation mechanism for query refinement, guiding the semantic query population towards novel relevant results. The interface also bears links at the top that allow users to view/edit sets of self and non-self. There is also an option to view the state of the query pool that gives a good insight into the expansion process. This feature may however be removed from a commercial application to avoid complexity.

3.2 AIS Algorithm

The algorithm for the AIS is given below followed by explanation of its constituents.

```
begin
  take initial user feedback
  initialise Q, query population based on feedback
  while (halting criteria not met)
    display the results of queries in Q and take user feedback
    add relevant results to non-self and irrelevant to self
    evaluate fitness of queries in Q
    select queries with highest fitness (Q_s) using fitness
  proportionate selection
    perform clonal expansion on the selected queries to form Q_c
    apply mutation operator to transform Q_c to Q_m
    replace the previously selected queries Q_s with Q_m
  end while
end
```

Organisation Details	
Vincent Wildlife Trust	
Organisation number	prorg0002
Acronym	VWT
Year formed	1977
Description	The Vincent Wildlife Trust operates an otter rehabilitation centre for orphaned or injured otters from throughout the UK with reintroductions occurring in Northern Ireland and eastern England.
Type	registered_charity
Topics	animal_welfare farming farming_fish_and_other_aquaculture resource_management management_water pollution_control_remediation recreation recreation_water-based species species_animals species_mammals wildlife_habitats
Telephone	voice: 0171-283 2089 fax: 0171-929 0604
Email	contact@vwt.org.uk
URL	http://www.vwt.org.uk/
Primary Contact	Secretary / Treasurer
Postal address	10 Lovat Lane, London , EC3R 8DT, England
FIND SIMILAR	

Fig. 2. User interface for semantic query invocation

Initial User Feedback

This involves a user specifying one organisation of interest and saying that s/he wants to find similar organisations.

Initialisation of AIS

When the user click the 'FIND SIMILAR' link the AIS is initialised with the query population equal to the input parameter INIT_POP_SIZE (5 in our case). This initial query population is generated randomly using two ontologies used in the SWED data, namely *organisation_type* and *topic*. The pseudo code for the initialisation of the AIS is given below

RESULTS				
SELF	PREVIOUSLY SELECTED QUERY POPULATION	CURRENTLY SELECTED QUERY POPULATION	COMPLETE CURRENT POOL [selected+nonselected]	NON-SELF
	1			
IRRELEVANT	QUERY SELECT organisations where TYPES = [private_limited_company] AND TOPICS = [recreation]			RELEVANT
☐	Festival of the Countryside			☐
	2			
IRRELEVANT	QUERY SELECT organisations where TYPES = [registered_charity] AND TOPICS = [built_environment]			RELEVANT
☐	Cathedral Camps			☐
☐	Campaign for the Protection of Rural Wales			☐
☐	National Trust for Scotland			☐
☐	Action with Communities in Rural England			☐
☐	Barn Owl Trust			☐
	3			
IRRELEVANT	QUERY SELECT organisations where TYPES = [registered_charity] AND TOPICS = [animal_welfare]			RELEVANT
☐	Zoological Society of London, The			☐
☐	Barn Owl Trust			☐
☐	National Animal Welfare Trust			☐
☐	Humane Slaughter Association			☐
	4			
IRRELEVANT	QUERY SELECT organisations where TYPES = [private_limited_company] AND TOPICS = [business_and_commerce]			RELEVANT
☐	Planning Exchange, The			☐
	5			
IRRELEVANT	QUERY SELECT organisations where TYPES = [registered_charity] AND TOPICS = [developing_world]			RELEVANT
☐	Pesticides Trust, The			☐
☐	Trust for Education and Development			☐

Fig. 3. User interface for semantic query expansion

```

set generatedQueries = 0
while(INIT_POP_SIZE > generatedQueries) {
    randomly select a organisation_type and assign it to the new
    query
    generate a random number, count, between 0 and
    MAX_TOPICS_IN_QUERY
    select count number of topics randomly from the ontology
    combine the organisation_type and topics to make a query
    if(query produces some results) {

```

```

        generatedQueries++;
        add query to the AIS
    }
}

```

Once the AIS is initialised, the antibodies/queries within it are extracted and displayed along with their results. As mentioned, the user may give feedback by specifying whether a particular result is irrelevant or relevant.

Fitness Evaluation

Once the user has given feedback, the antibodies/queries need to be evaluated. The fitness of an antibody in our case is the measure of how well it binds to the non-self while avoiding self. This is equivalent to a search for queries returning many relevant and few irrelevant results. The following formula was used to evaluate the antibodies

$$affinity = \frac{NonSelf \times w_{pos} + Self \times w_{neg} + New \times w_{neutral}}{total\_number\_of\_results} \quad (1)$$

Queries can return results that are relevant (NonSelf), irrelevant (Self) or have unknown relevance (New). The numbers of each result set are weighted and combined into a fitness function, whose weights are: $w_{pos} = 1$, $w_{neg} = 0$ and $w_{neutral} = 0.4$. The choice of the values for different weights was empirical.

Selection

In a pure AIS individuals are selected so as to maximise the collective affinity against the antigen called *affinity maturation*. Affinity maturation is fitness proportionate and thus can be modelled as roulette wheel selection.

Clonal Expansion

This is a two step process the first step involves generation of the clones based on fitness and the second step involves mutation of the clones using mutation operators. Any cloned antibody/query should fulfil the following constraint.

$$\{all\_results\} - \{Self \cup NonSelf\} \neq \emptyset \quad (2)$$

In other words a query should return some previously unseen results.

Mutation Operators

The two query mutation operators that we used for evaluation of the AIS are as follows:

ConstrainedMutationOperator

This operator appends, deletes or changes various characteristics of the individual, retaining all others 'as is'.

```

generate a random number between 0 and 1, random
if (random < TYP_CHG_PROB) {
    replace existing organisation_type with a one randomly chosen
    from the ontology
}

```

```

}
else {
    retain the old organisation_type
}
define three variable, append, delete and change
initialise the variables with random numbers between 0 and 1
for each topic in the query {
    if(append >= MUTATION_RATE) {
        append a random topic to the query
    }
    if(delete >= MUTATION_RATE) {
        delete the topic from the query
    }
    if(change >= MUTATION_RATE) {
        replace the topic with a randomly selected topic
    }
}

```

RandomMutationOperator

This is a more exploratory operator, which allows a considerable degree of novelty in the generated query.

```

generate a random number between 0 and 1, random
if(random < TYP_CHG_PROB) {
    replace existing organisation_type with a one randomly chosen
    from the ontology
}
else {
    retain the old organisation_type
}
generate a random number between 1 and MAX_TOPICS_IN_QUERY, count
while(count != 0) {
    generate a random number between 0 and 1, rate
    if(rate <= MUTATION_RATE) {
        choose a topic randomly from the old query and add to the new
one
    }
    else {
        choose a topic randomly from the topic ontology
        add the topic to the new query
    }
    decrement count
}

```

Replacement Strategy and Halting Criteria

All individuals that are selected during the selection phase are replaced with the offspring. The unselected individuals however, remain in the AIS to maintain diversity in the population. There are two possible halts to the search process. Firstly, when all the desired results have been found. Secondly, if further query expansion is not possible.

4 Experimental Setup and Results

We compared the proposed mutation operators in terms of precision, recall and convergence. In the first set of experiments the performance of the two operators was

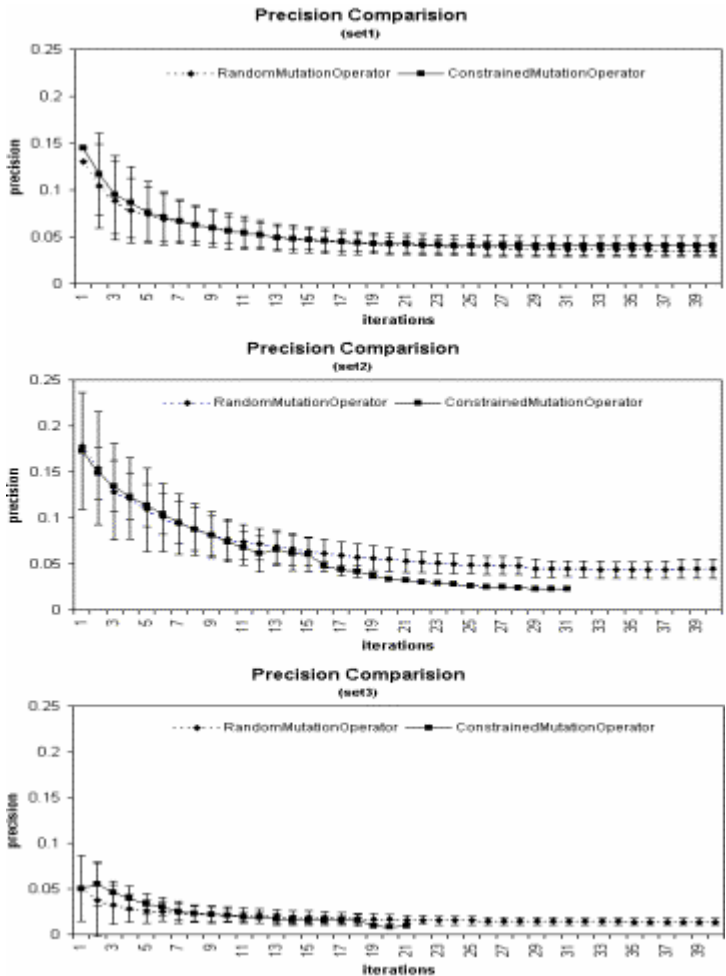


Fig. 4. Precision comparison on different input data sets averaged over 30 runs. Bars show standard deviation

observed on three different input data sets. The second set of experiments was aimed towards finding the change in performance with changing mutation rate on only one input data set. We implemented an automated test script to simulate a user interacting with the system. The script was controlled by various parameters for example the maximum number of iterations and results to be marked as relevant or irrelevant in every iteration. For the first set of experiments, three input data sets were selected by a real user of the system, each containing around 10 relevant and 90 irrelevant items. The task of the AIS was to find organisations in a particular input data set in minimum number of iterations. The starting point for the AIS was one randomly chosen organisation from the set. Figure 4 shows the precision for both operators on three different input data sets. The precision was measured cumulatively:

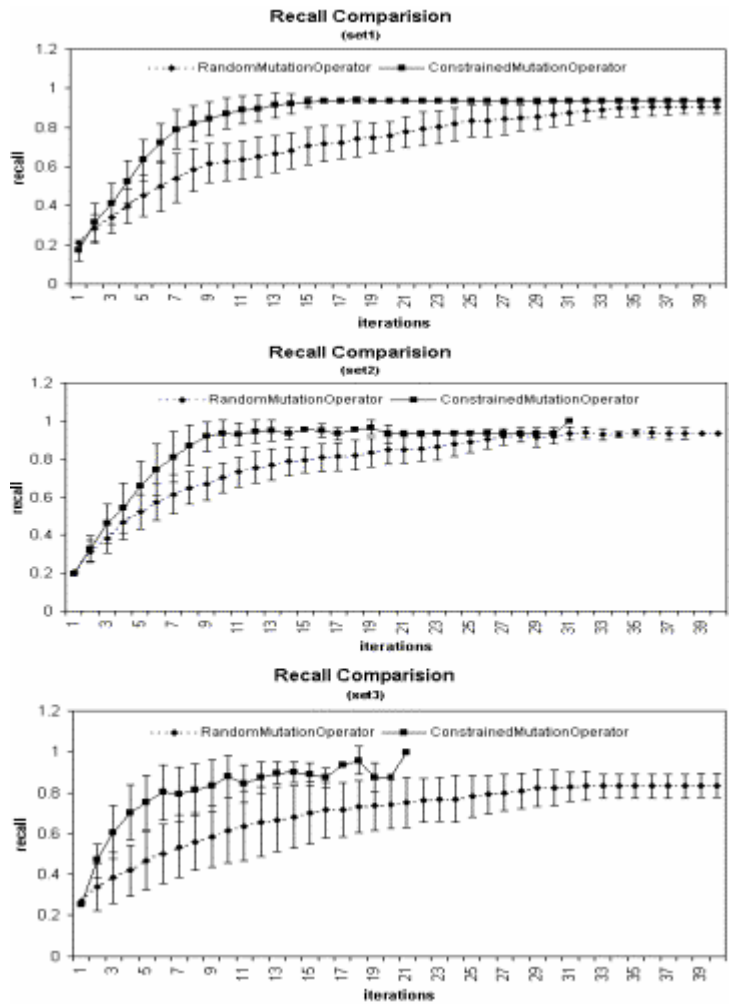


Fig. 5. Recall comparison on different input data sets, averaged over 30 runs. Bars show standard deviation

$$precision = \frac{relevant\_results\_so\_far}{all\_results\_so\_far} \quad (3)$$

This cumulative precision decayed asymptotically, as it becomes progressively harder to find the remaining relevant items. In the early stages, the constrained operator was significantly superior (e.g. iteration 4, dataset 1: p-value < 0.05, Student's t-test).

Figure 5 shows the recall comparison. Again, recall was calculated cumulatively, so it increases asymptotically to a theoretical maximum of 1.0.

$$recall = \frac{relevant\_results\_so\_far}{all\_relevant\_results} \quad (4)$$

The *ConstrainedMutationOperator* clearly performs better since it reaches a higher value of recall more quickly (iteration 10, all datasets: p-value < 0.0001).

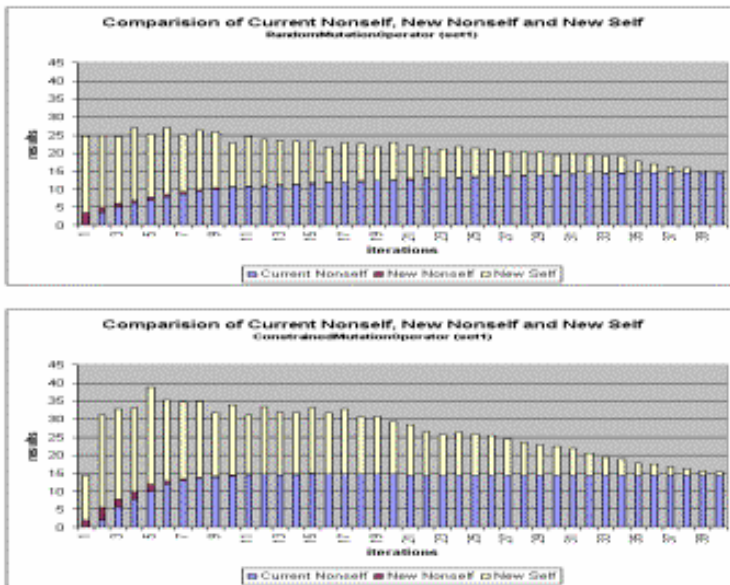


Fig. 6. Convergence comparison on input data set 1, averaged over 30 runs

Figures 6, 7 and 8 compare the convergence between the two operators. *RandomMutationOperator* exhibits delayed convergence and finds fewer relevant and irrelevant results. The *ConstrainedMutationOperator* on the other hand is aggressive in nature and converges quickly.

For the second set of experiments we selected the input data set 2 and changed the mutation rate from 0 to 1. We found no significant different between the operators in terms of precision which remained under 0.2. However, in case of recall we found that the two operators behave in an opposite way (figure 9). The exploratory

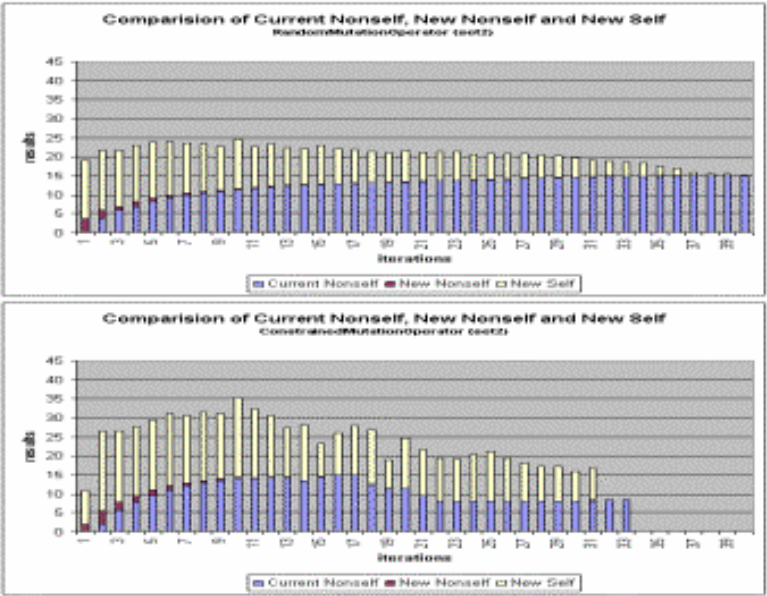


Fig. 7. Convergence comparison on input data set 2, averaged over 30 runs }

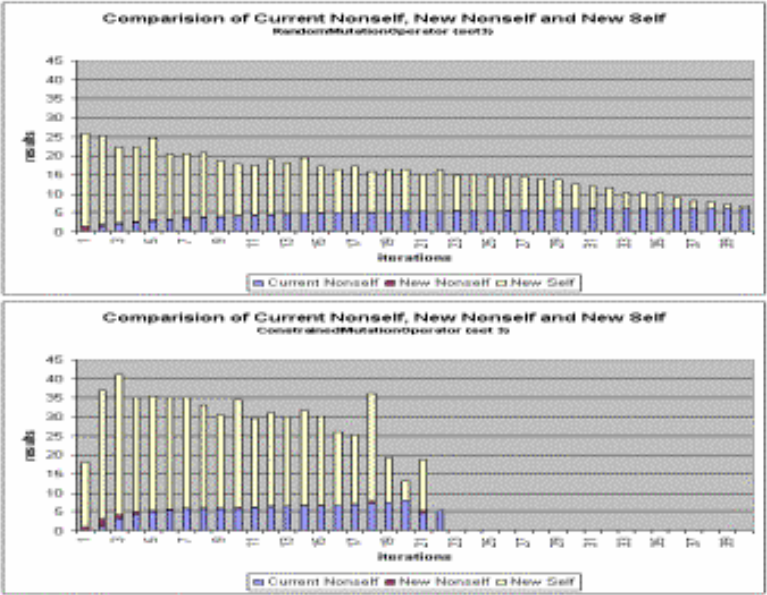


Fig. 8. Convergence comparison on input data set 3, averaged over 30 runs }

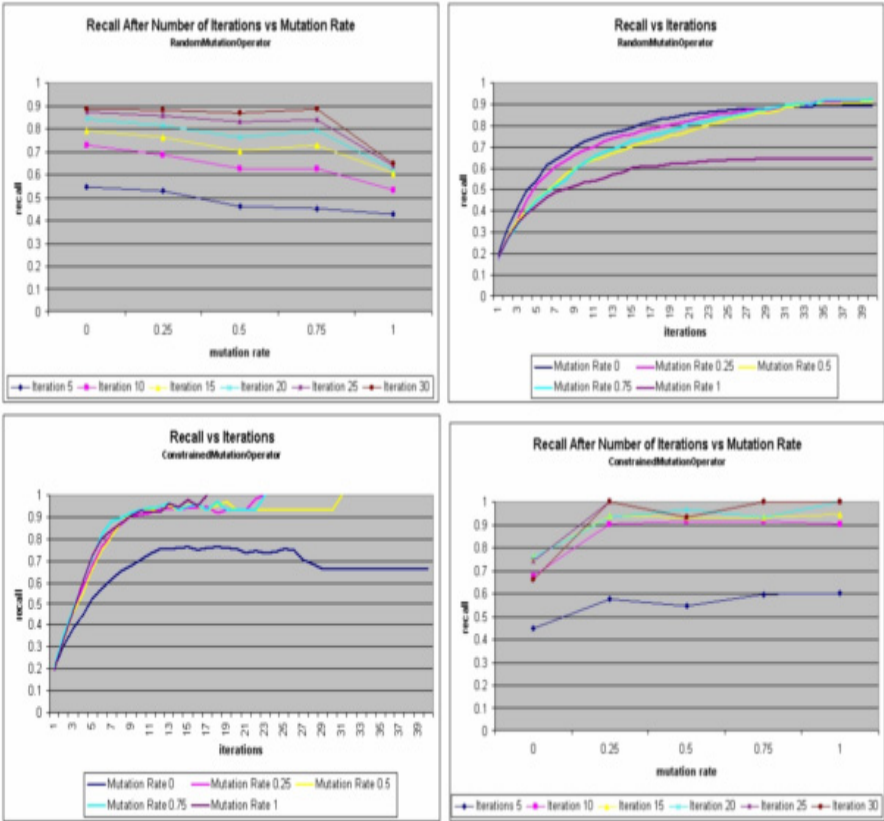


Fig. 9. Recall comparison with changing mutation rate

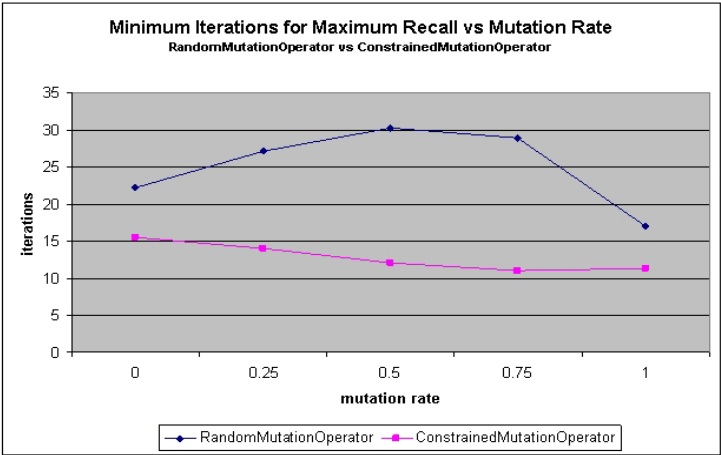


Fig. 10. Minimum iterations for maximum recall vs. mutation rate, averaged over 30 runs

RandomMutationOperator benefits from a low mutation rate, whereas the more aggressive *ConstrainedMutationOperator* requires some mutation in order to avoid premature convergence. These results are underlined by our experiments investigating the effect of mutation rate on speed of convergence (figure 10).

5 Conclusion and Future Directions

We have shown in this paper that AIS are a useful metaphor for query expansion on the semantic web. Our initial mutation operators demonstrate ways of exploring and exploiting the query space. An obvious next step would be to try the operators on a larger dataset (more than 100 organisations) with more sophisticated semantic markup (more than two ontologies). Another fruitful direction would be a study to explore suitable metaphors for the user interface. Finally it would be possible to integrate this work into the SWED portal and to provide value to a real semantic web community.

References

1. Lee, D., Kim, J., Jeong, M., Won, Y., Park, H., Lee, K.: Immune-Based Framework for Exploratory Bio -Information Retrieval from the Semantic Web. Artificial Immune Systems: Second International Conference, ICARIS 2003, Edinburgh, UK, September 1-3, 2003, Proceedings **2787** (2003) 128--135 Lecture Notes In Computer Science, Springer.
2. Efthimiadis N.E.: Annual Review of Information Systems and Technology (ARIST) Query Expansion **31** 1996 121--187 Information Today Inc Medford, NJ
3. Jena Semantic Web Framework <http://jena.sourceforge.net/>
4. Resource Description Framework (RDF) <http://www.w3.org/RDF/>
5. RDQL - A Query Language for RDF W3C Member Submission 9 January 2004
6. <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>
7. SWED - The Semantic Web Environmental Directory
8. <http://www.swed.org.uk/swed/index.html>
9. de Castro, L.N., Timmis, J.: Artificial Immune Systems: A New Computational Approach Sept 2002 Springer-Verlag London. UK.

Querying RDF Data from a Graph Database Perspective

Renzo Angles and Claudio Gutierrez

Department of Computer Science, Universidad de Chile
{`rangles`, `cgutierr`}@dcc.uchile.cl

Abstract. This paper studies the RDF model from a database perspective. From this point of view it is compared with other database models, particularly with *graph database models*, which are very close in motivations and use cases to RDF. We concentrate on query languages, analyze current RDF trends, and propose the incorporation to RDF query languages of primitives which are not present today, based on the experience and techniques of graph database research.

1 Introduction

The *Resource Description Framework* (RDF) can be viewed from at least two perspectives: (1) From a logical perspective, as a minimal fragment of logic that includes all relevant features needed as representation language for metadata, or as the W3C recommendation [1] says: RDF is an assertional language intended to be used to express propositions using precise formal vocabularies; and (2) From a database perspective, as an extension of data models used in the database community, in particular graph database models. The former point of view has been an active area of research. This does not come as surprise knowing that RDF emerged as a language to represent metadata on the Web, distilling the experience of the community of knowledge representation and Web researchers and developers [2]. The latter point of view has received less attention and will be the focus of this paper. We will consider RDF as a data model in the database tradition.

The term *data model* has been used in the database community with different meanings and in diverse contexts. In this paper we will use it in two senses. In a broad or abstract sense, a data model is a collection of conceptual tools for describing the real-world entities to be modeled in the database and the relationships among these entities [3]. In a strict or concrete sense, a *data model*, as defined by Codd [4], is as a combination of three components: (a) a collection of data structure types; (b) a collection of transformation operators and query language and, (c) a collection of general integrity rules.

In the broad sense, RDF can be considered a data model: a collection of conceptual tools for describing real-world entities, namely metadata on the Web. But also in the strict sense of the term, RDF qualifies as well: Point (a) has been more or less addressed at a basic level. One of the documents of the RDF suite [5]

speaks of *graph data model* meaning by this concept the data structure implicitly defined by sets of triples. Although one can discuss the precise meaning of this concept [6], the graph-like nature of RDF data is clear. Point (c), namely the integrity constraints, is an open issue for RDF, especially when considering the duality of RDF as an open world specification of distributed resources on the Web versus RDF as data model for large single-source repositories with all the issues of a standard database system. The topic of constraints is outside the scope of this paper.

This paper concentrates on point (b), namely query languages. This is an active area from a development and implementation point of view, and there is a W3C Working Group addressing the issue of RDF data access, which has a proposal of RDF query language directed mainly to access data on distributed sources [7]. There are also works addressing foundational issues, e.g. [8, 9, 10]. Nevertheless, the discussion about RDF as a full fledged strict database model and the design and primitives of a query language for such a model is a topic less developed, and probably one of the most needed if we want to take advantage of all the potentialities of the RDF data model (e.g. query optimization, query rewriting, views, update).

The consideration of RDF as database model puts forward the issue of developing coherently all its database features. In particular its query language should address the kind of queries and problems of the application domains the abstract data model is intended to represent. One of the motivations of this paper are those application domains where interconnection at large scale and navigation of a network is the main modeling theme. Examples of this are biology [11], police-like applications [12], navigation in bibliographic databases, etc. To give a flavor of the type of problems, consider queries like: “are suspects A and B related?”, submitted to a police database, or “what is the Erdős number of author X”, submitted to (an RDF version of) DBLP. The first asks for “relevant” paths connecting these resources in the (RDF) police database, and the second asks simply for the length of the shortest path between the nodes representing Erdős and X. Current proposals of RDF query languages [13, 14, 7] do not support¹ queries like these. To address these type of problems, the notions and techniques of *graph databases* can be very valuable. Graph databases are systems designed to support storing and querying information in the form of graphs. They were important, together with object-oriented databases, in the database research of the nineties, and lost part of their appeal after the irruption of semi-structured data models and XML. We claim that graph database models can be a sound support for the design of an RDF database model, particularly for RDF query languages.

Contributions. In this paper we study the RDF model from a database perspective, compare it with other abstract database models, focusing on query languages and graph databases. We restrict in this paper to the logical level,

¹ A language is said to *support* a feature if it provides facilities that make it convenient (reasonable easy, safe and efficient) to use that feature [15].

i.e., avoid –when is possible– physical, implementation and indexing considerations. In particular we:

- Compare the RDF model with classical abstract database models putting particular emphasis in graph database models.
- Study current RDF query languages with respect to their capabilities to support graph-like queries and conclude that they give little or no support for them.
- Survey the notions, techniques and systems developed in the area of graph database query languages, and its applicability to the RDF model.
- Propose primitives for RDF query languages based on the graph database experience.

Outline of the paper. Section 2 compares the RDF model with other database models. Section 3 surveys graph database models and their query languages. Section 4 presents a brief review of current RDF query languages and investigates the support they give for querying graph-like data. Finally, in Section 5 we propose a set of primitives to be incorporated into RDF query languages. Each of them is carefully reviewed against experience of query language development in graph databases.

2 Comparison of RDF with Other Abstract Database Models

Beginning in the seventies numerous data models have been proposed, each of them with their own concepts and terminology. Surveys and taxonomies of data models are as manifold as data models themselves (see e.g. [3, 16, 17], [18]). Several of these data models have features relevant for the RDF model. In this section we compare the RDF model with the most important of them. A summary is presented in Table 1.

Physical Models. They were the first ones to offer the possibility to organize large collections of data. Among the most important ones are the hierarchical [19] and network [20] models. These models lack good abstraction level and are very close to physical implementations. The data-structuring is not flexible and not apt to model non-traditional applications. For our discussion they do not much have relevance.

Relational Data Model. was introduced by Codd [21] to highlight the concept of level of abstraction by introducing a clean separation between physical and logical levels. Due to its simplicity of modeling it gained wide popularity among developers and business applications. It is based on the simple mathematical notion of relation, which together with its associated algebra and logic, made the relational model a primary model for database research. In particular, its standard query and transformation language, SQL, became a paradigmatic language for querying.

Although an RDF specification can be logically viewed as a set of binary relations, the differences with the relational model are manifold. Among the

Table 1. Summary of comparison among different database models. The parameters are: abstraction level, complexity of the data items modeled, degree of connectivity among the data and support to get this information, and finally, flexibility to store different types of data

MODEL	LEVEL	DATA COMPLEX.	CONNECTIVITY	TYPE of DATA
Network	physical	simple	high	homogeneous
Relational	logical	simple	low	homogeneous
Semantic	user	simple/medium	high	homogeneous
Object-O	logical/physical	complex	medium	heterogeneous
XML	logical	medium	medium	heterogeneous
RDF	logical	medium	high	heterogeneous

most relevant ones are: the relational model was directed to simple record-type data with a structure known in advance (airline reservations, accounting, etc.). The schema is fixed and extensibility is a difficult task. Integration of different schemas is not easy nor automatizable. The query language does not support paths, neighborhoods and queries that address connectivity (an exception is transitivity). There are no objects identifiers, but values.

Semantic Models. ([22]) have their origin in the necessity to provide more expressiveness and incorporate a richer set of semantics into the database from the user point of view. They allow database designers to represent objects and their relations in a natural and clear manner (similar to the way the user view an application) by using high-level abstraction concepts such as aggregation, classification and instantiation, sub- and super-classing, attribute inheritance and hierarchies [16]. A well-known example is the entity-relationship model [23]. It has become a basis for the early stages of database design, but due to lack of preciseness cannot replace models like relational or O-O.

For RDF database research, semantic models are relevant because they are based on a graph-like structure which highlights the relations between the entities to be modeled.

Object Oriented Data Models. ([24]) are based on the object-oriented programming paradigm. Their objective is representing data as a collection of objects that are organized in classes and have complex values and methods associated with them. They are intended to model non-conventional database applications consisting of complex objects systems with many semantically interrelated components as in CAD/CAM, computer graphics or information retrieval.

Object-oriented database models have been related to Graph database ones because the explicit or implicit graph structure in their definitions [25], [26], [27]. Nevertheless, there remain important differences rooted in the form that each of them models the world. O-O models view the world as a set of objects having certain state (data) and interacting among them by methods. On the contrary, graph database models, and RDF in particular, model the world as a network

of relations. The emphasis in RDF is on the interconnection of the data, the network of relations among the data and the properties of these relations. The emphasis of O-O is on the objects, their values and methods. However, there are proposals to apply O-O concepts to RDF [28].

Semistructured Data Models. ([29, 30, 31]) are oriented to model semi-structured data. Of all the most visible models in the literature, the semi-structured data model is one of the closest in several points to RDF. Semi-structured models deal with data whose structure is irregular, implicit and partial, and whose schema is usually very large, contained within the data itself, and rapidly evolving [31]. One of the best representative is OEM [32]. It is a model based on objects, which have unique identifiers, and values that can be simple types or object references. There is a natural graph representation: objects are nodes, and values are labeled arcs. The main differences with RDF are: the lightweight inferencing available, the existence of blank nodes, the stronger typing system and the fact that labels are also nodes in RDF.

Another representative is the XML model [33]. There are substantial differences between XML and RDF. First, RDF has a higher abstraction level; in fact RDF is an application of XML to represent metadata. Structurally XML has a ordered-tree-like structure against the graph structure of RDF. At the semantic level, in XML the information about the data is part of the data (in other words XML is self-describing); in contrast, RDF expresses explicitly the information about the data using relations between entities. An important advantage of RDF is its extensibility in both schema and instance level. See [34, 35] for a major comparison of these models.

3 Graph Database Models and Their Query Languages

3.1 Graph Database Models

Graph database models appeared with the objective of modeling information whose logical structure is a graph. In this sense, they are the closest to the RDF model by the data type used. Among the first ones, we have the the Logical Data Model [36, 37] and the Functional Data Model [38], which define an implicit structure of labeled graphs. The Logical data model introduces basic, composition, and collection nodes, all of which can be modeled in RDF. On the other hand, in many semantic and object oriented data models the conceptual representation of data is transparently graph-based. For example O₂ [39] defines basic, tuple-structured, and set-structured types (the first type is similar to an RDF blank node and the remainder two can be modeled as relations in RDF); GOOD [27] is oriented primarily to graphical user interfaces; OEM [32] addresses the information exchange problem, and is oriented to express resources and relations in a standard way (in agreement to the RDF philosophy); GDM [40] defines instances and schema graphs with features similar to RDF (e.g. domain and range of relations, typeOf properties). Models like G-BASE [41], Gram [42],

GraphDB [43] and GRAS [44] propose explicit graph data models.² Besides these models based on graphs, there are other approaches which use as formalization generalizations of the notion of graph, such as hypergraphs (e.g. see GROOVI [25], the hypernode model [45, 46]) and hygraphs (e.g. see Hy+ [47]). Note that strictly speaking, RDF graphs are ordered hypergraphs [6].

3.2 Graph Query Languages

There are several proposals of query languages for models that represent information with a explicit or implicit graph structure. In this context, from now on we assume that a graph database has n nodes and e edges.

Cruz et al. [48] propose the graphical query language G for querying data represented as a labeled graph. It introduces the concept of graphical query, which is based on a pattern graph that use regular expressions to represent recursive queries. G evolved into a more powerful language called $G+$ [49] where a query graph is the basic building block. Query graph nodes may be labeled with variables and edges labeled with regular expressions. A simple query has two elements, a query graph that specifies the class of patterns to search and a summary graph that represent how to restructure the answer obtained by the query graph.

GraphLog [50] is a query language for hypertext. It presents a extension of $G+$ by adding negation and unifying the concept of a query graph. A query is now only one graph pattern containing one distinguished edge (which corresponds to the restructured edge of the summary graph in $G+$). The effect of the query is to find all instances of the pattern that occur in the database graph and for each one of them define a virtual link represented by the distinguished edge.

Gram [42] presents a query algebra where regular expressions over data types are used to select walks (paths) in a graph. It uses a data model where walks are the basic objects. A walk expression is a regular expression without union, whose language contains only alternating sequences of node and edge types, starting and ending with a node type. The query language is based on a hyperwalk algebra with operations closed under the set of hyperwalks.

Gemis and Paredaens [51] present PaMal, a graphical model for describing schemes and instances of object-databases and a graphical data manipulation language based on pattern matching.

Gütting [43] proposed an object-oriented data model and query language for graph databases called GraphDB. A database in GraphDB is a collection of object classes divided in: simple classes (simple objects that represent nodes), link classes (links between nodes that represent edges) and path classes (representing several paths in the database). A query consists of several steps. Each step computes operations that specify argument subgraphs in the form of regular ex-

² Note that a direct applicability of a graph model to RDF is not possible due to the particular RDF graph property where resources possibly can occur as edge labels as well as node labels. To solve this problem an intermediate model (e.g bipartite graphs [6]) can be defined.

pressions over link class names that extend or restrict dynamically the database graph.

LoREL [30] is a query language for semistructured data designed for the Object Exchange Model (OEM) [32]. LoREL is an extension of OQL [52], extending its characteristics to handling semistructured data.

Oriented to search the Web, Flesca and Grego [53] show to how use partially ordered languages to define path queries to search databases and present results on their computational complexity. In addition, a query language based on the previous ideas was proposed in [54].

4 Current RDF Query Languages and Their Graph Support

4.1 Brief Overview of RDF Query Languages

Several languages for querying RDF data have been proposed and implemented, some in the lines of traditional database query languages (e.g. SQL, OQL), others based on logic and rule languages. Some of them are: *RQL* [9] is a typed language for querying RDF repositories; *SquishQL*³ is a SQL-style query language that permits simple graph navigation in RDF sources; *RDQL* [55] is an implementation of *SquishQL*; *RDFQL*⁴ is a statement-based query language with a SQL-style to perform queries, inference operations, and construction of views on RDF structured data; *TRIPLE* [56] is a language that allow rule definition, inference and transformation of RDF models; *Notation 3* (N3) [57] provides a text-based syntax for RDF; *Versa*⁵ is a graph-based language with some support for rules; *SeRQL* combines characteristics of languages like *RQL*, *RDQL*, *N-Triple*, *N3* plus some new features; *RXPath*⁶ is a query language based on XPath; Good surveys are [13, 14].

W3C members that conform the RDF DAWG presented a Working Draft in October 2004, which specifies a set of use cases, requirements, and objectives for an RDF query language and data access protocol [58]. *SPARQL* [7] is an RDF query language designed to meet such requirements and design objectives mentioned previously. It defines a query language with a SQL-like style, where a simple query is based on query patterns, and query processing consists of binding of variables to generate pattern solutions (graph pattern matching). *SPARQL* is still a work in progress.

4.2 Graph Properties in Current RDF Query Languages

To illustrate the problems of current RDF query languages in querying graph-like properties, we chose seven query languages and seven graph properties one

³ <http://ilrt.org/discovery/2001/02/squish/>

⁴ <http://www.intellidimension.com/>

⁵ <http://4suite.org/>

⁶ <http://rx4rdf.liminalzone.org/>

Table 2. Support of some current RDF query languages for some example graph properties (“±” indicates partial support and “×” no support)

PROPERTY	RQL	SeRQL	RDQL	Triple	N3	Versa	RXPath
Adjacent nodes	±	±	±	±	±	±	×
Adjacent edges	±	±	±	±	×	×	×
Degree of a node	±	×	×	×	×	×	×
Path	×	×	×	×	×	×	±
Fixed-length Path	±	±	±	±	±	×	±
Distance between two nodes	×	×	×	×	×	×	×
Diameter	×	×	×	×	×	×	×

would like to retrieve (see [59]). The summary of the results, presented in Table 2, are as follows. An RDF graph can be considered a directed graph. This direction produces problems in languages that do not have a union operator when retrieving neighborhoods, e.g. “all statements involving a given resource”. Some query results violate the query language property of closure [14] by returning results which are not in RDF format. There are two main problems concerning paths: (a) most languages support only querying for patterns of paths which are limited in length and form (the issue of edge direction blows up the size of the query exponentially); (b) RxPath is able to retrieve only paths starting from a fixed node and with some other restrictions. Aggregated functions like COUNT, MIN, MAX applied to paths could be used to answer queries as for the degree of a node, the distance between nodes, and the diameter of a graph. None of these functions is systematically supported, even though, for example, the original version of RQL has a COUNT function on the number of triples.

5 Graph Primitives for RDF Query Languages

In this section we present desirable graph primitives of a query language for the RDF data model, based on the experience of the graph database query languages discussed in previous sections. We stress the graph-like features that in our opinion are missing in today’s RDF query languages.

Before discussing the primitives in detail, let us enumerate desirables features for an RDF query language. They are very much inspired by a similar wish-list stated by Abiteboul [31] for semi-structured data. They are: Standard database-style query primitives; Navigation in the style of semi-structured data or Web-style browsing; Searching for patterns in an information-retrieval style; Temporal queries, including versioning; Querying both the data and the schema in the same query; Incorporating transparently the lightweight inferencing of RDF Schema and relevant polynomial-time extensions; Sound theoretical foundation;

The following groups of primitives comprise features of graph query languages (see Sec. 3), graph properties presented in section 4.2 and those found in the DAWG Draft. We think they constitute a starting point of graph properties

Table 3. Support of some graph database query languages for the example graph properties of Table 2 (“√” indicates support, “±” partial support, “×” no support, and “?” indicates there is no information available)

PROPERTY	G	G+	GraphLog	Gram	GraphDB	LoReL	F-G
Adjacent nodes	±	√	√	√	±	√	±
Adjacent edges	±	√	√	√	±	√	±
Degree of a node	×	√	√	×	?	×	×
Path	√	√	√	√	√	√	√
Fixed-length Path	√	√	√	√	√	√	√
Distance between two nodes	×	√	√	×	?	×	×
Diameter	×	√	√	×	?	×	×

that should be supported by an RDF query language. In each case we survey the support that graph database languages gives them. As motivation, Table 3 shows the support graph query languages give to the properties in Table 2.

Paths and Connectedness. One of the most fundamental graph problems is to compute reachability information (use case 2.5 in DAWG Draft [58]). In fact, many of the recursive queries that arise in relational databases and, more generally in data with graph structure, are in practice graph traversals characterized by path problems. The importance of such queries is studied in several works [60, 61, 62, 63]. One of the challenges to incorporate such notion into a query language is its computational complexity. Finding simple paths with desired properties in direct graphs is very difficult, and essentially every nontrivial property gives rise to an NP-complete problem [64]. Yannakakis [65] surveyed a set of paths problems relevant to the database area including computing transitive closures, recursive queries and the complexity of path searching. Extension of query languages for solve graph traversal problems are surveyed in [66].

In what follows, we describe the support that the query languages of the database models described in Section 3.1 give to path problems.

A initial implementation of G translate the graphical queries into C-Prolog programs. Simple paths are traversed using certain non-Horn clause constructs available in Prolog. Although, it does not support cycles or finding the shortest path, it was a good approximation to a graph query language.

The evaluation of path queries in G+ is a two-stage process consisting of a depth-first search of the graph database and use of a nondeterministic finite state automaton to control the search. In addition path queries are a subset of the class of linear chain queries and hence can be evaluated rapidly in parallel. The evaluation algorithm can be shown to compute the identity query in $O(e)$ time and the transitive closure in $O(ne)$ time. G+ was implemented in the HyperG system providing primitive operators like depth-first search, shortest path, transitive closure and connected components.

Motivated by the implementation of G+, Mendelzon and Wood [67] studied the problem of finding all pair of nodes connected by a simple path such that

the concatenation of the labels along the path satisfies a regular expression. Although the regular simple path problem is in general NP-complete, the paper presents an algorithm that runs in polynomial time in the size of the graph when some conditions fulfilled: the graph is acyclic, the regular expression is restricted (according to the definition in the paper), or the graph complies with a cycle constraint compatible with the regular expression. The evaluation algorithm uses a deterministic finite automaton to traverse paths in the graph. They also prove the intractability of certain types of simple paths in a particular class of directed graphs and characterize a class of queries about regular simple paths which can be evaluated in polynomial time. The analysis and implementation in this paper, assume that the graph can be entirely stored in main memory.

The expressive power of GraphLog is characterized by establishing the equivalence between GraphLog, stratified linear Datalog (a language of function-free Horn clauses), non deterministic logarithmic space, and transitive closure. The queries expressible in the language are exactly those that can be computed in space logarithm in the size of the database.

To implement graph operations in GraphDB, efficient graph algorithms are used. Shortest path and cycle both were implemented using the A* algorithm. Moreover, nodes, paths and subgraphs are indexed using path classes and index structures like B-Tree and LSD-Tree.

LoREL presents a SQL-style query language that support two types of path expressions, simple path expressions, which allow to obtain the set of objects reachable by following a sequence of labels starting from a named object in the OEM graph and a more powerful syntax for path expressions, called general path expressions based on wildcards and regular expressions. To outperform query execution, the Lore DBMS [68] implements the query language Lore and uses two kinds of indexes, a link (edge) index called Lindex, and a value index called Vindex. A Lindex takes an object identifier and a label, and returns the object identifiers of all parents via the specified label. A Vindex takes a label, operator, and a value, and returns all atomic objects having an incoming edge with the specific label and a value satisfying the specific operator and value. Vindexes and Lindexes are implemented using B+ trees and linear hashing respectively.

In graph databases where the number of nodes is very large (e.g. the Web) it is useful to subdivide the domain of evaluation by selecting subsets of the domain on the base of some criteria. With this objective, Flesca and Greco [53] introduce partially ordered regular languages based on some order on the nodes. Such languages are an extension of regular languages where strings are partially ordered, for example, two strings s_1 and s_2 , such that $s_1 > s_2$, denote two paths in the graph with the constraint that the path s_1 should be preferred to the path s_2 . In later work [54], they present an algebra for partially ordered relations, an algorithm for the computation of path queries and show that computing an instance of a graph query can be done in polynomial time. Also, they present a SQL-like language that consider general paths and extended regular expressions, and show how extended regular expressions can be used to search the Web. With similar motivations, and in the context of RDF, Anyanwu and Sheth [69] intro-

duced a path operator ρ to address relevant relationships between entities called semantic associations. Semantic associations are represented in a RDF graph as sequences (i.e. edges, paths) between entities or more complex structures of sequences, and a notion of similarity between them is defined. The implementation of the ρ -operator is evaluated on two strategies, first implementing a processing layer in existing RDF data storage technologies and, second the use of a memory resident graph representation of the RDF model along with the use of efficient graph traversal algorithms (e.g. transitive closure and isomorphism of paths).

Pattern Matching. consists in determining if there exists a mapping (or isomorphism) between a graph pattern and a subgraph of a database graph (use cases 2.1, 2.12 and 2.13 in DAWG Draft [58]). Pattern matching deal with two problems, the graph isomorphism problem that has a unknown computational complexity, and the subgraph isomorphism problem which is NP-complete. Pattern matching has attracted a great deal of attention specially on data mining (see [70] for a survey), update [51, 71], querying [48, 72, 50] and visualization [26]. Sasha *et al.* [64] present a survey of pattern-matching based algorithms for fast searching in trees and graphs.

PaMal use graph patterns to describes the part of the database instance that are affected by a operation (addition and deletion of nodes and edges). In the case of GraphDB, the subgraph problem is solved moving the conditions into subsequent graph operations or other database access.

Aggregate Functions. are operations non related to the data model that permit to summarize or operate on the query results (use cases 2.3, 2.4, 2.6, 2.8, 2.10, 2.11, 2.14 and 2.15 in DAWG Draft [58]). Such functions are oriented to deal directly with the structure of the underlying graph, such as the degree of a node, the diameter of the graph (or a set of nodes), the distance between nodes, etc.

With the purpose of performing computations on retrieved subgraphs product of a query operation, G+ defines two types of summary operators: path operators which summarize on the values of the attributes along paths and set operators which summarize on the values of the attributes on a set of paths. The set of such operators include sum, products, maximum and count.

GraphLog becomes more expressive that relational algebra and calculus with aggregates, adding aggregate operators (e.g. MAX, SUM, etc.) and path summarization. The implementation of GraphLog use algorithms discussed in [67].

Gram, consistent with its SQL-like syntax, defines two types of algebraic operations: unary (projection, selection, renaming) and binary (join, concatenation, set operations) which are closed under the set of hyperwalks. PaMal provides a reduce-operation to work with a special group of instances called reduced instances and programming constructs (loop, procedure and program). Finally, GraphDB query language support further operations, e.g. for sorting, grouping, and aggregate functions (e.g. Sum).

Neighborhoods. The notion of neighborhood is relevant for information having a graph-like nature (use case 2.7 in DAWG Draft [58]). In these models, in-

formation (represented by nodes) closed (in the graph) is usually semantically related. The primary notion is *adjacency*. Both node and edge adjacency in RDF are important in various contexts. A more advanced notion of adjacency, like *the k-neighborhood of a node*, is necessary in several contexts. The need of 1-neighborhood retrieval in an RDF Graph is argued in [73] and [74]. In the RDF context, inference of new triples is relevant in Vertex and Edge adjacency queries. To the best of our knowledge, the notion of neighborhood as primitive for query languages has not been studied systematically in the database literature.

6 Conclusions

We considered RDF from the perspective of graph database modeling. We compared it with other database models. We surveyed graph database models and query languages in order to argue the convenience that the RDF community incorporate database experience and technologies into further development of the RDF model and query language design. In concrete, we propose that RDF query language should incorporate graph database query language primitives. Further work includes developing use cases, formalizing requirements and building benchmarks for queries using the graph-like structure of the model.

Acknowledgments. This research was supported by Millenium Nucleus, Center for Web Research (P01-029-F), Chile. R. Angles was supported by Mecesup project No. UCH0109. C. Gutierrez was partially supported by FONDECYT No. 1030810.

References

1. Hayes, P.: RDF Semantics. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/> (2004)
2. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/> (1999)
3. Silberschatz, A., Korth, H.F., Sudarshan, S.: Data models. *ACM Computing Surveys* 28 (1996) 105-108
4. Codd, E.F.: Data Models in Database Management. In: *Proc. of the workshop on Data abstraction, databases and conceptual modeling*, ACM Press (1980) 112-114
5. Klyne, G., Carroll, J.: Resource Description Framework (RDF) Concepts and Abstract Syntax. <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/> (2004)
6. Hayes, J., Gutierrez, C.: Bipartite Graphs as Intermediate Model for RDF. In: *Proc. of the 3th ISWC Conference*. Number 3298 in LNCS, Springer-Verlag (2004) 47-61
7. Prudhommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. <http://www.w3.org/TR/rdf-sparql-query/> (2005)
8. Horrocks, I., Tessaris, S.: Querying the Semantic Web: A Formal Approach. In: *Proc. of the 13th ISWC*. Number 2342 in LNCS, Springer-Verlag (2002) 177-191

9. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proc. of the 11th WWW conference, ACM Press (2002) 592–603
10. Gutierrez, C., Hurtado, C., Mendelzon, O.: Foundations of Semantic Web Databases. In: Proc. of the 23th ACM PODS. (2004)
11. Olken, F.: Tutorial on Graph Data Management for Biology. IEEE Computer Society Bioinformatics Conference (CSB) (2003)
12. Sheth, A., Aleman-Meza, B., Arpinar, I.B., Halaschek-Wiener, C., Ramakrishnan, C., Bertram, C., Warke, Y., Avant, D., Arpinar, F.S., Anyanwu, K., Kochut, K.: Semantic Association Identification and Knowledge Discovery for National Security Applications. *Journal of Database Management* 16 (2005) 33–53
13. Magkanaraki, A., Karvounarakis, G., Anh, T.T., Christophides, V., Plexousakis, D.: Ontology Storage and Querying. Tech. Report 308, ICS-FORTH - Hellas (2002)
14. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: Proc. of the 3th ISWC conference. Number 3298 in LNCS, Springer-Verlag (2004) 502
15. Stroustrup, B.: What Is Object-Oriented Programming? *IEEE Softw.* 5 (1988) 10–20
16. Navathe, S.B.: Evolution of data modeling for databases. *Communications of the ACM* 35 (1992) 112–123
17. Beeri, C.: Data Models and Languages for Databases. In: Proc. of the 2nd ICDT. Volume 326 of LNCS., Springer-Verlag (1988) 19–40
18. Kerschberg, L., Klug, A.C., Tsichritzis, D.: A Taxonomy of Data Models. In: Systems for Large Data Bases, North Holland and IFIP (1976) 43–64
19. Tsichritzis, D.C., Lochovsky, F.H.: Hierarchical Data-Base Management: A Survey. *ACM Comput. Surv.* 8 (1976) 105–123
20. Taylor, R.W., Frank, R.L.: CODASYL Data-Base Management Systems. *ACM Comput. Surv.* 8 (1976) 67–103
21. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 26 (1983) 64–69
22. Peckham, J., Maryanski, F.J.: Semantic Data Models. *ACM Computing Surveys* 20 (1988) 153–189
23. Chen, P.P.: The Entity-relationship Model-toward a Unified View of Data. *ACM TODS* 1 (1976) 9–36
24. Kim, W.: Object-Oriented Databases: Definition and Research Directions. *IEEE TKDE* 2 (1990) 327–341
25. Levene, M., Poulouvanasilis, A.: An Object-oriented Data Model Formalised through Hypergraphs. *DKE* 6 (1991) 205–224
26. Andries, M., Gemis, M., Paredaens, J., Thyssens, I., Bussche, J.: Concepts for Graph-Oriented Object Manipulation. In: 3rd EDBT Conference. Volume 580 of LNCS., Springer-Verlag (1992) 21–38
27. Gyssens, M., Paredaens, J., Bussche, J., Gucht, D.: A Graph-Oriented Object Database Model. *IEEE TKDE* 6 (1994) 572–586
28. Bassiliades, N., Vlahavas, I.P.: R-DEVICE: A Deductive RDF Rule Language. In: Proc. of the 3th RuleML. (2004) 65–80
29. Buneman, P.: Semistructured Data. In: Proc. of the 16th PODS, ACM Press (1997) 117–121
30. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.: The Lorel Query Language for Semistructured Data. *Int. Journal on Digital Libraries* 1 (1997) 68–88

31. Abiteboul, S.: Querying Semi-Structured Data. In: Proc. of the 6th Int. Conference on Database Theory. Volume 1186 of LNCS., Springer-Verlag (1997) 1–18
32. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object Exchange across Heterogeneous Information Source. In: Proc. of the 11th ICDE, Taipei, Taiwan, IEEE (1995) 251–260
33. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998. (<http://www.w3.org/TR/1998/REC-xml-19980210>)
34. Gil, Y., Ratnakar, V.: A Comparison of (Semantic) Markup Languages. In: Proc. of the 15th FLAIRS Conference. (2002)
35. Arroyo, S., Ding, Y., Lara, R., Stollberg, M., Fensel, D.: Semantic Web Languages. Strengths and Weakness. In: International Conference in Applied computing. (2004)
36. Kuper, G.M., Vardi, M.Y.: A New Approach to Database Logic. In: Proc. of the 3th ACM PODS, ACM Press (1984) 86–96
37. Kuper, G.M., Vardi, M.Y.: The Logical Data Model. ACM TODS 18 (1993) 379–413
38. Shipman, D.W.: The Functional Data Model and the Data Language DAPLEX. ACM TODS 6 (1981) 140–173
39. Lécluse, C., Richard, P., Véléz, F.: O2, an Object-Oriented Data Model. In: Proc. of the 1988 ACM SIGMOD Intl. Conference on Management of Data, ACM Press (1988) 424–433
40. Hidders, J.: Typing Graph-Manipulation Operations. In: Proc. of the 9th ICDT, Springer-Verlag (2002) 394–409
41. Kunii, H.S.: DBMS with Graph Data Model for Knowledge Handling. In: Proc. of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow, IEEE (1987) 138–142
42. Amann, B., Scholl, M.: Gram: A Graph Data Model and Query Language. In: European Conference on Hypertext Technology, ACM Press (1992) 201–211
43. Güting, R.H.: GraphDB: Modeling and Querying Graphs in Databases. In: Proc. of 20th VLDB Conference, Morgan Kaufmann (1994) 297–308
44. Kiesel, N., Schurr, A., Westfechtel, B.: GRAS: A Graph-Oriented Software Engineering Database System. In: IPSEN Book. (1996) 397–425
45. Levene, M., Poulouvasilis, A.: The Hypernode Model and its Associated Query Language. In: Proc. of the 5th Jerusalem IT Conference, IEEE (1990) 520–530
46. Poulouvasilis, A., Levene, M.: A Nested-graph Model for the Representation and Manipulation of Complex Objects. ACM Transactions on Information Systems 12 (1994) 35–68
47. Consens, M., Mendelzon, A.: Hy+: A Hygraph-based Query and Visualization System. SIGMOD Rec. 22 (1993) 511–516
48. Cruz, I.F., Mendelzon, A.O., Wood, P.T.: A Graphical Query Language Supporting Recursion. SIGMOD Rec. 16 (1987) 323–330
49. Balmin, A., Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D., Wang, T.: A System for Keyword Proximity Search on XML Databases. In: Proc. of 29th VLDB Conference. (2003) 1069–1072
50. Consens, M.P., Mendelzon, A.O.: Expressing Structural Hypertext Queries in Graphlog. In: Proc. of the 2th ACM Conf. on Hypertext, ACM Press (1989) 269–292
51. Gemis, M., Paredaens, J.: An Object-Oriented Pattern Matching Language. In: Proc. 1th ISOTAS, Springer-Verlag (1993) 339–355

52. Alashqur, A.M., Su, S.Y.W., Lam, H.: OQL: A Query Language for Manipulating Object-oriented Databases. In: Proc. of the 15th VLDB Conference, Morgan Kaufmann (1989) 433–442
53. Flesca, S., Greco, S.: Partially Ordered Regular Languages for Graph Queries. In: Proceedings of the 26th ICALP. Volume 1644 of LNCS., Springer-Verlag (1999)
54. Flesca, S., Greco, S.: Querying Graph Databases. In: Proceedings of the 7th EDBT Conference. Volume 1777 of LNCS., Springer-Verlag (2000) 510–524
55. Seaborne, A.: RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004. <http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>
56. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. Proc. of the 1th ISWC (2002)
57. Berners-Lee, T.: Notation 3 - An RDF Language for the Semantic Web. <http://www.w3.org/DesignIssues/Notation3> (2001)
58. Clark, K.G.: RDF Data Access Use Cases and Requirements, W3C Working Draft. <http://www.w3.org/TR/rdf-dawg-uc/> (2004)
59. Angles, R., Gutierrez, C., Hayes, J.: RDF Query Languages Need Support for Graph Properties. Technical Report TR/DCC-2004-3, Department of Computer Science, University of Chile (2004)
60. Agrawal, R., Jagadish, H.V.: Algorithms for Searching Massive Graphs. IEEE TKDE 6 (1994) 225–238
61. Agrawal, R., Jagadish, H.V.: Materialization and Incremental Update of Path Information. In: Proc. of the 5th ICDE, IEEE Computer Society (1989) 374–383
62. Agrawal, R., Jagadish, H.V.: Efficient Search in Very Large Databases. In: Proc. of the 14th VLDB Conference. (1988) 407–418
63. Guha, R.V., Lassila, O., Miller, E., Brickley, D.: Enabling Inferencing. The Query Languages Workshop (1998)
64. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and Applications of Tree and Graph Searching. In: Proc. of the 21th ACM PODS, ACM Press (2002) 39–52
65. Yannakakis, M.: Graph-theoretic Methods in Database Theory. In: Proc. of the 9th ACM PODS, ACM Press (1990) 230–242
66. Mannino, M.V., Shapiro, L.D.: Extensions to Query Languages for Graph Traversal Problems. IEEE TKDE 2 (1990) 353–363
67. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. In: Proc. of the 15th VDLB Conference, Morgan Kaufmann (1989) 185–193
68. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record 26 (1997) 54–66
69. Anyanwu, K., Sheth, A.: The ρ -operator: Enabling Querying for Semantic Associations on the Semantic Web. In: The 12th WWW Conference. (2003)
70. Washio, T., Motoda, H.: State of the Art of Graph-based Data Mining. SIGKDD Explor. Newsl. 5 (2003) 59–68
71. Hidders, J., Paredaens, J.: GOAL, A Graph-Based Object and Association Language. CISM - Advances in Database Systems 1993 (1993) 247–265
72. Cruz, I.F., Mendelzon, A.O., Wood, P.T.: G+: Recursive Queries without Recursion. In: Proc. of the 2th International Conference on Expert Database Systems, Addison-Wesley (1989) 645–666
73. Sayers, C.: Node-centric RDF Graph Visualization. Technical Report HPL-2004-60, HP Laboratories (2004)
74. Guha, R., McCool, R., Miller, E.: Semantic search. In: Proc. of the 12th WWW conference, ACM Press (2003) 700–709

DRAGO: Distributed Reasoning Architecture for the Semantic Web^{*}

Luciano Serafini¹ and Andrei Taminlin²

¹ ITC-IRST, Trento 38050, Italy
luciano.serafini@itc.it

² DIT, University of Trento, Trento 38050, Italy
andrei.taminlin@dit.unitn.it

Abstract. The paper addresses the problem of reasoning with multiple ontologies interconnected by semantic mappings. This problem is becoming more and more relevant due to the necessity of building the interoperable Semantic Web. In contrast to the so called global reasoning approach, in this paper we propose a *distributed reasoning technique* that accomplishes reasoning through a combination of local reasoning chunks, internally executed in each separate ontology. Using Distributed Description Logics as a formal framework for representation of multiple semantically connected ontologies, we define a sound and complete distributed tableau-based reasoning procedure which is built as an extension to standard Description Logic tableau. Finally, the paper describes the design and implementation principles of a distributed reasoning system, called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), that implements such distributed decision procedure.

1 Introduction

The number of ontologies appearing on the Web is growing steadily. Each ontology describes a domain of interest from a subjective perspective and level of granularity. This fact inevitably leads to a *heterogeneity* between ontologies describing even the very same domain. As a consequence, making multiple heterogeneous ontologies *interoperate*, is becoming a significant problem on the Semantic Web.

The common approach for supporting ontology interoperability is based on the definition of semantic relations between entities belonging to different ontologies, called a *semantic mapping*. A simple example of semantic mapping is the one stating that the concept **Student** in one ontology is more specific than the concept **Person** of another ontology.

^{*} We thank Alexander Borgida for very inspiring discussions on the DDL framework. We also grateful to Fausto Giunchiglia, Maxym Mykhalchuk and Yuting Zhao for discussions about C-OWL.

Several proposals of languages for expressing semantic mappings have been done so far. Some of them have a well-defined formal semantics, for example C-OWL [3], $\mathcal{E}$ -connected OWL [9]. Examples of less formally grounded proposals are RDF Transformation [18] and MAFRA Semantic Bridge Ontology [16].

However, semantic mappings are not enough to guarantee ontology interoperability. One has also to provide the capability of *reasoning* within a system comprised of multiple ontologies interconnected by semantic mappings. So far, the reasoning approach dominating on the current Semantic Web rephrases the problem of reasoning with multiple interconnected ontologies into a problem of reasoning in a *global* ontology that encodes both ontologies and mappings into a unique blob. This approach, however, brings a number of drawbacks, such as (i) non-scalability, (ii) loosing language and reasoning specificity of distinct ontologies, (iii) losing privacy and autonomy of ontological knowledge.

In this paper, we suggest an alternative approach which is based on the contextual reasoning paradigm. Namely, the reasoning with multiple ontologies is proposed to be accomplished through a suitable combination, via semantic mappings, of local reasoning chunks, internally executed in each distinct ontology. In a nutshell, we propose a distributed tableau algorithm, which is capable of checking concept satisfiability in a set of interconnected ontologies by combining local (standard) tableaux procedures that check satisfiability inside of each single ontology. This first proposal focuses on ontologies which can be expressed in the *SHIQ* fragment of Description Logic [14]. The suggested decision procedure is sound and complete w.r.t. Distributed Description Logics [2], the framework used to represent multiple semantically connected ontologies.

In comparison to the global approach, the proposed distributed reasoning technique is more scalable, since the reasoning process is performed in a partitioned search space and propagates through semantic mappings, which are used to guide the search. It respects privacy and supports information hiding by requiring access to *local reasoning services* rather than the direct access to ontologies. Finally, it supports languages specificity, since it combines different local reasoning procedures, each of which can be tailored on the local ontology language.

The distributed tableaux proposed in this paper has been implemented in a system called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies). DRAGO represents a peer-to-peer like architecture in which every peer registers a set of ontologies and provides reasoning services for them. The key issue of DRAGO is that it supports the assignment of semantic mappings to the registered ontologies and performs reasoning with such ontologies coupled with semantic mappings in a distributed manner, i.e. using local reasoner for ontology and by coordinating with other reasoners, via mappings, when local ontology is semantically connected with other ontologies.

The paper is structured as follows. In the first part we recall the Distributed Description Logics framework and enunciate the main properties. In Section 3 we describe the abstract distributed tableau algorithm that computes subsumption in DDL. In Section 4 we describe the ongoing work on DRAGO system and then

in Section 5 compare DRAGO with other approaches and systems relevant to reasoning with multiple distributed ontologies.

2 Distributed Description Logics

Description Logic (DL) has been advocated as the suitable formal tool to represent and reason about ontologies. Distributed Description Logics (DDL) [2] is a *natural* generalization of the DL framework designed to formalize multiple ontologies interconnected by semantic mappings. In this section we briefly recall the definitions of DDL.

As defined by Borgida and Serafini in [2], *Distributed Description Logics* provides a syntactical and semantical framework for formalization of multiple ontologies *pairwise* linked by semantic mappings. In DDL, ontologies correspond to description logic theories (T-boxes), while semantic mappings correspond to collections of *bridge rules* ($\mathfrak{B}$).

Given a non empty set I of indexes, used to identify ontologies, let $\{\mathcal{DL}_i\}_{i \in I}$ be a collection of description logics¹. For each $i \in I$ let us denote a T-box of $\mathcal{DL}_i$ as $\mathcal{T}_i$. In this paper, we assume that each $\mathcal{DL}_i$ is description logic weaker or at most equivalent to $\mathcal{SHIQ}$. Thus a T-box will contain all the information necessary to define the terminology of a domain, including not just concept and role definitions, but also general axioms relating descriptions, as well as declarations such as the transitivity of certain roles.

We call $\mathbf{T} = \{\mathcal{T}_i\}_{i \in I}$ a family of T-Boxes indexed by I . Intuitively, $\mathcal{T}_i$ is the description logic formalization of the i -th ontology. To make every description distinct, we will prefix it with the index of ontology it belongs to. For instance, the concept C that occurs in the i -th ontology is denoted as $i : C$. Similarly, $i : C \sqsubseteq D$ denotes the fact that the axiom $C \sqsubseteq D$ is being considered in the i -th ontology.

Semantic mappings between different ontologies are expressed via collections of *bridge rules*.

Definition 1 (Bridge rules). *A bridge rule from i to j is an expression of the following two forms:*

1. $i : A \xrightarrow{\sqsupseteq} j : G$, onto-bridge rule
2. $i : B \xrightarrow{\sqsubseteq} j : H$, into-bridge rule

where A, B and G, H are concepts of $\mathcal{DL}_i$ and $\mathcal{DL}_j$ respectively².

Bridge rules do not represent semantic relations stated from an external *objective* point of view. Indeed, there is no such global view on the Web. Instead,

¹ We assume familiarity with Description Logic and related reasoning systems, described in [4].

² This is a restricted case of bridge rules w.r.t. definition in [2].

bridge rules from i to j express relations between i and j viewed from the *subjective* point of view of the j -th ontology.

Intuitively, the into-bridge rule $i : B \xrightarrow{\sqsubseteq} j : H$ states that, from the j -th point of view the concept B in i is less general than its local concept H . Similarly, the onto-bridge rule $i : A \xrightarrow{\supseteq} j : G$ expresses the fact that, according to j , A in i is more general than G in j . Therefore, bridge rules from i to j provide the possibility of translating into j 's ontology (under some approximation) the concepts of a foreign i 's ontology. Note, that since bridge rules reflect a subjective point of view, bridge rules from j to i are not necessarily the inverse of the rules from i to j , and in fact there may be no rules in one or both the directions.

Example 1. From on-line DAML ontology library we have selected two small and largely overlapping ontologies. First, the Semantic Web research community ontology (SWRC)³ that models the research community, its researches, topics, publications, etc. Second, a DAML version of SHOE ontology for describing universities and the activities that occur at them⁴. Figure 1 shows extracts of the

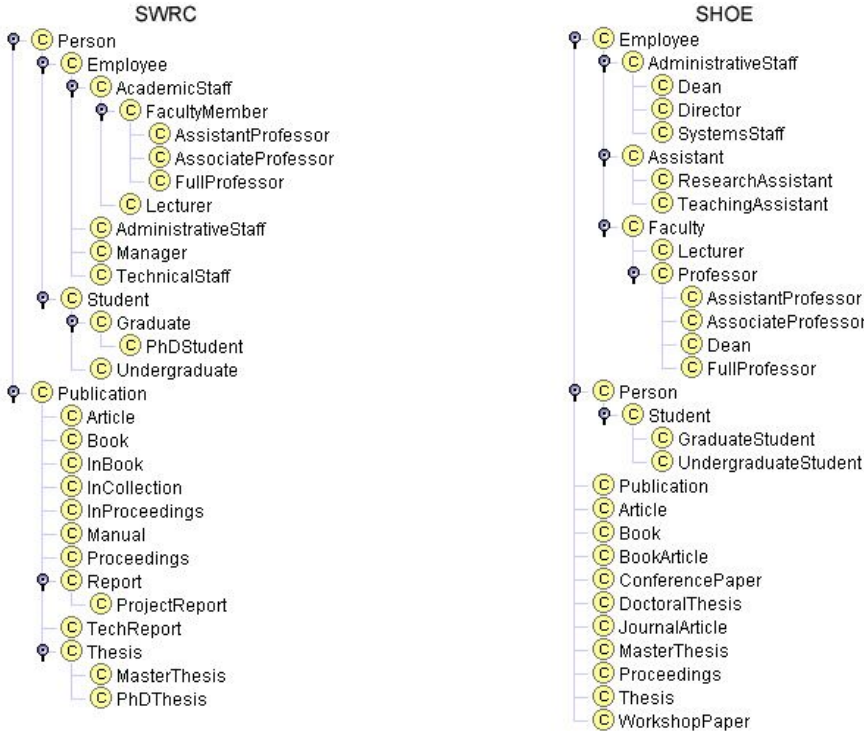


Fig. 1. Extracts of the class hierarchies of SWRC and SHOE

³ <http://www.semanticweb.org/ontologies/swrc-onto-2000-09-10.daml>
⁴ <http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml>

class hierarchies of these two ontologies. Note, that for the sake of demonstrating the value of mappings, we considered oversimplified SHOE ontology without imports.

The following are examples of bridge rules from SWRC to SHOE.

$$\text{SWRC} : \text{Article} \xrightarrow{\exists} \text{SHOE} : \text{ConferencePapers} \quad (1)$$

$$\text{SWRC} : \text{Article} \xrightarrow{\sqsubseteq} \text{SHOE} : \text{Article} \quad (2)$$

$$\text{SWRC} : \text{Article} \xrightarrow{\exists} \text{SHOE} : \text{Article} \quad (3)$$

$$\text{SWRC} : \text{PhDStudent} \xrightarrow{\exists} \text{SHOE} : \text{GraduateStudent} \quad (4)$$

You can see a richer set of possible bridge rules between OWL encodings of SWRC and SHOE ontologies⁵. We have defined these bridge rules manually, but in many cases bridge rules can be produced by a (semi-)automatic process.

Definition 2 (Distributed T-box). *A distributed T-box (DTBox)*

$\mathfrak{T} = \langle \{\mathcal{T}_i\}_{i \in I}, \mathfrak{B} \rangle$ consists of a collection of T-boxes $\{\mathcal{T}_i\}_{i \in I}$, and a collection of bridge rules $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i \neq j \in I}$ between them.

The semantics of DDL is the customization of Local Models Semantics for Multi Context Systems [5, 20]. The basic idea is that each ontology $\mathcal{T}_i$ is *locally interpreted* on a *local domain*. The first component of the semantics of a DTBox is therefore a family of interpretations $\{\mathcal{I}_i\}_{i \in I}$, one for each T-box $\mathcal{T}_i$. Each $\mathcal{I}_i$ is called a *local interpretation* and consists of *possibly empty* domain $\Delta^{\mathcal{I}_i}$ and a valuation function $\cdot^{\mathcal{I}_i}$, which maps every concept to a subset of $\Delta^{\mathcal{I}_i}$, every role to a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$. The interpretation on the empty domain is denoted with the apex ϵ .

Notice that, in DL, interpretations are defined always on a non empty domain. Therefore $\mathcal{I}^\epsilon$ is not an interpretation in DL. In DDL however we need to provide a semantics for *partially inconsistent* distributed T-boxes, i.e. DTBoxes in which some of the local T-boxes are inconsistent. $\mathcal{I}^\epsilon$ provides an “impossible interpretation” which can be associated to inconsistent T-boxes. Indeed, $\mathcal{I}^\epsilon$ satisfies every axiom $X \sqsubseteq Y$ (also $\top \sqsubseteq \perp$) since $X^{\mathcal{I}^\epsilon} = \emptyset$ for every concept and role X .

The second component of the DDL semantics is the family of domain relations.

Definition 3 (Domain relation). *A domain relation r_{ij} from $\Delta^{\mathcal{I}_i}$ to $\Delta^{\mathcal{I}_j}$ is a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$. We use $r_{ij}(d)$ to denote $\{d' \in \Delta^{\mathcal{I}_j} \mid \langle d, d' \rangle \in r_{ij}\}$; for any subset D of $\Delta^{\mathcal{I}_i}$, we use $r_{ij}(D)$ to denote $\bigcup_{d \in D} r_{ij}(d)$; for any $R \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$ we use $r_{ij}(R)$ to denote $\bigcup_{\langle d, d' \rangle \in R} r_{ij}(d) \times r_{ij}(d')$.*

Domain relation r_{ij} does not represent a semantic mapping seen from an external objective point of view. Rather, it represents a possible way of mapping

⁵ <http://trinity.dit.unitn.it/drago/examples/eswc05/swrc-shoe.cowl>

the elements of $\Delta^{\mathcal{I}_i}$ into its domain $\Delta^{\mathcal{I}_j}$, seen from j 's perspective. For instance, if $\Delta^{\mathcal{I}_1}$ and $\Delta^{\mathcal{I}_2}$ are the representation of time as Rationals and as Naturals, r_{ij} could be the round off function, or some other approximation relation.

Definition 4 (Distributed interpretation). A distributed interpretation $\mathfrak{I} = \langle \{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I} \rangle$ of a DTBox $\mathfrak{T}$ consists of local interpretations $\mathcal{I}_i$ for each $\mathcal{I}_i$ on local domains $\Delta^{\mathcal{I}_i}$, and a family of domain relations r_{ij} between these local domains.

Definition 5. A distributed interpretation $\mathfrak{I}$ satisfies the elements of a DTBox $\mathfrak{T}$ according to the following clauses: for every $i, j \in I$

1. $\mathfrak{I} \models i : A \sqsubseteq B$, if $\mathcal{I}_i \models A \sqsubseteq B$
2. $\mathfrak{I} \models \mathcal{T}_i$, if $\mathfrak{I} \models i : A \sqsubseteq B$ for all $A \sqsubseteq B$ in $\mathcal{T}_i$
3. $\mathfrak{I} \models i : x \xrightarrow{\sqsubseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \subseteq y^{\mathcal{I}_j}$
4. $\mathfrak{I} \models i : x \xrightarrow{\supseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \supseteq y^{\mathcal{I}_j}$
5. $\mathfrak{I} \models \mathfrak{B}_{ij}$, if $\mathfrak{I}$ satisfies all bridge rules in $\mathfrak{B}_{ij}$
6. $\mathfrak{I} \models \mathfrak{T}$, if for every $i, j \in I$, $\mathfrak{I} \models \mathcal{T}_i$ and $\mathfrak{I} \models \mathfrak{B}_{ij}$

Definition 6 (Distributed Entailment and Satisfiability). $\mathfrak{T} \models i : C \sqsubseteq D$ (read as “ $\mathfrak{T}$ entails $i : C \sqsubseteq D$ ”) if for every $\mathfrak{I}$, $\mathfrak{I} \models \mathfrak{T}$ implies $\mathfrak{I} \models_d i : C \sqsubseteq D$.

$\mathfrak{T}$ is satisfiable if there exists a $\mathfrak{I}$ such that $\mathfrak{I} \models \mathfrak{T}$. Concept $i : C$ is satisfiable with respect to $\mathfrak{T}$ if there is a $\mathfrak{I}$ such that $\mathfrak{I} \models \mathfrak{T}$ and $C^{\mathcal{I}_i} \neq \emptyset$.

Some important properties of DDL are listed below:

Monotonicity. Bridge rules do not obstruct local subsumptions. Formally:

$$\mathcal{T}_i \models A \sqsubseteq B \implies \mathfrak{T} \models i : A \sqsubseteq B \quad (5)$$

Directionality. T-box without incoming bridge rules is not affected by other T-boxes. Formally, if $\mathfrak{B}_{ki} = \emptyset$ for any $k \neq i \in I$, then:

$$\mathfrak{T} \models i : A \sqsubseteq B \implies \mathcal{T}_i \models A \sqsubseteq B \quad (6)$$

Simple subsumption propagation. A combination of onto- and into-bridge rules allows the propagation of the subsumptions across ontologies. For example, if $\mathfrak{B}_{ij}$ contains $i : A \xrightarrow{\supseteq} j : G$ and $i : B \xrightarrow{\sqsubseteq} j : H$, then:

$$\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathfrak{T} \models j : G \sqsubseteq H \quad (7)$$

Generalized subsumption propagation. If $\mathfrak{B}_{ij}$ contains $i : A \xrightarrow{\supseteq} j : G$ and $i : B_k \xrightarrow{\sqsubseteq} j : H_k$ for $1 \leq k \leq n$ and $n \geq 0$, then:

$$\mathfrak{T} \models i : A \sqsubseteq \bigsqcup_{k=1}^n B_k \implies \mathfrak{T} \models j : G \sqsubseteq \bigsqcup_{k=1}^n H_k \quad (8)$$

(Notationally, $\bigsqcup_{k=1}^0 D_k$ denotes $\perp$.)

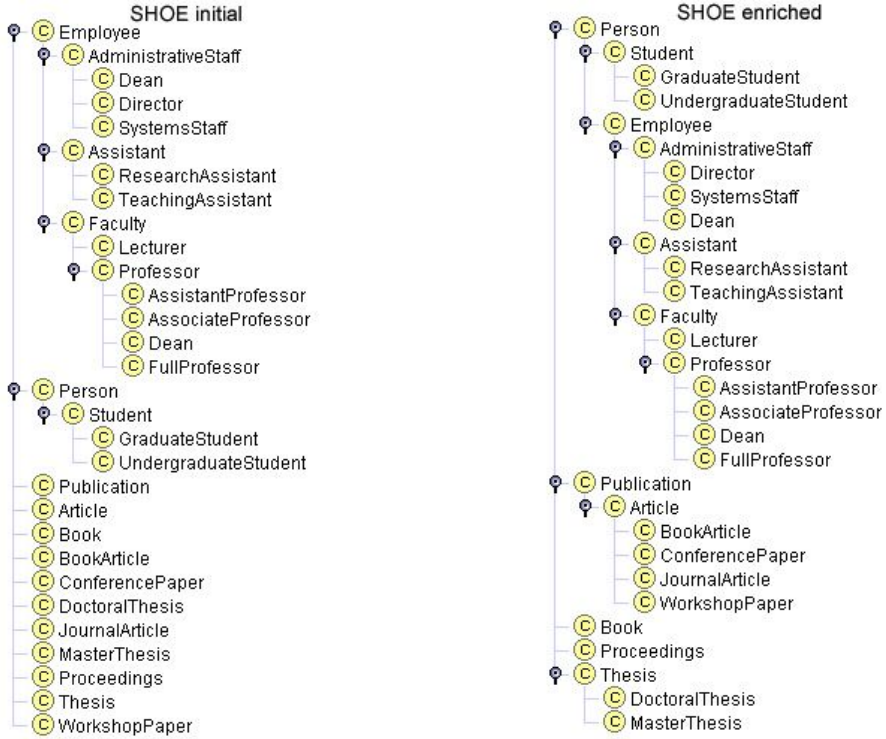


Fig. 2. Initial and enriched via bridge rules hierarchy of SHOE

We would like to stress the importance of subsumption propagation property since it constitutes a main reasoning pattern in DDL. For the full set of DDL properties and formal proofs we refer reader to a technical report [19].

Example 2. Taking the bridge rules from Example 1 and applying the subsumption propagation property we can infer in the hierarchy SHOE that *ConferencePaper* is a subclass of *Article*, i.e. that $\text{SHOE} : \text{ConferencePaper} \sqsubseteq \text{Article}$.

Figure 2 shows how the initial SHOE hierarchy can be enriched (without its modification) via the whole set of possible bridge rules mentioned in Example 1.

3 Distributed Tableau for Reasoning in DDL

Although both in DL and DDL the fundamental reasoning task lays in a verification of concepts subsumption, in DDL besides the ontology itself the subsumption depends also on other ontologies that affect it through the semantic mappings. In this section we investigate a decision procedure that computes DDL subsumption and propose a distributed tableau reasoning algorithm for determining whether $\mathcal{T} \models i : A \sqsubseteq B$.

In order to get the intuition of the algorithm, let us first present an example with some simplifying assumptions. Later on, we relax these assumptions and extend our results to a more general case.

Example 3. Consider a distributed T-box $\mathfrak{T}_{12} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$ with only two T-boxes and unidirectional bridge rules between them. Suppose that $\mathcal{T}_1$ contains axioms $\text{Student} \sqsubseteq \text{Person}$ and $\text{Pianist} \sqsubseteq \text{Musician}$, $\mathcal{T}_2$ does not contain any axiom and $\mathfrak{B}_{12}$ contains the following bridge rules:

$$1 : \text{Person} \xrightarrow{\sqsubseteq} 2 : \text{Agent} \quad 1 : \text{Musician} \xrightarrow{\sqsubseteq} 2 : \text{Artist} \quad (9)$$

$$1 : \text{Student} \xrightarrow{\sqsupseteq} 2 : \text{Graduate} \quad 1 : \text{Pianist} \xrightarrow{\sqsupseteq} 2 : \text{JazzPianist} \quad (10)$$

Let us show that $\mathfrak{T}_{12} \models 2 : \text{Graduate} \sqcap \text{JazzPianist} \sqsubseteq \text{Agent} \sqcap \text{Artist}$, i.e. that for any distributed interpretation $\mathfrak{J} = \langle \mathcal{I}_1, \mathcal{I}_2, r_{12} \rangle$, $(\text{Graduate} \sqcap \text{JazzPianist})^{\mathcal{I}_2} \sqsubseteq (\text{Agent} \sqcap \text{Artist})^{\mathcal{I}_2}$.

1. Suppose that by contradiction there is an $x \in \Delta_2$ such that $x \in (\text{Graduate} \sqcap \text{JazzPianist})^{\mathcal{I}_2}$ and $x \notin (\text{Agent} \sqcap \text{Artist})^{\mathcal{I}_2}$.
2. Then $x \in \text{Graduate}^{\mathcal{I}_2}$, $x \in \text{JazzPianist}^{\mathcal{I}_2}$, and either $x \notin \text{Agent}^{\mathcal{I}_2}$ or $x \notin \text{Artist}^{\mathcal{I}_2}$.
3. Let us consider the case where $x \notin \text{Agent}^{\mathcal{I}_2}$. From the fact that $x \in \text{Graduate}^{\mathcal{I}_2}$, by the bridge rule (10), there is $y \in \Delta^{\mathcal{I}_1}$ with $\langle y, x \rangle \in r_{12}$, such that $y \in \text{Student}^{\mathcal{I}_1}$.
4. From the fact that $x \notin \text{Agent}^{\mathcal{I}_2}$, by bridge rule (9), we can infer that for all $y \in \Delta^{\mathcal{I}_1}$ if $\langle y, x \rangle \in r_{12}$ then $y \notin \text{Person}^{\mathcal{I}_1}$.
5. But, since $\text{Student} \sqsubseteq \text{Person} \in \mathcal{T}_1$, then $y \in \text{Person}^{\mathcal{I}_1}$, and this is a contradiction.
6. The case where $x \notin \text{Artist}^{\mathcal{I}_2}$ is analogous.

The above reasoning steps can be seen as a combination of a tableau **Tab**₂ in $\mathcal{T}_2$ with a tableau **Tab**₁ in $\mathcal{T}_1$ as it is illustrated in Figure 3.

Let us formalize the above example.

Definition 7 (Bridge operator). *Given a set of bridge rules $\mathfrak{B}_{12}$ from $\mathcal{DL}_1$ to $\mathcal{DL}_2$, the operator $\mathfrak{B}_{12}(\cdot)$, taking as input a T-box in $\mathcal{DL}_1$ and producing a T-box in $\mathcal{DL}_2$, is defined as follows:*

$$\mathfrak{B}_{12}(\mathcal{T}_1) = \left\{ G \sqsubseteq \bigsqcup_{k=1}^n H_k \left| \begin{array}{l} \mathcal{T}_1 \models A \sqsubseteq \bigsqcup_{k=1}^n B_k, \\ 1 : A \xrightarrow{\sqsupseteq} 2 : G \in \mathfrak{B}_{12}, \\ 1 : B_k \xrightarrow{\sqsubseteq} 2 : H_k \in \mathfrak{B}_{12}, \\ \text{for } 1 \leq k \leq n, n \geq 0 \end{array} \right. \right\}$$

(Again notationally, we stipulate that $\bigsqcup_{k=1}^0 D_k$ denotes $\perp$.)

Theorem 1 (Soundness and completeness). *Let $\mathfrak{T}_{12} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$ be a distributed T-box, then:*

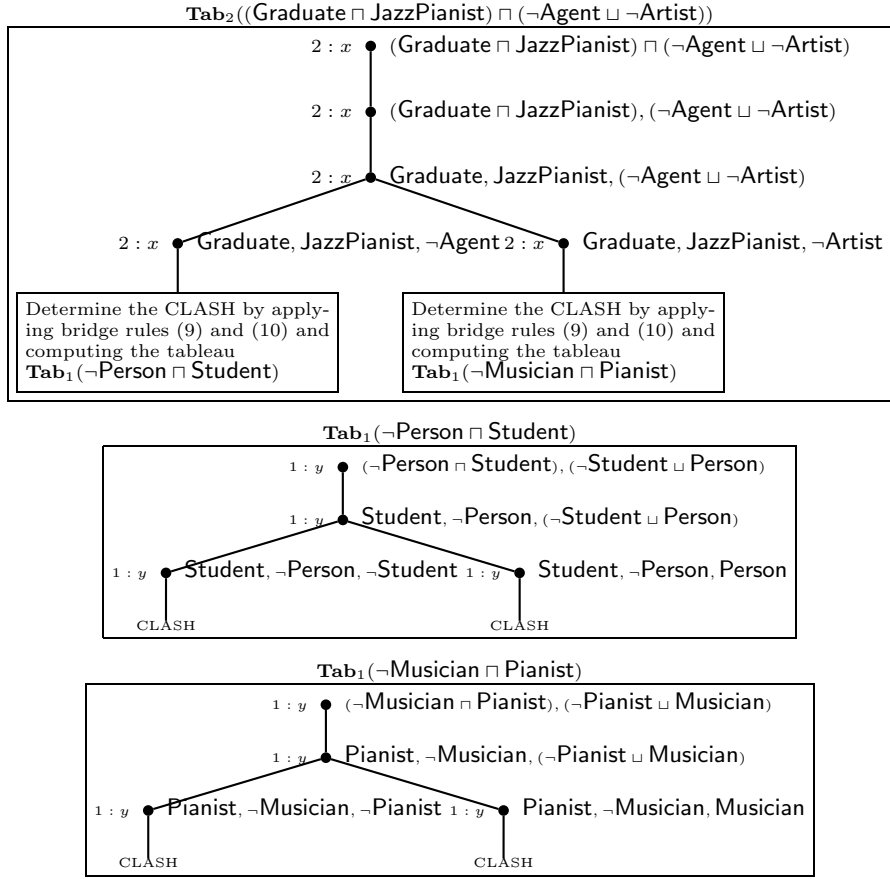


Fig. 3. Illustration of reasoning combination for the DDL subsumption

$$\mathfrak{T}_{12} \models 2 : X \sqsubseteq Y \iff \mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{T}_1) \models X \sqsubseteq Y \quad (11)$$

For the formal proof of Theorem 1 we refer reader to a technical report [19].

The main message of Theorem 1 is that in DDL the decision whether $\mathfrak{T}_{12} \models 2 : X \sqsubseteq Y$ can be *correctly* and *completely* rephrased into a *standard* DL subsumption in $\mathcal{T}_2$ extended by application of the bridge operator $\mathfrak{B}_{12}(\cdot)$. Due to that, the main computational task of DDL subsumption algorithm is to calculate the application of the bridge operator.

Theorem 1 can be generalized to the case of an *acyclic distributed T-box*, i.e. any $\mathfrak{T}$ in which the set of indexes I is a partial order $\langle I, < \rangle$ such that $i < j$ if and only if $\mathfrak{B}_{ij} \neq \emptyset$. Generalized version of the DDL subsumption algorithm represents a *backward-chaining* method that checks standard subsumption in a T-box $\mathcal{T}_i$ extended by applying bridge operators to the T-boxes which affect $\mathcal{T}_i$ via bridge rules.

3.1 Description of the Algorithm

Similarly to description logics reduction of subsumption to unsatisfiability, we rephrase the problem of deciding whether $\mathfrak{T} \models i : A \sqsubseteq B$ into the problem of not finding a distributed interpretation $\mathcal{I}$ of $\mathfrak{T}$, such that $(A \sqcap \neg B)^{\mathcal{I}_i} \neq \emptyset$.

Given an *acyclic distributed T-box* $\mathfrak{T} = \langle \{\mathcal{T}_i\}_{i \in I}, \{\mathfrak{B}_{ij}\}_{i \neq j \in I} \rangle$, we suppose to have a set of procedures **Tab_j**, one for each $j \in I$. Each **Tab_j** is composed of a set of standard *SHIQ*-expansion rules⁶.

On top of each procedure **Tab_j** we define a distributed tableau procedure **DTab_j**, one for each $j \in I$. The function **DTab_j** takes as an input a concept D and tries to build a representation of $\mathcal{I}_j$ with $D^{\mathcal{I}_j} \neq \emptyset$ (called a *completion tree*), using the expansion rules of **Tab_j**, plus an additional “bridge” expansion rule:

B_{ij}-rule

if 1. $G \in \mathcal{L}(x)$, $i : A \xrightarrow{\sqsupseteq} j : G \in \mathfrak{B}_{ij}$, $i : B_k \xrightarrow{\sqsubseteq} j : H_k \in \mathfrak{B}_{ij}$ for $1 \leq k \leq n$, $n \geq 0$, and 2. DTab_i $(A \sqcap \neg(\sqcup \mathbf{B})) = \text{Unsatisfiable}$ for some $\mathbf{H} \not\sqsubseteq \mathcal{L}(x)$, then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\sqcup \mathbf{H}\}$

The idea behind the **DTab** procedures is inspired by the bridge operator given in Definition 7. To verify whether the concept D is satisfiable in a T-box $\mathcal{T}_j$ of acyclic distributed T-box $\mathfrak{T}$, we invoke the corresponding distributed procedure **DTab_j**. First, it applies local tableaux rules of **Tab_j** in order to build a local completion tree. Each node x introduced during creation of the completion tree is labeled with a function $\mathcal{L}(x)$ containing concepts that x must satisfy. Whenever **DTab_j** encounters a node x in the completion tree such that it contains a label G , which is a consequence of an onto-bridge rule, then if $G \sqsubseteq \sqcup \mathbf{H}$ is entailed by the bridge rules, the label $\sqcup \mathbf{H}$ is added to x . To determine if $G \sqsubseteq \sqcup \mathbf{H}$ is entailed by bridge rules $\mathfrak{B}_{ij}$, **DTab_j** invokes **DTab_i** on the satisfiability of the concept $A \sqcap \neg(\sqcup \mathbf{B})$. In its turn, **DTab_i** will build independently from **DTab_j** an interpretation $\mathcal{I}_i$.

The proposed algorithm has several limitations. It admits acyclic distributed T-boxes without individuals and only allows bridge rules connecting atomic concepts. Despite these restrictions, we see the main advantage of the algorithm in the simplicity of its implementation.

4 DRAGO Reasoning System

In this section we will describe a design and implementation principles that lay in the base of DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), the system for reasoning with multiple ontologies connected by pairwise semantic mappings.

⁶ See [14] for more details about *SHIQ*-tableau.

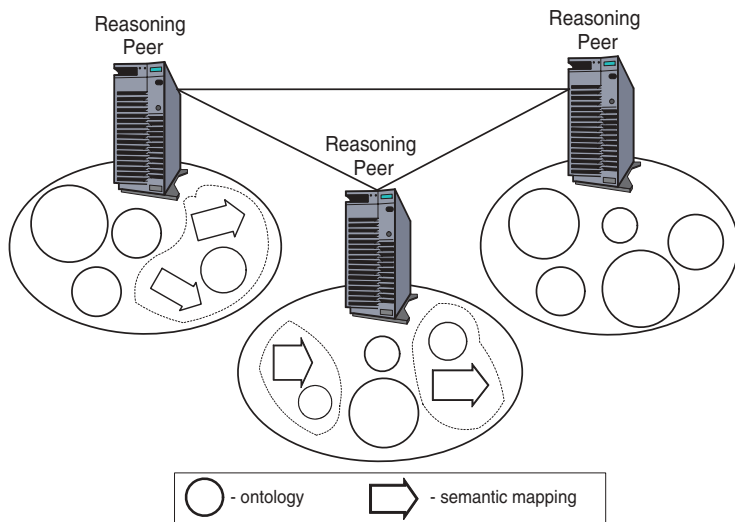


Fig. 4. DRAGO vision

4.1 Vision

As depicted in Figure 5, DRAGO envisages a Web of ontologies being distributed amongst a peer-to-peer network of *DRAGO Reasoning Peers* (DRP).

The role of a DRP is to provide reasoning services for ontologies registered to it, as well as to request reasoning services of other DRPs when this is required for fulfillment of distributed reasoning algorithm. The key issue of the DRP is that it provides possibility to register not just a stand alone ontology, but an ontology coupled with a set of semantic mappings.

In order to register an ontology to a DRP, the users specify a logical identifier for it, a Unified Resource Identifier (URI), and give a physical location of ontology on the Web, a Unified Resource Locator (URL). Besides that, it is possible to assign to an ontology a set of semantic mappings, providing in the same manner their location on the Web. As we discussed in the previous sections, attaching mappings to ontology enriches its knowledge due to the subsumption propagation mechanism. To prevent the possibility of attaching malicious mappings that can obstruct or falsify reasoning services, only the user that registered the ontology is allowed to add mappings to it.

When users or applications want to perform reasoning with a one of registered ontologies, they refer to the corresponding DRP and invoke its reasoning services giving the URI of the desired ontology.

4.2 Architecture

A DRP constitutes the basic element of DRAGO. The major components of a DRP are depicted in Figure 5.

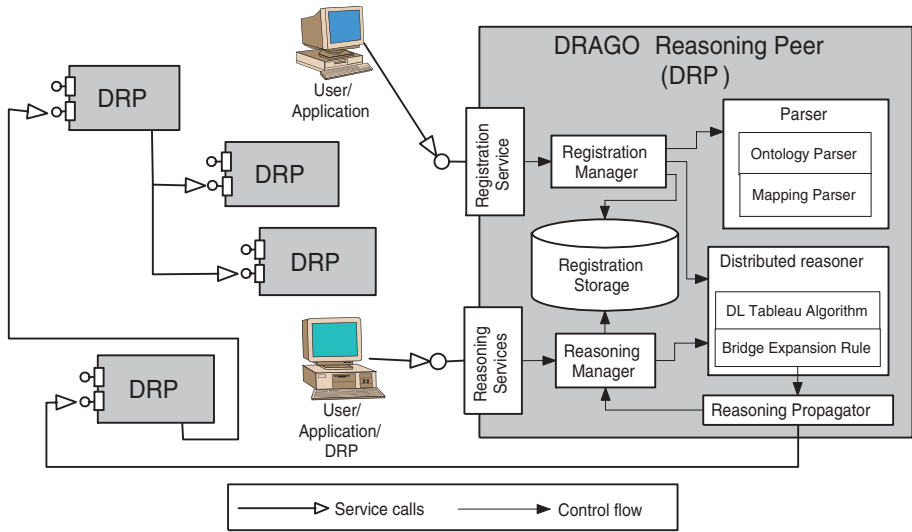


Fig. 5. DRAGO architecture

A DRP has two interfaces which can be invoked by users or applications:

- A *Registration Service* interface is meant for creating/modifying/deleting of registrations of ontologies and mappings assigned to them.
- A *Reasoning Services* interface enables the calls of reasoning services for registered ontologies. Among the reasoning services can be a possibility to check ontology consistency, build classification, verify concepts satisfiability and check entailment.

All accessibility information about registered ontologies and mappings is stored by a DRP in its local *Registration Storage*.

In order to register an ontology with a collection of semantic mappings attached to it (both available on the Web) a user or application invokes the Registration Service of a DRP and sends to it the following registration information:

- URI to which the ontology will be bound.
- URLs of ontology and semantic mappings attached to it, if any.
- If the semantic mappings connect this ontology with ontologies registered to external DRPs then additionally the URLs of these DRPs should be specified. This requirement is explained by the necessity to know who is responsible for reasoning with these ontologies.

The Registration Service interface is implemented by the *Registration Manager*. When the Manager receives a registration request, it (i) consults the Registration Storage and verifies if the URI has not occupied yet, (ii) if not it accesses ontologies and assigned mappings from their URLs, (iii) asks Parser component to process them, (iv) initializes the Distributed Reasoner with the parsed data, and (v) finally adds a new record to the Registration Storage.

The *Parser* component translates ontologies and mappings source files to the internal format used by the Distributed Reasoner. For doing so, the Parser consist from two sub components: the ontology parser, tailored on ontology language formats (for example, OWL [13]), and the mapping parser, tailored on mapping formats (for example, C-OWL [3]).

The *Reasoning Manager* component implements the Reasoning Services interface. When users, applications or other DRPs invoke this interface sending the URI of requested ontology, the Manager verifies with the Registration Storage whether the URI is registered to the DRP and, if yes, asks the Distributed Reasoner to execute corresponding reasoning task for that ontology.

The *Distributed Reasoner* represents a brain of a DRP. It realizes the distributed algorithm proposed in the Section 3 and reasons on ontologies with attached mappings that are registered to the DRP. The Distributed Reasoner is built on top of standard tableau reasoner whose algorithm was extended with the additional Bridge Expansion Rule in accordance with the distributed tableau algorithm. When the Bridge Expansion Rule is applied it analyses semantic mappings and possibly generates reasoning sub tasks that are required to be executed in the ontologies participating in mappings.

To dispatch the reasoning tasks generated by a Distributed Reasoner to the responsible reasoners, the *Reasoning Propagator* component refers to the Reasoning Manager and either dispatches reasoning to the local Distributed Reasoner or sends out a request of reasoning service to the corresponding external DRP.

4.3 Implementation

The described DRAGO architecture was implemented by us for the case of OWL [1] ontology space. For expressing semantic mappings between OWL ontologies we use a C-OWL [3]. According to C-OWL, mapping consists of references to the source and target ontologies and a series of bridge rules relating classes between these ontologies. Due to the limitations of introduced distributed tableau algorithm (see Section 3) among the possible C-OWL bridge rule types DRAGO supports the use of $\equiv$, $\sqsubseteq$, $\sqsupseteq$ rules connecting atomic concepts.

A Distributed Reasoner was implemented as an extension to an open source OWL reasoner Pellet⁷. Originally, Pellet parses OWL ontology to a Knowledge Base (T-box/A-box). To satisfy the needs of DRAGO we extended a Pellet's Knowledge Base with a M-box containing parsed C-OWL mappings. Another extension of Pellet was done by adding a Bridge Expansion Rule to the core tableau algorithm in order to transform it to the distributed tableau. This rule is called for every node created by the core tableau algorithm and consist in finding such bridge rules in M-box that are capable of importing new subsumptions from mapping-related ontologies. The distributed tableau algorithm was implemented in a straightforward way without advanced optimization techniques as, for example, caching.

⁷ <http://www.mindswap.org/2003/pellet>

DRAGO is implemented to operate over HTTP and to access ontologies and mappings published on the Web. A DRP represents several java servlets that should be deployed to a java-enabled Web-server, for example Tomcat⁸.

5 Related Work

From a theoretical perspective, presented work is an extension of the results introduced in [2, 21].

DDL inherited a lot of ideas from the other logics for distributed systems, among them Multi Context Systems (MCS) [7], the general framework for contextual reasoning, propositional MCS [8, 5, 20] and Distributed First Order Logics (DFOL) [6].

In [15], it has been shown that DDL can be represented in a much richer theoretical framework for integrating different logics, called $\mathcal{E}$ -connections. $\mathcal{E}$ -connections allow to state relations between a set of logical frameworks using *n-ary link relations*. Bridge rules can be seen as a special case of binary link relations. In this case, the satisfiability problem for DDL can be reduced to the satisfiability problem for basic $\mathcal{E}$ -connections.

The combined tableau algorithm for the restricted case of $\mathcal{E}$ -connections has been proposed recently in [9]. In contrast to the distributed tableau algorithm proposed in this paper, described combined tableau for $\mathcal{E}$ -connections is rather a selective global approach organized in a single reasoner, whereas in distributed tableau we combine different reasoning procedures for mapped ontologies.

The idea of having the system providing reasoning services for ontologies on the Semantic Web is not new. There are a number of reasoning servers based on the state of the art reasoners like RACER [10] or FaCT [12]. What makes DRAGO different from them is the capability of reasoning with ontologies coupled with semantic mappings using a distributed algorithm. While these reasoning servers are tightly connected with ontology repositories for achieving a higher level of optimization, DRAGO is a lightweight implementation which directly uses ontologies and mappings published on the Web.

Also from the practical point of view, DRAGO architecture can be related to a variety of systems for mediation and integration of distributed heterogeneous sources like Piazza [11], OBSERVER [17] and others.

6 Conclusions and Future Work

In this paper, we have introduced DRAGO, a system which provides reasoning services for multiple OWL ontologies interconnected via C-OWL mappings⁹.

⁸ <http://jakarta.apache.org/tomcat>

⁹ Demonstration version of DRAGO is available for download after the registration on the project home page <http://trinity.dit.unitn.it/drago>

The theoretical support of DRAGO is provided by the Distributed Description Logics framework. In this paper, we have described a sound and complete distributed tableau reasoning technique for DDL. According to it, a reasoning in DDL can be fulfilled by a suitable combination of existing local tableaux for Description Logics. Although the suggested reasoning algorithm has been considered for the limited case of DDL with acyclic distributed T-box without individuals and bridge rules connecting atomic concepts, we see a main benefit of the algorithm in its simplicity and easy implementation on top of existing tableau-based reasoning systems.

As promising paths for further research we plan to explore the caching techniques for improving the distributed algorithm, investigate the use of more expressive mappings connecting complex concepts, and extend our results for general distributed T-boxes.

References

1. G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In *Handbook on Ontologies in Information Systems*, pages 67–92, 2003.
2. A. Borgida and L. Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics*, pages 153–184, 2003.
3. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *Proc. of the 2d International Semantic Web Conference (ISWC2003)*, pages 164–179, 2003.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
5. C. Ghidini and F. Giunchiglia. Local Model Semantics, or Contextual Reasoning = Locality + Compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.
6. C. Ghidini and L. Serafini. Distributed First Order Logics. In *Proc. of the Frontiers of Combining Systems*, pages 121–139, 2000.
7. F. Giunchiglia. Contextual Reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993.
8. F. Giunchiglia and L. Serafini. Multilanguage Hierarchical Logics (or: How we can do without modal logics). *Artificial Intelligence*, 65(1):29–70, 1994.
9. B. C. Grau, B. Parsia, and E. Sirin. Working with Multiple Ontologies on the Semantic Web. In *Proc. of the 3d International Semantic Web Conference (ISWC2004)*, 2004.
10. V. Haarslev and R. Moller. RACER System Description. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR2001)*, pages 701–706, 2001.
11. A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proc. of the 12th International World Wide Web Conference (WWW 2003)*, 2003.
12. I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. of the Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, pages 27–30, 1998.
13. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

14. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for very Expressive Description Logics. *Logic Journal of IGPL*, 8(3):239–263, 2000.
15. O. Kutz, C. Lutz, F. Wolter, and M. Zakharyashev. $\mathcal{E}$ -connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.
16. A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - a Mapping Framework for Distributed Ontologies. In *Proc. of Knowledge Engineering and Knowledge Management (EKAW-02)*, volume 2473 of *Lecture Notes in Computer Science*. Springer, 2002.
17. E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. *International Journal on Distributed and Parallel Databases (DAPD)*, ISSN 0926-8782, 8(2):223–272, April 2000.
18. B. Omelayenko. RDFT: A Mapping Meta-Ontology for Business Integration. In *Proc. of the Workshop on Knowledge Transformation for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW2002)*, pages 77–84, 2002.
19. L. Serafini, A. Borgida, and A. Tamin. Distributed Reasoning in SHIQ Ontology Space. Technical Report T05-02-03, ITC-IRST, 2005.
20. L. Serafini and F. Giunchiglia. ML Systems: A Proof Theory for Contexts. *Journal of Logic, Language and Information*, 11(4):471–518, 2002.
21. L. Serafini and A. Tamin. Local Tableaux for Reasoning in Distributed Description Logics. In *Proc. of the 2004 International Workshop on Description Logics (DL'04)*, CEUR-WS, 2004.

Dually Structured Concepts in the Semantic Web: Answer Set Programming Approach

Patryk Burek and Rafał Graboś*

Department of Computer Science, University of Leipzig, Germany
{Burek, grabos}@informatik.uni-leipzig.de

Abstract. There is an ongoing discussion whether reasoning in the Semantic Web should be monotonic or not. However, it seems that the problem concerns not only reasoning over knowledge but knowledge itself, where apart from nondefeasible knowledge the defeasible knowledge can be distinguished. In the current paper we rely on the Dual Theory of Concepts, according to which concepts are dually structured into defeasible and nondefeasible parts. We develop a metaontology for representing both types of a concept's structure and apply it for annotating OWL axioms. The translation of annotated OWL axioms into a logic program under answer set semantics is provided. Hence the answer set solver Smodels may be used as reasoner for annotated ontologies, handling properly the distinction between monotonic and nonmonotonic reasoning.

Keywords: Ontology, Knowledge Representation, Reasoning in the Semantic Web, Semantic Web Inference Schemes

1 Introduction

Current standards and techniques used in the ontology and Semantic Web communities, like the Web Ontology Language (OWL) [25] and its underlying description logics (DL) with their reasoners, are basically monotonic. On the other hand there is an ongoing discussion whether reasoning in the Semantic Web should be monotonic or not. It seems however that the problem concerns not only the reasoning over the knowledge but the knowledge itself. The knowledge may be defeasible and as such may require nonmonotonic inferences on it. The main question then is what the knowledge is like and whether it is defeasible.

Since Wittgenstein [30] and Rosch [28] there is a debate in cognitive science concerning the prototypical structure of concepts. Both Wittgenstein and Rosch pointed out that concepts do not have to be specified by a set of necessary (and sufficient) conditions. They argued that often not all of a concept's referents share all of the properties assigned to the concept. Prototypically structured concepts then lack necessary characteristics but have only typical and therefore defeasible characteristics. On the other hand a prototypical concept structure raises several problems. For example it

* This paper was written fully collaboratively; the order of the authors' names is arbitrary.

is clear that not all concepts are fuzzy and prototypically structured but there are some that have at least necessary characteristics.

In the current paper we propose a framework for specifying concepts within ontologies based on the cognitive Dual Theory, according to which concepts are dually structured involving defeasible and nondefeasible parts.

To represent the structure of concepts and to delimit a defeasible part of a concept's definition from a nondefeasible one, we develop a simple metaontology and use it for annotating OWL axioms. The translation of annotated OWL axioms into a logic program under answer set semantics is provided. Hence the answer set solver *Smodels* [22] may be used as reasoner for annotated ontologies, handling the distinction between monotonic and nonmonotonic reasoning properly.

For simplicity of reading, we do not represent annotated OWL axioms in OWL abstract syntax [25]. Instead we use description logic syntax extended by the provided meta-tags. This is justified since there is a correspondence between the syntaxes of OWL and description logics, such that each element of OWL can be represented by elements of the corresponding description logic¹. However, one should note that the description logic syntax is used here for readability purpose only.

The paper is structured as follows. In the next section we introduce a running example. In the following two sections advantages and drawbacks of nonmonotonic solutions to the Semantic Web are presented. In the fifth section we present the foundations of our approach based on the cognitive Dual Theory. Moreover, in the fifth section the metaontology and the mechanism of annotating OWL axioms are presented. In the sixth section the translation algorithm of annotated OWL axioms into logic programming rules is introduced and illustrated by our running example. In the last two sections related work and conclusions are presented.

2 Example

Let us consider a hypothetical definition of desert. Desert could be defined as an area with little precipitation (the average amount of precipitation is less than 25 centimeters a year), partly covered by sand or gravel, having scanty vegetation or sometimes almost none, and capable of supporting only a limited and specially adapted animal population. The definition could be formally written in DL using the following form:

$$\text{Desert} \sqsubseteq \text{Area} \sqcap \exists \text{coveredby} . (\text{oneof}\{\text{Sand}, \text{Gravel}\}) \sqcap \forall \text{precipitation} . \text{Low} \sqcap \forall \text{vegetation} . \text{Scanty} \sqcap \forall \text{populatedby} . \text{AdaptedAnimal}$$

The typical criticism raised by representatives of Prototype Theory here could be such, that there are deserts that do not fulfill all the conditions of the above definition but which nevertheless are treated as good representatives of deserts. Consider for example cold deserts covered rather with ice than sand. Moreover, in many deserts there are regions where the vegetation is not sparse at all.

If we would like to include exceptions like those above to a knowledge base containing our definition of desert, clearly we will end up with inconsistencies. Hence we

¹ For instance description logics *SHIF(D)* and *SHOIN(D)* underlie OWL Lite and OWL DL respectively.

see that there is a need to adopt less restrictive forms of knowledge representation that include prototypes and permit nonmonotonic reasoning.

3 Nonmonotonic Reasoning

Long since it has been recognized in AI that classical logic is not sufficiently robust to adequately reason as humans do. The main feature of classical logic (as well as DLs) is monotonicity, in the sense that one cannot reject conclusions by adding new premises. However, people tend to retract previous conclusions, when new evidences appear. This is called nonmonotonic reasoning, which's subject matter is developing reasoning systems, that model the way in which commonsense is used by humans.

Formally, a consequence relation $\models$ is nonmonotonic if there is a formula β and a set of formulas α such that $\alpha \models \beta$ holds, then $\alpha, \gamma \not\models \beta$ may hold when new information γ appears (where γ is a formula). Brewka pointed out in [8] that nonmonotonic reasoning, in its broadest sense, is reasoning to conclusions on the basis of incomplete information. In this sense *default* conclusions are accepted until new information arises.

Nonmonotonicity may be seen as a property of knowledge and property of reasoning over knowledge. We accent here the former view and suggest that first of all the knowledge is defeasible or not defeasible. Knowledge is defeasible, if it may be invalidated in the light of certain additional information, called defeaters.

Reasoning in the Semantic Web is usually monotonic. The formal semantics of Semantic Web languages like OWL DL is given by corresponding Description Logics [3], which are mainly monotonic. This means that there is no way for retracting any conclusions in the light of new information. This may cause knowledge on the Web to get inconsistent as new knowledge is added. In addition, it has been recognized in AI that a knowledge system should allow a construction of elaboration tolerant knowledge bases, i.e. bases in which small modification of the informal body of knowledge corresponds to small modifications of the formal base representing this knowledge [13]. Nonmonotonicity helps to satisfy this requirement.

In the current paper we propose answer set semantics to be added to the Semantic Web language OWL and useage of answer set solvers (such as Smodels[22]) as reasoners. To the best of our knowledge, only the work of Bertino [5] considers a similar idea, namely encoding the Semantic Web representation of knowledge by a logic program under answer set semantics² and thus allowing default reasoning. The mentioned approach consists in providing a new default semantics to RDF type inheritance primitives, in particular to DAML+OIL properties `daml:Class` and `daml:subClassOf`. Then DAML+OIL sentences have translations into a logic program. Answer set semantics to basic DAML+OIL constructor relations is provided and a direct translation of RDF statements into logic facts is done. At the same time the answer sets of the program are logically equivalent to the RDF statements inferred from the given KB. The main advantage of such a method is that ASP semantics permits any conclusion thrown by default to be dropped if contrary explicit knowledge is found. Since it

² Some authors propose a translation from DLs to logic programming under answer set semantics. An overview is given in a section 7 concerning the related work.

makes the conclusions defeasible, nonmonotonicity has been successfully adopted to DAML+OIL. However, we show in the next section that the above solution sometimes leads to unintuitive results.

4 Against Absolute Nonmonotonicity

Nonmonotonic solutions and prototype theory in general do not remain without problems. In [6], [7] problems are reported, which arise by the adoption of defaults to frame systems. For example, the cancellation and overriding of a concept's features results in mishmash of subsumption trees and disturbs the classification algorithm. If all properties of the concept can be canceled then the concept may be subsumed by anything, which gives rise to absurd cases like the one reported by Brachman: "a rock is an elephant except it has no trunk, it isn't alive, it has no legs..."[7].

Although some, or even most of our knowledge permits exceptions, there is also knowledge that does not. In other words not all concepts have prototypical structure, but there are some that per se exclude any form of cancellation of their properties. Consider for instance mathematical concepts. They are not prototypically structured, but instead they have provided at least necessary conditions and therefore no exceptions are permissible. Each triangle must have exactly three sides and therefore no atypical, not three sided triangles are allowed. Reasoning about such not prototypically structured concepts should remain monotonic.

Thus, for at least two above reasons the solutions like the one of Bertino [5], interpreting all of our knowledge as defeasible, seem to be problematic. Not in all cases `daml:Class` and `daml:subClassOf` require nonmonotonic interpretation. In the next section we present a different approach that tries to overcome the above problems but still permits defaults.

5 Dual Theory Approach

In [23] Osherson and Smith proposed the Dual Theory, which is a hybrid approach to the prototype problem. They found that the application of the Zadeh's fuzzy set theory [32] for representing prototypes leads to several problems while representing conjunctive concepts as well as logically empty and logically universal concepts. These problems of fuzzy-set theory, and ubiquitous prototypical categories forced Osherson and Smith to compose prototypes with definitions of necessary (and sufficient) conditions.

According to the Dual Theory, concepts have a binary structure and are composed of two types of information [20], [21]. One of these has a prototypical character and another that does not. The prototypes are considered to be used in an identification procedure of a concept's membership and are responsible for rapid decisions about concept's membership. Properties that form the prototype of a concept we will call here *typical* since they are those, which typically are considered to identify a concept's instances. But since not all of a concept's instances share such typical features, we treat typical features as being defeasible, in the sense that they ought to be assumed in the absence of any contrary information [26].

The second type of information constituting concepts consists of, as called here, *core* properties. Core properties are not defeasible and they provide the truth conditions to hold for concept membership. They are used for combinations of concepts and are responsible for our most considered categorization judgments [23]. Typical properties then may not fully determine the concept instances, but this may be done by the concept's core. Secondly not all concepts must have a prototypical part (for example mathematical concepts). The identification procedure in those cases is fully held by the core of the concept.

Considering our desert example in the context of the Dual Theory we see that the properties of being covered with sand and having scanty vegetation are the typical properties of a desert. Usually when meeting a region satisfying these conditions we recognize it as being a desert. Those features form the prototype of a desert but they do not fully determine the concept's instances. As mentioned above, there are deserts that do not fulfill those conditions. In such cases we need to refer to more subtle conditions to decide about concept's membership. In this case it could be the feature of having a low average amount of precipitation. That feature is not defeasible and it constitutes the core of the concept of desert.

Typical features are defeasible while core features are not. If in a knowledge base inconsistency concerning typical features appears, it should be treated differently from inconsistencies concerning core features. In the first case typical feature should be canceled so that the inconsistency is avoided, but in the second case the cancellation should not take place. As an example, let us consider a stony desert. Although assuming that a stony desert is not covered by sand it is still considered to be a desert. A classification of a stony desert under the concept desert would result in inconsistency of the knowledge base. However, since being covered by sand is not a core property but only a typical one, it may be canceled and this enables us to place stony deserts in the hierarchy under the concept of desert without falling into inconsistency. The same does not hold for the property of having low precipitation which is considered to be a core property. A region not having a low precipitation, even if covered by sand and having scanty vegetation should not be classified as desert without leading to inconsistency.

The typical features are considered in the Dual Theory to be identification features of a concept. This means that from the features an individual has, one can infer under what concept the individual falls. To enable the inference in this direction, the implication from features to a concept is needed. Features should then not be necessary but sufficient conditions for concept membership, which does not hold in our example, however. In our case we can infer from a concept to its features, which means that features are considered as necessary conditions for concept membership.

5.1 Annotation of Statements

To make it explicit which part of the concept definition is core and which is only typical we use a simple metaontology consisting of two disjoint classes *typical* and *core*. Corresponding tags `[core]` and `[typical]` are used to annotate OWL axioms. These tags are assigned neither to OWL classes or properties directly nor to whole OWL axioms. The class or property is not core or typical in general but only in the context of some axiom. For example a property of being covered by sand is typi-

cal in case of a desert but may be core for a sandy beach. On the other hand not the entire axiom must be typical or core but only its parts, here called characteristics. The tags are assigned to the `rdfs:subClassOf` construct³, thus to RDF statements, where the resource `rdfs:subClassOf` is a predicate. We treat each such reified RDF statement as an instance of one of the two classes – core and typical. A reified statement is an instance of the class *core* iff the statement must always be satisfied for the given concept. A reified statement is typical iff it is satisfied for the concept, unless there is information to the contrary.

To illustrate this we can assign the tags to each statement of the exemplar definition of desert. We represent it in semi-description logics notation in the following way:

```
Desert  $\sqsubseteq$  [core]Area  $\sqcap$ 
    [typical] $\exists$ coveredby.(oneof{Sand, Gravel})  $\sqcap$ 
    [core] $\forall$ precipitation.Low  $\sqcap$ 
    [typical] $\forall$ vegetation.Scanty  $\sqcap$ 
    [typical] $\forall$ populatedby.AdaptedAnimal
```

From the above we can read that the characteristics saying that desert is an area having low precipitation are core, whereas all others are only typical. The above definition now says that it may happen that there is a desert not covered by sand or gravel but it may not happen that a desert is not a subclass of area.

To assign metatags to OWL ontologies we use reified RDF statements and treat them as instances of one of the given metaclasses. For example in OWL abstract syntax the first tagged statement of the above definition looks as follows:

```
Individual(desertStatement1
    type(rdf:statement)
    type(metaontology:core)
    value(rdf:object Area)
    value(rdf:predicate rdfs:subClassOf)
    value(rdf:subject Desert))
```

Metaontology refers to the metaontology, where the classes: *core* and *typical* are defined.

6 Answer Set Programming for the Semantic Web

In this section the translation of annotated OWL axioms into a logic program under answer set semantics is provided and the answer set solver Smodels [22] is used to find a model for the given KB. For simplicity of reading we use a semi-description logic notation instead of the OWL abstract syntax.

³ In the current paper we consider only partial definitions, but the approach can be extended to complete definitions as well. In that case also `owl:equivalentClass` should be tagged.

6.1 Answer Set Programming

Answer set programming (ASP) is a new logic programming paradigm for knowledge representation and problem solving in artificial intelligence [19]. Representation of a problem is purely declarative, suitable for many aspects of commonsense reasoning (diagnosis, configuration, planning etc.). Instead of a query technique (Prolog), it bases upon a notion of possible solution, called *answer set*.

Consider a propositional language L , with atomic symbols called atoms. A literal is an atom or a negated atom (by classical negation $\neg$). The symbol *not* is called epistemic negation and the expression *not a* is true, if there is no reason to believe that a is the case. Epistemic negation makes ASP a nonmonotonic system. Default rules, whose conclusions are defeasible in the light of certain knowledge, are represented using epistemic negation. Knowledge which is not defeasible is represented by means of the rules without negation *not*. The symbol $\vee$ is called epistemic disjunction and it is interpreted as follows: at least one literal is believed to be true. Formally, a rule r is an expression of the form:

$$c_1 \vee \dots \vee c_k \leftarrow a_1, \dots, a_m, \text{not } b_{m+1}, \dots, \text{not } b_n \quad (1)$$

where $k \geq 0$, $n \geq m \geq 0$, c_i , a_i , b_k are ground literals r , $Body^-(r) = \{b_{m+1}, \dots, b_n\}$, $Body^+(r) = \{a_1, \dots, a_m\}$ and the disjunction $\{c_1 \vee \dots \vee c_k\}$ is *Head*(r) of the rule r . A rule with an empty *Head* (i.e. a rule of the form: $\leftarrow Body$) is usually referred to as integrity constraint. A logic program is a finite set of the rules.

Intuitively the above rule r means that if $Body^+(r)$ of that rule is believed to be true and it is not the case that $Body^-(r)$ is believed to be true, then at least one literal of *Head*(r) must be believed to be true.

The semantics for ASP is defined by means of minimal set of literals satisfying all rules of the program P . Let us now assume now, that Lit_P is the set of all literals being present in the extended logic program P and I is an interpretation of P , $I \subseteq Lit_P$. We say that a set of literals I satisfies a rule of the form (1), if $\{a_1, \dots, a_m\} \subseteq I$ and $\{b_{m+1}, \dots, b_n\} \cap I = \emptyset$ imply that $\{c_1, \dots, c_k\} \cap I \neq \emptyset$.

The Gelfond-Lifschitz (GL) transformation of P with respect to I is a positive logic program P' which is obtained in two steps [19]:

- deletion of all rules r of P , for which $Body^-(r) \cap I \neq \emptyset$
- deletion of the negative bodies ($Body^-(r)$) from the remaining rules of P

Then, I is an *answer set* of the logic program P , if I is a minimal model (no proper subset of I is a model of P') of the positive (without *not*) logic program P' ; i.e. I is a minimal set of literals satisfying every rule in P' or if I contains a pair of complementary literals l and $\neg l$, then $I = Lit_P$.

Example. Consider the program $P_1 = \{b \leftarrow \text{not } a; a \leftarrow \text{not } b; f \leftarrow a\}$, $Lit = \{a, b, f\}$, and let $I = \{a, f\}$. The GL reduction of the program P_1 w. r. t. I is the program $P_1' = \{a \leftarrow; f \leftarrow a\}$. According to the definition, I is an answer set for the program P , since I is a minimal set of literals satisfying all rules in P_1' . The second answer set of the program P is $\{b\}$ and these are the only answer sets for this program.

In general, programs under answer set semantics describe a family of intended models. Therefore they encode possible solutions to a problem, being a constraint sat-

isfaction problem, and described by the program, where each rule is interpreted as a constraint.

Although answer set programs are basically propositional, it is possible to use a rule schemata containing variables. These schemata are representations of their ground instances, and answer set solvers [11], [22] use intelligent ground instantiation techniques before the actual answer set computation takes place⁴. In this case logic programs with variables may be used to represent more complex problems. We will use such a technique as a convenient representation of our running example.

6.2 Translation OWL to ASP

We translate the ontology represented in annotated OWL to a logic program under answer set semantics and use the answer set solver as reasoning tool. To handle defeasibility of typical knowledge, we use ASP rules with epistemic negation in the *body* for encoding default information. In the light of certain information being defeaters, the conclusions assumed to be defeasible may be retracted. Knowledge, which is not defeasible, i.e. the *core* properties in our case, is represented in a form of positive rules not allowing exceptions.

Any inconsistency w.r.t. the nondefeasible knowledge leads to the inconsistency of the whole KB indicating the fact that such conclusions may not be retracted. By that means nonmonotonic reasoning is enabled, but on the other hand the deletion of all conclusions, in opposite to [5], is avoided.

In general, we base on the work of Baral [1] who successfully translates DL *ALCQI* into declarative logic programming under answer set semantics. Our algorithm provides a translation of annotated RDF statements into suitable logical rules. Since, in case of the *core* properties of concepts the standard DL semantics is appropriate, the translation given in [1], except for step 6 can be used. Therefore the inference over the *core* knowledge is strictly monotonic. We are mainly interested in handling the default part of knowledge; therefore we show how statements annotated by the tag [typical] can be represented in ASP.

Below the general schema of the translation of default knowledge into corresponding logic program rules is given. In detail, we encode one type of formulae, namely default subsumption between an atomic concept and a formula. Instead of the standard OWL semantics, default ASP semantics is provided to such subsumption. The formula encoded in the semi DL notation: $C_1 \sqsubseteq [\text{typical}] C_2$ has the following intuitive meaning: concept C_1 by default subsumes C_2 where C_2 may be more a complex formulae. In order to determine the defeaters (knowledge capable to cancel the default conclusions), for the typical subsumption, we use for each default rule a predicate ab_k where k is a unique index. The precise algorithm is provided in the current section.

For all steps, C_1 denotes an atomic concept, while C_2 corresponds to an arbitrary complex formula. The Herbrand universe (HU), which is a set of ground terms constructed from the constants and functions in the program, is divided into disjoint sets of sub-domains. This way the search space for a solver is restricted significantly,

⁴ The answer set semantics of ground programs can be extended straightforwardly to programs with variables by employing the notion of Herbrand models.

since smaller the Herbrand base (HB, being the set of atomic ground formulae built from the HU and the predicate symbols of the program) is considered.

Since the translation of atomic concepts, roles and individuals is provided in [1] (Steps 1-5), we skip it in the algorithm presented below and demonstrate it only in the context of our running example.

Steps 1-5 remain unchanged as given in [1] with the exception that we divide HU (hence as well HB) into domains, containing disjoint elements.

Step 6.

C_2 is an atomic concept

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_1(X).$

$ab_1(X) \leftarrow c_1(X), \neg c_2(X).$

Step 7.

a) C_2 is of the form: $\neg C_3$:

$c_2(X) \leftarrow \text{domain}(X), \text{not } c_3(X).$

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_2(X).$

$ab_2(X) \leftarrow c_1(X), \neg c_2(X).$

b) C_2 is of the form $C_3 \sqcap C_4$:

$C_1 \sqsubseteq [\text{typical}]C_3$

$C_1 \sqsubseteq [\text{typical}]C_4$

and step 6

c) C_2 is of the form: $C_3 \sqcup C_4$:

$c_2(X) \leftarrow \text{domain}(X), c_3(X).$

$c_2(X) \leftarrow \text{domain}(X), c_4(X).$

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_3(X).$

$ab_3(X) \leftarrow c_1(X), \neg c_3(X), \neg c_4(X).$

d) C_2 is of the form: $\forall R.C_3$:

$\text{not\_}c_2(X) \leftarrow \text{domain}(X), \text{domain}(Y), r(X,Y), \text{not } c_3(Y).$

$c_2(X) \leftarrow \text{domain}(X), \text{domain}(Y), \text{not not\_}c_2(Y), r(X,Y).$

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_4(X).$

$ab_4(X) \leftarrow c_1(X), r(X,Y), \neg c_3(Y).$

e) C_2 is of the form: $\exists R.C_3$:

$c_2(X) \leftarrow r(X,Y), c_3(Y).$

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_4(X).$

$ab_4(X) \leftarrow c_1(X), r(X,Y), \neg c_3(Y).$

f) C_2 is of the form: $\exists R^{\geq n}.C_3$:

$c_2(X) \leftarrow r(X,Y_1), \dots, r(X,Y_n), c_3(Y_1), \dots, c_3(Y_n), Y_1 \neq Y_2 \neq \dots \neq Y_n.$

$C_1 \sqsubseteq [\text{typical}]C_2$:

$\leftarrow c_1(X), \text{not } c_2(X), \text{not } ab_5(X).$

$ab_5(X) \leftarrow \text{not } c_2(X), c_1(X).$

g) C_2 is of the form: $\exists R^{\leq n}.C_3$:
 $\text{not\_}c_2(X) \leftarrow r(X, Y_1), \dots, r(X, Y_{n+1}), c_3(Y_1), \dots, c_3(Y_{n+1}), Y_1 \neq Y_2 \neq \dots \neq Y_{n+1}$.
 $c_2(X) \leftarrow \text{not not\_}c_2(X), \text{domain}(X)$.
 $C_1 \sqsubseteq [\text{typical}]C_2$:
 $\leftarrow c_1(X), \text{not } c_2(X), \text{not ab}_6(X)$.
 $\text{ab}_6(X) \leftarrow \text{not\_}c_2(X), c_1(X)$.
h) C_2 is of the form: $\text{oneOf} \{a_1, \dots, a_n\}$
 $c_2(X) \leftarrow \text{domain}(X), X=a_1$.
 $\dots$
 $c_2(X) \leftarrow \text{domain}(X), X=a_n$.
 $C_1 \sqsubseteq [\text{typical}]C_2$:
 $\leftarrow c_1(X), \text{not } c_2(X), \text{not ab}_7(X)$.
 $\text{ab}_7(X) \leftarrow \text{not } c_2(X), c_1(X)$.

In order to explain the above algorithm, we consider our running example and show how it can be translated into a logic program under answer set semantics. Then the answer set solver Smodels [22] is used to compute answer sets of the logic program such that models of the program correspond to the conclusions of the given KB. Note that we consider the Herbrand universe only, thus models are restricted to Herbrand models only (subsets of the Herbrand base). Since our approach is non-monotonic, multiple models of a program may exist. We distinguish between brave and cautious reasoning as follows:

- **Brave reasoning:** given a logic program P and a ground literal l , decide whether l is true in some answer set of P (denoted $P \models_b l$)
- **Cautious reasoning:** given a logic program P and a ground literal l , decide whether l is true in all answer sets of P (denoted $P \models_c l$)

Note that to decide whether l is true in answer set S means to check whether l belongs to answer set S . Thus, literals belonging to all answer sets may be called the cautious conclusions of the given KB (under HB), while literals belonging to some answer sets are called brave default conclusions of the given KB (under HB). Therefore, the cautious conclusions of a logic program P are consequences of the KB represented by P under the Herbrand base, while the brave conclusions of P are *possible* consequences of the KB represented by P , obtained under incomplete knowledge.

It is clear that two features of desert are *core* properties: *area* and *precipitation*. *low*. Since they are not defeasible knowledge, we represent them in a form of not defeasible logic program rules, as showed in [1]. The remaining properties can be seen as the typical properties of desert, thus they are encoded as default rules by means of the method given above. Each atomic concept, role and fact assertion as well as elements of HU are encoded as logical rules and domain predicates respectively. Let us assume that $HU = \{\text{sturt, stone, gravel, sand, few\_cactus, twenty\_cm, camel}\}$. We use here Smodel's syntax to encode the above example.

Step 1. Elements of HU are encoded as facts belonging to the domains:

```
d_cover(gravel;sand;stone). d_desert(sturt). d_animal(camel).
d_vegetation(few_cactus). d_prec_level(twenty_cm).
```

Step 2. Atomic concepts are represented by the following rules:

```

area(X) :- d_desert(X), not not_area(X).
not_area(X) :- d_desert(X), not area(X).
low(X) :- d_prec_level(X), not high(X).
high(X) :- d_prec_level(X), not low(X).
scanty(X) :- d_vegetation(X), not not_scanty(X).
not_scanty(X) :- d_vegetation(X), not scanty(X).
adopted_animal(X) :- d_animal(X), not not_adopted_animal(X).
not_adopted_animal(X) :- d_animal(X), not adopted_animal(X).

```

Step 3. Atomic roles are represented by the following rules:

```

coveredby(X,Y) :- d_desert(X), d_cover(Y), not not_coveredby(X,Y).
not_coveredby(X,Y) :- d_desert(X), d_cover(Y), not coveredby(X,Y).
precipitation(X,Y) :- d_desert(X), d_prec_level(Y), not
not_precipitation(X,Y).
not_precipitation(X,Y) :- d_desert(X), d_prec_level(Y), not precipi-
tation(X,Y).
vegetation(X,Y) :- d_desert(X), d_vegetation(Y), not
not_vegetation(X,Y).
not_vegetation(X,Y) :- d_desert(X), d_vegetation(Y), not
vegetation(X,Y).
populatedby(X,Y) :- d_desert(X), d_animal(Y), not
not_populatedby(X,Y).
not_populatedby(X,Y) :- d_desert(X), d_animal(Y), not
populatedby(X,Y).

```

Step 4. Fact assertions are encoded as logic programming facts:

```
desert(sturt). low(twenty_cm).
```

Step 5. Each role assertion is translated to the facts:

```
coveredby(sturt, stone).
```

Step 7. A subsumption, where a super concept is a complex concept in form of a conjunction, is split first into the conjunction of subsumptions (step 7 (b) of our procedure):

- (I) desert $\sqsubseteq$ [core] area
- (II) desert $\sqsubseteq$ [typical] $\exists$ coveredby.(oneof{Sand, Gravel})
- (III) desert $\sqsubseteq$ [core] $\forall$ precipitation.Low
- (IV) desert $\sqsubseteq$ [typical] $\forall$ vegetation.Scanty
- (V) desert $\sqsubseteq$ [typical] $\forall$ populatedby.AdoptedAnimal

Since a *core* subsumption corresponds to the semantics of standard DL subsumption, we translate it according to the method given in [1]. Then:

(I) triggers translation of the core subsumption:

```
:- desert(X), not area(X).
```

(II) appeals first step 7 (e):

```
ok_coveredby(X) :- coveredby(X,Y), oneOf(Y), d_desert(X).
```

which triggers step 7 (h):

```

oneOf(X) :- d_cover(X), X=sand.
oneOf(X) :- d_cover(X), X=gravel.

```

and step 7 (e) for default subsumption:

```

:- desert(X), not ok_coveredby(X), not ab_2(X).
ab_2(X) :- desert(X), coveredby(X,Y), not oneOf(Y), d_cover(Y).

```

(III) is translated according to step 7 (d):

```

not_precipitation_ok(X) :- precipitation(X, Y), not low(Y),
d_desert(X), d_prec_level(Y).
precipitation_ok(X) :- not not_precipitation_ok(X), precipita-
tion(X,Y), d_desert(X), d_prec_level(Y).
:- desert(X), not precipitation_ok(X).

```

(IV) is represented due to step 7 (d) as:

```
not_vegetation_ok (X) :- vegetation(X, Y), not scanty (Y),
d_desert(X), d_vegetation(Y).
vegetation_ok (X) :- not not_vegetation_ok (X), vegetation(X,Y),
d_desert(X), d_vegetation(Y).
```

and step 7 (d) for default subsumption:

```
:- desert(X), not vegetation_ok(X), not ab_3 (X).
ab_3(X) :- desert(X), vegetation(X, Y), -scanty(Y).
```

(V) is encoded similar to the previous formulae by means of step 7 (d) :

```
not_populatedby_ok (X) :- populatedby(X, Y), not adopted_animal(Y),
d_desert(X), d_animal(Y).
populatedby_ok (X) :- not not_populatedby_ok (X), populatedby(X,Y),
d_desert(X), d_animal(Y).
```

and step 7 (d) for default subsumption:

```
:- desert(X), not populatedby_ok(X), not ab_5 (X).
ab_5(X) :- desert(X), populatedby(X, Y), -adopted_animal(Y).
```

Smodels computes 4 answer sets, where each contains the following:

```
desert(sturt), area(sturt), ab_2(sturt), populatedby(sturt, camel),
scanty(few_cactus), vegetation(sturt, few_cactus), precipita-
tion(sturt, twenty_cm).
```

We can see that Sturt is an abnormal desert w.r.t. the covering (ab_2) since it is not covered by sand or gravel but only by stones. The condition of being covered by sand or by gravel is not fulfilled in the case of Sturt desert, nevertheless Sturt is considered being a desert, as intended. It would not be the case if the core property of having low precipitation is not satisfied. Then we would obtain no model which corresponds to the intuition that there is no desert without low precipitation.

7 Related Work

Related work can be organized into two groups. The first group contains approaches that explicitly distinguish between defeasible and nondefeasible parts of a concept's structure. In MultiNet [15], which is a knowledge representation paradigm along the line of Semantic Networks, three types of concept descriptions are distinguished: categorical, prototypical and situational. Categorical and prototypical knowledge corresponds to our core and typical characteristics, respectively.

The idea of partial nonmonotonicity for the Semantic Web is introduced in [4]. The system called DR-DEVICE is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules. The implementation is declarative because it interprets the *not* operator using well-founded semantics. Compared to our one, this approach uses well-founded semantics instead of answer set semantics. In fact, the second is more robust and general in the sense that every well-founded model is an answer set but not conversely.

On the other hand many authors investigate the problem of integrating DLs with other defeasible logics. Baader in [2] considers the problem of integrating Reiter's default logic into a terminological representation system. In [24] a general framework for Preferential Default Description Logics (PDDL) is developed. Recently Heymans in [16] extended the description logic $\mathcal{SHOQ}(\mathbf{D})$ by a preference order on the axioms and thus effectively introduced nonmonotonicity into $\mathcal{SHOQ}(\mathbf{D})$.

As we are interested in adding defeasibility to Semantic Web languages directly by means of answer set semantics, only the work of Bertino [5] and colleagues developed a similar idea. However, instead of giving a non-standard interpretation to the properties of DAML+OIL, we divide knowledge into defeasible and not defeasible parts, thus avoiding all conclusions to be canceled.

A second group of related work forms proposals for translating DLs to various logic programming languages. In [14] Grosz investigates many possible interactions among DLs and Datalog logic programs (def-LPs). Similar approaches are presented in [10] and [18] where plain Datalog is combined with certain DL paradigms and hybrid languages are proposed. Finally, Rosati in [27] combines disjunctive Datalog with *ALC* based on a generalized answer set semantics.

On the other hand some authors discuss approaches combining DLs with a more expressive logic programming paradigm, namely Answer Set Programming. In [17] the DL *SHIF* is simulated by free disjunctive logic programs (DLP). Alsac and Baral in [1] show a translation, used as the foundation of presented framework, of DL *ALCQI* into declarative logic programming under answer set semantics. Swift [29] reduces inference in the description logic *ALCQI* to query answering from answer sets of logic programs. In the paper [12] the integration of rules and ontologies in the Semantic Web in a form of combining logic programming under the answer set semantics with description logics is presented. The paper is focused on *SHIF(D)* and *SHOIN(D)*.

Mainly, the intention of the papers mentioned above is either to combine the semantic and computational strengths of the two systems or to use powerful logic programming technology for inference in description logics, as noticed in [12]. The idea behind our work is similar to the former view, since the answer set solver is used as reasoner. On the other hand, we are close to the spirit of [2], [16] and [4], where conclusions are divided explicitly into defeasible and not defeasible ones.

8 Conclusions and Future Work

In the current paper we have presented a framework that permits to delimit the defeasible part of a concept here called *typical* from the nondefeasible one - *core*. It is a balanced way between purely monotonic approaches to representing ontologies in the Semantic Web on the one hand and purely nonmonotonic on the other. Both types of knowledge, defeasible and not defeasible, can be coherently represented by means of the developed framework.

We developed a system of metatags for annotating characteristics of concepts. The tags are defined in form of a metaontology. The tags [core] and [typical] assigned to each of a concept's characteristic provide the information whether the characteristic has a defeasible or nondefeasible nature.

A translation of annotated OWL axioms into a logic program under answer set semantics is given. ASP provides an intuitive semantics to the Semantic Web languages and extends them by nonmonotonicity. On the other hand a powerful logic programming technology is used for inference purposes. We use the answer set solver *Smodels* as reasoner for annotated ontologies, thus handling properly the distinction between monotonic and nonmonotonic reasoning.

The framework is grounded on the cognitive theory of a concept's dual structure. Adoption of the Dual Theory provides a cognitive foundation for concept specifications in ontologies. The concepts in ontologies may then be cognitively more adequate, which means that when formalized they still preserve the structure of their originals. It seems, that cognitive adequacy of concepts reduces the deformation of knowledge which often takes place in the process of knowledge modeling and formalization [31].

Moreover, the approach permits handling inconsistency in the knowledge base, which, as presented, should be treated differently in the context of defeasible or non-defeasible knowledge. The inconsistency of the core, i.e. of nondefeasible knowledge is not allowed but when concerning only a defeasible part of a concept's specification, it may be reduced to abnormality. As far as inconsistency among default conclusions is concerned, it is handled in answer set semantics by means of multiple models, representing possible solutions. In case of incomplete information, we admit default conclusions as long as no defeating information arises.

In future work the presented annotations can be extended so that not only the distinction of defeasible and not defeasible knowledge is possible, but also levels of defeasibility may be introduced. This may be done by adopting one among many approaches, extending answer set programming by priorities. One of the major proposals is Logic Programming with Ordered Disjunction, proposed by Brewka in [9]. In this framework, priorities among literals may be expressed such that degrees of defeasibility among abnormalities may be formulated. Moreover, the translation should be extended so the OWL datatypes are handled.

Acknowledgements. We are indebted to Hesham Khalil, Sören Auer, Frank Loebe and Sebastian Dietzold for fruitful discussions and to the anonymous reviewers for feedback on earlier versions of this paper.

References

1. Alsac, G., Baral, Ch.: Reasoning in description logics using declarative logic programming. Abstract, ASU Technical Report (2001-2002)
2. Baader, F., Hollunder, B.: Embedding Defaults into Terminological Representation Systems. *Automated Reasoning* 14 (1995) 149–180
3. Baader F. et.al.: *The Description Logic Handbook*. Cambridge University Press, 2003
4. Bassiliades, N. et. al.: A Defeasible Logic Reasoner for the Semantic Web. *RuleML* (2004) 49-64
5. Bertino E. et. al.: Local Closed-World Assumptions for reasoning about Semantic Web-data. *Proc. of AGP'03 APPIA-GULP-PRODE* 2003
6. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2) (1985) 171-216
7. Brachman, R.J.: 'I Lied about the Trees' or, Defaults and Definifons in Knowledge Representation. *AI Magazine* 6(3) (1985) 80-93
8. Brewka G.: *Nonmonotonic Reasoning: Logical Foundations of Commonsense* Cambridge: Cambridge University Press, 1991
9. Brewka, G.: Logic Programming with Ordered Disjunction. *Proc. of AAAI'02, Canada* (2002) 100-105

10. Dionini, F.M. et. al.: AL-log: integrating datalog and description logics. *Journal of Intelligent and Cooperative Information Systems* 10 (1998) 227-252
11. Eiter, T. et.al.: The DLV system for knowledge representation and reasoning: Infsys Research Report, Austria (2002)
12. Eiter, T., et.al.: Combining Answer Set Programming with Description Logics for the Semantic Web. *Proc. of KR'04, Canada* (2004) 141-151
13. Gelfond M. and N. Leone: Logic Programming and Knowledge Representation - A-Prolog perspective , *AI* 138 (2002) 3-38
14. Grof, B.N. et. al.: Description Logic Programs: Combining Logic Programs with Description Logic. *Proc. of WWW'03 Hungary* (2003) 48-57
15. Helbig, H., Gnörlich, C.: Multilayered Extended Semantic Networks as a Language for Meaning Representation in NLP Systems. In: Gelbukh, A. F. (eds.): *Computational Linguistics and Intelligent Text Processing Proceedings of the Third International Conference, CICLING'02, Mexico* (2002) 17-23
16. Heymans, S., Vermeir, D.: A Defeasible Ontology Language. *Proc. of ODBASE'02, USA* (2002) 1033-1046
17. Heymans, S., Vermir, D.: Integrating ontology languages and answer set programming. *Proc. of DEXA'03, Czech Republic* (2003) 584
18. Levy, A., Rousset, M.: CARIN: A representation language combining Horn rules and description logics. *Proc. of ECAI'96, Hungary* (1996) 323-327
19. Lifschitz, V.: Answer set programming and plan generation. *AI* 138 (2002) 39-54
20. Margolis E., Laurence S.: Concepts and Cognitive Science. In E. Margolis E., Laurence S. (eds.): *Concepts: Core Readings*. Cambridge, MA.: Bradford Books/MIT Press, 1999.
21. Margolis E., Laurence S.: Concepts. In Warfield T., Stich S. (eds): *The Blackwell Guide to the Philosophy of Mind*. Blackwell. 2003
22. Niemelä, I., Simons, P.: Smodels-an implementation of the stable model and well-founded semantics for normal logic programs. *Proc. Of LPNMR'97, Germany* (1997) 420-429
23. Osherson D.N., Smith E.E.: On the adequacy of prototype theory as a theory of concepts. *Cognition* 9(1) (1981) 35-58
24. Quantz, J.J., Ryan, M.: Preferential Default Description Logics. KIT-Report 110. Technische Universität Berlin (1993)
25. Patel-Schneider, P.F. et. al.: Web ontology language (owl) abstract syntax and semantics. *W3C Recommendation* (2004)
26. Reiter, R.: On reasoning by default. *Proc. of Theoretical Issues in Natural Language Processing USA* (1978) 210 - 218
27. Rosati, R.: Towards expressive KR systems integrating datalog and description logics: Preliminary report. *Proc. of DL'99, Sweden* (1999) 160-164
28. Rosch, E., Mervis, C.: Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology* 7 (1975) 573-603
29. Swift, T.: Deduction in ontologies via ASP. *Proc. of LPNMR'04, USA* (2004) 275-288
30. Wittgenstein, L.: *Philosophical Investigations*. Blackwell. Oxford. 1953
31. Zhang, J.: Representation of Health Concepts: Cognitive Perspective. *Journal of Biomedical Informatics* 35 (2002) 17-24
32. Zadeh, L.: Fuzzy Sets. *Inform.control* 8 (1965) 338-353

Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs

Stijn Heymans, Davy Van Nieuwenborgh*, and Dirk Vermeir**

Dept. of Computer Science,
Vrije Universiteit Brussel, VUB,
Pleinlaan 2, B1050 Brussels, Belgium
{sheymans, dvnieuwe, dvermeir}@vub.ac.be

Abstract. We present *extended conceptual logic programs (ECLPs)*, for which reasoning is decidable and, moreover, can be reduced to finite answer set programming. ECLPs are useful to reason with both ontological and rule-based knowledge, which is illustrated by simulating reasoning in an expressive description logic (DL) equipped with DL-safe rules. Furthermore, ECLPs are more expressive in the sense that they enable nonmonotonic reasoning, a desirable feature in locally closed subareas of the Semantic Web.

1 Introduction

Reasoning with both ontological knowledge, in the form of a description logic (DL)[3] knowledge base, and rule-based knowledge has recently gained in interest in the Semantic Web community. The purpose of adding rules to ontological knowledge is to have additional expressiveness. E.g., [23] extends a DL knowledge base with *DL-safe rules*, i.e. Horn clauses where variables must appear in non-DL-atoms in the body of rules. DL-safe rules can, e.g., express triangular knowledge not expressible with DLs alone: $uncle(a, c) \leftarrow brother(a, b), parent(b, c)$.

DL-safe rules do not include the *negation as failure (naf)* operator, and as a consequence, do not cope well with incomplete or dynamically changing knowledge: like reasoning with DL, reasoning with DL knowledge bases and DL-safe rules is monotonic. However, nonmonotonic reasoning may be useful in applications that involve well-defined closed subareas of the Semantic Web, as illustrated in the following example. Assume a business is setting up its website for processing customer feedback. It decides to commit to an ontology $\mathcal{O}$ which defines that if there are no complaints for a product, it is a good product.

$$good\_product(X) \leftarrow not\ complaint(X)$$

The business has its own particular business rules, e.g. $i : invest(tps, 10K) \leftarrow not\ good\_product(tps)$ saying that if its particular top selling product tps cannot be

* Supported by the FWO.

** This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

shown to be a good product, then the business has to invest 10K in *tps*. Finally, the business maintains a repository of dynamically changing knowledge, originating from user feedback collected on the site, e.g. at a certain time the repository contains $R_1 = \{complaint(tps) \leftarrow\}$ with a complaint for *tps*.

If the business wants to know whether to invest more in *tps* it needs to check $\mathcal{O} \cup \{i\} \cup R_1 \models invest(tps, 10K)$, i.e. whether the ontology, combined with its own business rules, and the information repository, demand for an investment or not.

One can use *extended conceptual logic programming (ECLP)* to express the above knowledge. Intuitively, any model of $\mathcal{O} \cup \{i\} \cup R_1$, must verify *complaint(tps)*, and thus *good_product(X) ← not complaint(X)* will not trigger and *good_product(tps)* will be false, which in turn, with rule *i*, allows to conclude that the business should indeed invest.

Evaluating the same query with an updated repository $R_2 = \{complaint(tps) \leftarrow, good\_product(tps) \leftarrow\}$ containing a survey result saying that *tps* is a good product, no matter what complaints of individual users there may be, leads to $\mathcal{O} \cup \{i\} \cup R_2 \not\models invest(tps, 10K)$, such that no further investments are necessary. Adding knowledge thus invalidates previous conclusions making reasoning nonmonotonic; similar scenarios can easily be imagined in any environment with dynamically changing knowledge.

In this paper, we formally introduce ECLP programs which consist of two (possibly empty) parts: a *conceptual logic program (CLP)* capable of expressing conceptual knowledge, as in e.g. DL knowledge bases, and an arbitrary *finite grounded program* which allows to relate constants/individuals in arbitrary ways, enabling e.g. the expression of triangular knowledge. More specifically, ECLPs can simulate reasoning in the DL $\mathcal{ALCH}OQ(\sqcup, \sqcap)$ equipped with DL-safe rules. Besides the advantage of uniform syntax and semantics that ECLPs have over DLs equipped with DL-safe rules¹, ECLPs are capable, as indicated above, of nonmonotonic reasoning as well.

Furthermore, we will show that reasoning with ECLPs can be reduced to finite answer set programming by virtue of the forest-model property and the bounded finite model property. The reduction to finite ASP makes reasoning with ECLPs amenable for existing answer set solvers such as DLV[21] or SMODELS[25].

The remainder of the paper is organized as follows. After recalling the open answer set semantics in Section 2, ECLPs are formally introduced in Section 3. Section 4 describes the simulation of an expressive class of DLs equipped with DL-safe rules. Section 5 highlights some related work while Section 6 contains conclusions and directions for further research. Due to space restrictions all proofs have been omitted; they can be found at <http://tinf2.vub.ac.be/~sheymans/tech/oasp-sw.ps.gz>.

2 Answer Set Programming with Open Domains

Answer set programming (ASP)[5] is a logic programming paradigm where knowledge is represented by programs and answer sets provide for the intended seman-

¹ SWRL[20] also combines ontologies and rules in one uniform syntax and semantics; reasoning with it is, however, undecidable.

tics of that knowledge. However, in certain cases ASP fails to capture the intention of the program. Take the program consisting of the rules $bad(X) \leftarrow not\ good(X)$ and $good(heather) \leftarrow$, where one is bad if not good and Heather is a good person. In ASP a program is grounded with the constants in the program, resulting in $bad(heather) \leftarrow not\ good(heather)$ and $good(heather) \leftarrow$, after which the unique answer set $\{good(heather)\}$ can be calculated. One would thus wrongfully conclude that there can never be bad individuals. In [17], this was solved by considering *open domains*, i.e. the program may be grounded with any superset of the present constants: grounding with a universe $\{x, heather\}$ yields $bad(heather) \leftarrow not\ good(heather)$; $bad(x) \leftarrow not\ good(x)$ and $good(heather) \leftarrow$, which has an answer set $\{bad(x), good(heather)\}$, correctly capturing the intended meaning of the program.

We briefly recall the open answer set semantics from [17]. We call individual names *constants* and write them as lowercase letters, *variables* will be denoted with uppercase letters. Variables and constants are *terms*. *Atoms* are of the form $a(t)$ or $f(t_1, t_2)$, with a a unary predicate, f a binary predicate, and t, t_1 and t_2 terms. A *literal* is an atom or an atom preceded by $\neg$. An *extended literal* is a literal l or a *naf-literal* $not\ l$ where l is a literal. We will often denote a set of unary extended literals $\{a_1(s), \dots, a_n(s)\}$, ranging over a common term s , as $\alpha(s)$ with $\alpha = \{a_1, \dots, a_n\}$. A set of binary extended literals can be similarly denoted as $\alpha(s, t)$. The positive part of a set of extended literals β is $\beta^+ = \{l \mid l \in \beta, l \text{ literal}\}$, the negative part is $\beta^- = \{l \mid not\ l \in \beta\}$. Furthermore, we assume the existence of a binary predicate $\neq$, with the usual interpretation.

A *disjunctive logic program* (DLP) is a finite set of rules $r : \alpha \leftarrow \beta$ where α and β are finite sets of extended literals and $|\alpha^+| \leq 1$. If $\alpha = \emptyset$, the rule is called a *constraint*. The set α is the *head* of the rule r , denoted $head(r)$, while β is called the *body*, denoted $body(r)$. As usual, atoms, (extended) literals, rules, and programs that do not contain variables are *ground*. For a set X of literals, $\neg X = \{\neg l \mid l \in X\}$, where, by definition, $\neg\neg a \equiv a$. A set of ground literals X is *consistent* if $X \cap \neg X = \emptyset$.

For a DLP P , let $\mathcal{H}_P$ be the constants in P and $vars(P)$ its variables. A (possibly infinite) non-empty set of constants $\mathcal{H}$ such that $\mathcal{H}_P \subseteq \mathcal{H}$, is called a *universe* for P . We call $P_{\mathcal{H}}$ the *grounded program* obtained from P by substituting every variable in P by every possible constant in $\mathcal{H}$. Let $\mathcal{L}_P$ be the set of literals that can be formed from a grounded program P , $preds(P)$ are the predicates² in P , and $upreds(P)$ and $bpreds(P)$ the unary and binary predicates respectively.

An *interpretation* I of a grounded P is any consistent subset of $\mathcal{L}_P$. For a ground literal l , we write $I \models l$, if $l \in I$, which extends to $I \models not\ l$ if $I \not\models l$, and, for a set of ground extended literals X , $I \models X$ if $I \models x$ for every $x \in X$. A ground rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. I , denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$, i.e. r is *applied* whenever it is *applicable*. A ground constraint $\leftarrow \beta$ is satisfied w.r.t. I if $I \not\models \beta$. For a *simple* grounded program P (i.e. a program without *not*), I is a *model* of P if I satisfies every rule in P ; it is an *answer set* of P if it is a subset minimal model of P . For grounded programs P containing *not*, the *GL-reduct*[13] w.r.t. I is defined as P^I , where P^I contains $\alpha^+ \leftarrow \beta^+$ for $\alpha \leftarrow \beta$ in P , $\beta^- \cap I = \emptyset$ and $\alpha^- \subseteq I$. I is an *answer set* of a grounded P if I is an answer set of P^I . An *open interpretation* of P

² When speaking of predicates, also the (classically) negated predicates are assumed.

is a pair $(\mathcal{H}, I)$ where $\mathcal{H}$ is a universe for P and I is an interpretation of $P_{\mathcal{H}}$. An *open answer set* of P is then an open interpretation $(\mathcal{H}, M)$ with M an answer set of $P_{\mathcal{H}}$. In the following, we will usually omit the “open” qualifier. We express the motivation of a literal in an answer set formally by means of the operator T that computes the closure of a set of literals w.r.t. a GL-reduct. For a DLP P and an interpretation $(\mathcal{H}, M)$ of P , $T_{P_{\mathcal{H}}}^M : \mathcal{L}_{P_{\mathcal{H}}}^M \rightarrow \mathcal{L}_{P_{\mathcal{H}}}^M$ is defined as³ $T(B) = B \cup \{a \mid a \leftarrow \beta \in P_{\mathcal{H}}^M \wedge \beta \subseteq B\}$. Additionally, we have $T^0(B) = B$, and $T^{n+1}(B) = T(T^n(B))$. More detail than the T -operator is provided by the *support* of a literal a in an answer set $(\mathcal{H}, M)$, which explicitly indicates the literals that support the presence of a in the answer set. For the least n such that $a \in T^n$, we inductively define the support $S^k(a)$ on a certain level $1 \leq k \leq n$ as $S^n(a) = \{a\}$ and $S^k(a) = \{\beta \mid b \leftarrow \beta \in P_{\mathcal{H}}^M, \beta \subseteq T^k, b \in S^{k+1}(a)\}$, $1 \leq k < n$. A support for a is then $S(a) = \cup_{k=1}^n S^k(a)$.

Take, for example, the program P with a rule $p(X) \vee \text{not } p(X) \leftarrow$. Grounding w.r.t. to a universe $\{x, y\}$ yields the program $P_{\{x, y\}}$ consisting of $p(x) \vee \text{not } p(x) \leftarrow$ and $p(y) \vee \text{not } p(y) \leftarrow$. We have that $\{p(x)\}$ is an answer set of $P_{\{x, y\}}$, since the GL-reduct is $p(x) \leftarrow$ which has only one minimal model: $\{p(x)\}$ itself. Thus $(\{x, y\}, \{p(x)\})$ is an answer set of P . Actually, a rule such as in P allows one to freely introduce p -literals (provided no other rules constrain this). We call a predicate p *free* if $p(X, Y) \vee \text{not } p(X, Y) \leftarrow$ or $p(X) \vee \text{not } p(X) \leftarrow$ is in the program, for a binary or unary p respectively. Similarly, a ground literal l is free if we have $l \vee \text{not } l \leftarrow$.

A program P is *consistent* if it has an answer set. For a unary predicate p , appearing in P , p is *satisfiable* w.r.t. P if there exists an answer set $(\mathcal{H}, M)$ of P such that $p(a) \in M$ for some $a \in \mathcal{H}$. For a ground literal α , we have $P \models \alpha$ if for all answer sets $(\mathcal{H}, M)$ of P , $\alpha \in M$. Checking whether $P \models \alpha$ is called *query answering*. We can reduce query answering to consistency checking, i.e. $P \models \alpha$ iff $P \cup \{\text{not } \alpha \leftarrow\}$ is not consistent. Consistency checking can be reduced to satisfiability checking, by introducing some new free predicate p .

Finally, note that satisfiability checking for DLPs under the open answer set semantics is undecidable since the undecidable *domino problem*[4] can be reduced to it[17].

3 Adding Grounded Rules to Conceptual Logic Programs

In [17], the syntax of DLPs was restricted in order to regain decidability of reasoning and to enable a reduction of reasoning to normal answer set programming, resulting in *conceptual logic programs (CLPs)*. We recall the intuition and definition of CLPs.

Consider a program P_1 defining when one cheats one’s spouse, i.e. if one is married to someone that is different than the person one is dating. We have a specialized rule saying when one is cheating one’s spouse with the spouse’s friend Jane. Furthermore, some justice is introduced by a constraint ensuring that cheaters will in turn be cheated.

³ We omit the subscript if it is clear from the context and, furthermore, we will usually write T instead of $T(\emptyset)$.

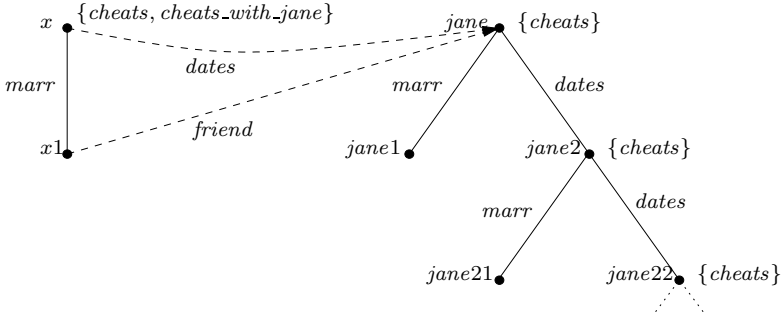


Fig. 1. Forest-Model

$$\begin{aligned}
 \text{cheats}(X) &\leftarrow \text{marr}(X, Y_1), \text{dates}(X, Y_2), Y_1 \neq Y_2 \\
 \text{cheats\_with\_jane}(X) &\leftarrow \text{marr}(X, Y), \text{friend}(Y, \text{jane}), \text{dates}(X, \text{jane}), Y \neq \text{jane} \\
 &\quad \leftarrow \text{cheats}(X), \text{dates}(X, Y), \text{not marr}(X, Y), \text{not cheats}(Y)
 \end{aligned}$$

with *marr*, *friend* and *dates* free predicates. An (infinite) answer set of this program that satisfies *cheats_with_jane* is depicted in Figure 1, where e.g. *cheats* in the label of *x* indicates that *cheats*(*x*) is in the answer set. One sees that *x* cheats his spouse with Jane since *x* dates Jane but is married to *x1*. Furthermore, by the constraint, we must have that Jane is also a cheater, and thus, by minimality of answer sets, we must have that Jane is married to some *jane1* and dates *jane2*, who in turn must be cheating, resulting in an infinite answer set⁴. Formally, a CLP is a DLP consisting of the following types of rules[17]:

- free rules $l \vee \text{not } l \leftarrow$ for a literal *l*,
- unary rules $a(s) \leftarrow \beta(s), \cup_m \gamma_m(s, t_m), \cup_m \delta_m(t_m), \cup_{i \neq j} t_i \neq t_j$, such that, if $\gamma_m \neq \emptyset$ then $\gamma_m^+ \neq \emptyset$, and, in case t_m is a variable: if $\delta_m \neq \emptyset$ then $\gamma_m \neq \emptyset$,
- binary rules $f(s, t) \leftarrow \beta(s), \gamma(s, t), \delta(t)$ with $\gamma^+ \neq \emptyset$ if t is a variable,
- constraints $\leftarrow a(s)$.

where *i* and *j* are within the range of *m*. Note that the example program P_1 is not directly a CLP due to the presence of the literals *marr*(*X*, *Y*), *friend*(*Y*, *jane*) in the second rule where *jane* is not directly connected to *X*, as is required for unary rules. However, we can easily rewrite it as a CLP rule by replacing *friend*(*Y*, *jane*) by some *a*(*Y*) and adding the unary rule $a(Y) \leftarrow \text{friend}(Y, \text{jane})$. In general, programs where the rules have a tree-like body can be easily rewritten as CLPs. Although CLPs allow only constraints of the very simple form $\leftarrow a(s)$ we can easily reduce more complicated constraints $\leftarrow \beta$ to a CLP rule by introducing the unary rule $a(s) \leftarrow \beta$ and $\leftarrow a(s)$.

CLPs were designed to ensure the *forest-model property* (and to a lesser extent the bounded finite model property, cfr. infra). This forest-model property ensures that if a CLP has an answer set where a certain unary predicate is satisfied, then there must be an

⁴ We represent the *n* successors of a node *x*, as *x1*, ..., *xn*.

answer set that has the form of a forest such that the predicate is true at the root of a tree in such a forest. E.g., the answer set in Figure 1 consists of a tree with an *anonymous*⁵ element x as root and the constant *jane* as the root of another tree. It appears that the clean forest structure (i.e. disjoint trees) is perturbed by the connections between x , $x1$ and *jane*. However, it is easy to see that we can encode e.g. $dates(x, jane)$ as $dates^a(x)$ and thus keep $dates^a$ in the label of x . Since there are only a finite number of constants in a program, the labels of the trees are also finite. In effect, a forest-model is a set of trees, with arbitrary connections from elements to constants. As a consequence, the connections between constants, i.e. the roots of the trees, may form an arbitrary graph.

A particular forest-model constructed from an answer set of a program with n constants contains $n + 1$ trees, i.e. one for each constant (which is the root of that tree) and an additional one for some anonymous element that contains the predicate of which satisfiability is being checked.

The rules in a CLP make sure that the forest-model property is valid for CLPs[17]. E.g. one cannot have $p(X) \leftarrow not\ f(X, Y)$, since an answer set $(\{x, y\}, \{p(x)\})$ cannot be transformed into a tree: we have nothing to connect x with y . Similarly, we cannot have $f(X, Y) \leftarrow p(X)$ since, for $p(x)$, this would introduce arbitrary connections between x and all other domain elements y , and thus would clearly violate the tree structure. However, it is allowed to have $p(X) \leftarrow q(a)$ for a constant a , since, intuitively, a is a root of its own tree.

As the tree-like rules impose a rather strict format upon the representation of knowledge, we now extend CLPs by allowing for arbitrary ground DLP rules.

Definition 1. An *extended conceptual logic program (ECLP)* P is a program $Q \cup R$, where Q is a CLP and R is a finite ground DLP. We denote Q with $clp(P)$ and R with $e(P)$.

For example, in addition to P_1 , we may have a rule representing that if Leo is married to Jane, Jane dates Felix, and Leo himself is not cheating, then Leo dislikes Felix: $dislikes(leo, felix) \leftarrow marr(leo, jane), dates(jane, felix), not\ cheats(leo)$. This ground rule does not have a tree structure, it relates the three constants in an arbitrary graph-like manner. Note that the ground rules can be full-fledged DLP, i.e. with negation as failure. Moreover, predicates in $e(P)$ may be defined in the CLP $clp(P)$, as is the case for *marr*, *dates* and *cheats*. Vice versa, we may have predicates appearing in the CLP part that are defined in the ground rule part, e.g. *dislikes* could appear in the CLP part as a *dislikes(X, Y)* literal.

ECLPs still have the forest-model property, since, intuitively, graph-like connections between constants are allowed in a forest, which is all the ground part $e(P)$ of an ECLP P can ever introduce.

Theorem 1. *Extended conceptual logic programs have the forest-model property.*

A forest-model of the example ECLP would be the forest-model of Figure 1 with additionally $\{dislikes(leo, felix), marr(leo, jane), dates(jane, felix)\}$. As for CLPs in

⁵ I.e. a domain element not appearing as a constant in the program.

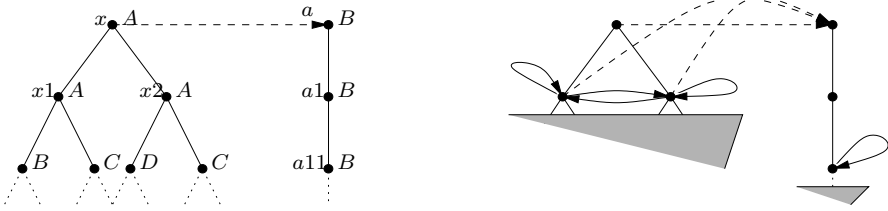


Fig. 2. Cutting a Forest-Model

[17], we would like to establish a bounded finite model property for ECLPs. This property enables the transformation of an (infinite) answer set into a finite one, and, more specifically, it establishes a bound on the number of domain elements that are needed for such a construction. Moreover, this bound depends solely on the program at hand, such that, by introducing a sufficient number of domain elements, we can simulate reasoning with ECLPs by normal finite answer set programming.

We sketch the *cutting* technique from [17] to transform an infinite forest-model into a finite answer set. For every path in a tree in such a forest-model, and every first pair of nodes with equal labels on such a path from the root, we cut away the tree below the second node in the pair and duplicate the outgoing edges of the first node in the second node in the pair. Intuitively, once we encounter on a path a label (a “state”) we already encountered, we act as if in the first occurrence of the label instead of going down the tree thereby ignoring the infinite part. For example, Figure 2 shows the cutting of the forest-model on the left, resulting in the finite answer set on the right. Since x_1 and x_2 have the same label A as x we replace all outgoing edges from x_1 and x_2 with the outgoing edges from x : we have connections from x to x_1 , from x to x_2 , and from x to the constant a . Thus we introduce for x_i , $i \in \{1, 2\}$ connections from x_i to x_1 , from x_i to x_2 , and from x_i to a . The tree with constant root is cut in a similar way, but note that one only starts considering duplicate pairs from below the root and thus (a_1, a_{11}) is the first pair with duplicate labels to consider. This because it might be that a rule $t(a) \leftarrow$ introduces t in the label of a , however, such a rule cannot be used to motivate the presence of t lower in the tree. Below the root, we would not have this problem as t there would be motivated by a rule with head $t(X)$, which can be matched against any lower node.

Taking into account that forest-models have a finite bounded branching, and that on every path we must always encounter duplicate labels after a bounded depth, together with the fact that there are $n + 1$ trees, for n constants, leads to a finite bound k of needed domain elements, which can be read from the program: the branching can be determined from the branching of the unary rules, and the number of possible labels depends on the number of unary predicates in the program. The number of different labels is exponential in the size of the program such that, taking into account the branching of the program, k is in general double exponential.

However, one has to be cautious with this cutting, e.g. the program with rules $a(X) \leftarrow f(X, Y)$, $a(Y)$, and $a(X) \leftarrow b(X)$ with b and f free, has a tree-model⁶

⁶ A tree-model is a forest-model containing only one tree.

$\{a(x), f(x, x1), a(x1), f(x1, x11), a(x11), b(x11)\}$. If one cuts at the first occurrence of a duplicate label, which would be at $x1$ in this case, then $a(x)$ would no longer have a valid support - $b(x11)$ has been cut away - and thus the resulting model would not be minimal. Note that cutting is somewhat similar in spirit to blocking in description logics[3], however, the minimality of answer sets demands some extra precautions, as indicated above.

This problem was solved in [17] for CLPs by enforcing the local model property: forest-models of a CLP should be *locally supported*, i.e. for every literal $q(x)$ ($f(x, y)$) the forest-model can only be motivated by x , one of x 's successors, and/or constants. This way, when we cut the trees we never remove the support of any higher nodes in the tree. An extra condition for local supportedness was that a $g(xi, a)$, although it involves only a successor of x and a constant, cannot be in the support of $q(x)$ ($f(x, y)$) since upon cutting at xi , $g(xi, a)$ could be removed while it provides support for $q(x)$ ($f(x, y)$). In the cheating example we have that the forest-model depicted in Figure 1 is not locally supported since $friend(x1, jane)$ is in the support of $cheats\_with\_jane(x)$ - to derive $cheats\_with\_jane(x)$ we need $friend(x1, jane)$.

In the ECLP case, however, where we have an arbitrary ground part, the local model property of [17] is not sufficient. Take, for example, a rule $doesn\_care(felix) \leftarrow marr(leo, jane), dates(jane, felix), cheats(leo)$, where Felix does not care about dating the married Jane if her husband Leo is cheating as well. Together with the *cheats* rule from the cheating example, one has that $doesn\_care(felix)$ is in an answer set if $marr(leo, jane), dates(jane, felix), cheats(leo), marr(leo, leo1)$, and $dates(leo, leo2)$ for successors $leo1$ and $leo2$ of leo are in the answer set. Thus, although the *cheats* rule in itself does not violate the local model property, adding a ground rule does so, since supports may involve also successors of constants which is not allowed according to the local model property definition for CLPs in [17].

However, cutting of forest-models never removes any successors of constants and, moreover, a successor of a constant is never considered as a candidate for the second node in a duplicate pair since, by definition, the root in a constant tree is not taken into account as a candidate for the first node in a duplicate pair. Thus, we can safely relax the local model property definition from [17] for ECLPs by also allowing successors of constants in the support. In the definition below, we use $\mathcal{H}_{S(l)}$ to denote the domain elements in $S(l)$, the support of l .

Definition 2. A forest-model $(\mathcal{H}, M)$ of an ECLP P is **locally supported** if

$$\forall l = q(x) \in M \vee l = f(x, y) \in M \cdot$$

$$(\mathcal{H}_{S(l)} \subseteq \{x, xi \mid xi \text{ successor of } x\} \cup \{a, ai \mid a \in \mathcal{H}_P, ai \text{ successor of } a\}) \wedge$$

$(\forall f(z, a) \in S(l), a \in \mathcal{H}_P \cdot z \neq xi), p \in upreds(P)$ is **locally satisfiable** w.r.t. P if there is a locally supported forest-model, a **local model** for short, $(\mathcal{H}, M)$ such that $p(\varepsilon) \in M$ for a root ε in $\mathcal{H}$. An ECLP P has the **local model property** if the following holds: if $p \in upreds(P)$ is satisfiable w.r.t. P then it is locally satisfiable.

Thus, a forest-model is locally supported if the support for every $q(x)$ or $f(x, y)$ involves only x itself, successors of x , constants and/or successors of constants. ECLPs with the local model property then have the desired bounded finite model property, i.e. if a (unary) predicate p is satisfiable w.r.t. an ECLP P then it is satisfiable by a finite answer set $(\mathcal{H}, M)$ with $|\mathcal{H}| < k$ where k is solely determined by the program P .

Theorem 2. *Let P be an ECLP with the local model property. Then, P has the bounded finite model property.*

Thanks to this property we can reduce reasoning with ECLPs to normal answer set programming by introducing a sufficiently large bound.

Theorem 3. *Let P be an ECLP with the local model property. $p \in \text{upreds}(P)$ is satisfiable w.r.t. P iff there is an answer set M of $\psi(P)$ containing a $p(x_i)$, $1 \leq i \leq k$, where k is as derived above and $\psi(P) = P \cup \{cte(x_i) \leftarrow \mid 1 \leq i \leq k\}$.*

The local model property characterizes those ECLPs for which reasoning can be reduced to normal finite answer set programming. However, it is a semantical characterization, which makes it non-trivial to recognize ECLPs satisfying this property. We now identify a class of ECLPs, based on their syntactic structure, that have the local model property.

Local CLPs are CLPs where each unary $a(s) \leftarrow \alpha(s), \gamma_m(s, t_m), \beta_m(t_m), t_i \neq t_j$ and each binary $f(s, t) \leftarrow \alpha(s), \gamma(s, t), \beta(t)$ is such that every $b \in \beta_m^+$ is either a free predicate, or if $t_{(m)}$ is a constant, $b(t_{(m)})$ is a free literal, or for every $r : b(u) \leftarrow \text{body}(r)$, $\text{body}(r)^+ = \emptyset$. Intuitively, to prove an $a(s)$ ($f(s, t)$) one needs to descend at most one level in the tree, where one can locally prove $a(s)$ ($f(s, t)$), i.e. without the need to go further down the tree. Of course, in the level below s one may need to check more literals which could amount going further down the tree, but whilst doing this one does not need to remember which literals need to be proven above in the tree - in a way a local CLP is memoryless. In [17] local CLPs were shown to have the local model property.

We then define *local ECLPs* as the union of a local CLP and an arbitrary ground DLP.

Definition 3. *A local ECLP P is an ECLP where $\text{clp}(P)$ is local.*

By the extension of the local model property of CLPs to accommodate for ECLPs, where also successors of constants are allowed in the local support, local ECLPs have the local model property, i.e. the arbitrary ground rules have no influence on the locality.

Theorem 4. *Local ECLPs have the local model property.*

Furthermore, adding a finite number of ground rules to a CLP does not augment the complexity of reasoning.

Theorem 5. *Let P be an ECLP with the local model property. Satisfiability checking w.r.t. P is in 3-NEXPTIME.*

Indeed, we have that the bound k of needed domain elements to simulate reasoning w.r.t. an ECLP P with finite answer set programming is double exponential in the size of P , and thus the size of the translated program $\psi(P)$ (as in Theorem 3) is double exponential in the size of P . Since satisfiability checking w.r.t. $\psi(P)$ is in NEXPTIME w.r.t. the size of the program[9, 5], we have a 3-NEXPTIME bound w.r.t. the size of the original ECLP.

4 Nonmonotonic Ontological and Rule-Based Reasoning with Extended Conceptual Logic Programs

We consider the DL $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ which is the basic DL $\mathcal{ALC}$ with support for role hierarchies ($\mathcal{H}$), nominals/individuals ($\mathcal{O}$), qualified number restrictions ($\mathcal{Q}$), and conjunction ($\sqcap$) and disjunction ($\sqcup$) of roles. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ is a DL related to the ontology language OWL DL[7], extending it in certain aspects and restricting it in others: OWL DL is a notational variant of the DL $\mathcal{SHOIN}(\mathbf{D})$ [18], which adds transitive roles (turning $\mathcal{ALC}$ into $\mathcal{S}$), inverse roles ($\mathcal{I}$), and data types ($\mathbf{D}$) to $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ while removing support for role constructors and qualified number restrictions from it, and allowing only unqualified number restrictions ($\mathcal{N}$).

Formally, the syntax of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ concept and role expressions can be defined as in Table 1 for concept expressions D , E , concept names A , role expressions R , S , role names Q , and nominals o . The semantics is given by a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set, representing the set of available domain elements, and $\cdot^{\mathcal{I}}$ is an interpretation function such that $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and $Q^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for concept names A and role names Q , and every nominal o is mapped to some $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. For complex concept expressions, $\cdot^{\mathcal{I}}$ is defined as in Table 1, where we additionally assume the

Table 1. Syntax and Semantics $\mathcal{ALCHOQ}(\sqcup, \sqcap)$

concept names	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role names	$Q^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
individuals	$\{o\}^{\mathcal{I}} = \{o^{\mathcal{I}}\}$
conjunction of concepts	$(D \sqcap E)^{\mathcal{I}} = D^{\mathcal{I}} \cap E^{\mathcal{I}}$
disjunction of concepts	$(D \sqcup E)^{\mathcal{I}} = D^{\mathcal{I}} \cup E^{\mathcal{I}}$
conjunction of roles	$(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$
disjunction of roles	$(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$
existential restriction	$(\exists R.D)^{\mathcal{I}} = \{x \exists y : (x, y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\}$
universal restriction	$(\forall R.D)^{\mathcal{I}} = \{x \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in D^{\mathcal{I}}\}$
qualified number restriction	$(\leq n R.D)^{\mathcal{I}} = \{x \#\{y (x, y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \leq n\}$
	$(\geq n R.D)^{\mathcal{I}} = \{x \#\{y (x, y) \in R^{\mathcal{I}} \wedge y \in D^{\mathcal{I}}\} \geq n\}$

unique name assumption for nominals, i.e. if $o_1 \neq o_2$, then $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$. Note that OWL does not have the unique name assumption[26], and thus different individuals can point to the same resource. However, the open answer set semantics gives an Herbrand interpretation to constants, i.e. constants are interpreted as themselves, and for consistency we assume that also DL nominals are interpreted this way. Thus, from a Semantic Web point of view, we assume all individuals are URI's that point to a unique resource.

A DL *knowledge base* consists of *terminological axioms* $C_1 \sqsubseteq C_2$ and *role axioms* $R_1 \sqsubseteq R_2$ for concept expressions C_1 and C_2 , and role expressions R_1 and R_2 . Axioms express a subset relation: an interpretation $\mathcal{I}$ *satisfies* an axiom $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ ($R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$). An interpretation is a *model* of a knowledge base Σ if it satisfies every axiom in Σ . A concept C is *satisfiable* w.r.t. Σ if there is a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$.

The ontology layer for the Semantic Web is becoming a reality with languages such as OWL DL. Consequently, the rule layer, which provides additional inferencing capabilities on top of DL reasoning, is gaining interest in the Semantic Web community. For example, in [23], integrated reasoning of DLs with *DL-safe* rules was introduced. DL-safe rules are unrestricted Horn clauses where only the communication between the DL knowledge base and the rules is restricted; they enable one to express knowledge inexpressible with DLs alone, e.g. triangular knowledge such as [23]

$$BadChild(X) \leftarrow Grandchild(X), parent(X, Y), parent(Z, Y), hates(X, Z)$$

saying that a grandchild that hates its sibling is a bad child.

We introduce DL-safe rules as in [23]. For a DL knowledge base Σ let N_C and N_R be the concept and role names in Σ and N_P is a set of predicate symbols such that $N_C \cup N_R \subseteq N_P$. A *DL-atom* is an atom of the form $A(s)$ or $R(s, t)$ for $A \in N_C$ and $R \in N_R$. A *DL-safe rule* is a rule of the form $a \leftarrow b_1, \dots, b_n$ where a, b_i are atoms and every variable in the rule appears in a non-DL-atom in the rule body. A *DL-safe program* is a finite set of DL-safe rules. Let $cts(\Sigma, P)$ be the set of nominals in Σ and constants in P .

The semantics of the combined (Σ, P) for a knowledge base Σ and a DL-safe program P is given by interpreting Σ as a first-order theory $\pi(\Sigma)$, see e.g. [8], every DL-safe rule $a \leftarrow b_1, \dots, b_n$ as the clause $a \vee \neg b_1 \vee \dots \vee \neg b_n$, and then considering the first-order interpretation of $\pi(\Sigma) \cup P$. The main reasoning procedure in [23] is *query answering*, i.e. checking whether a ground atom α is true in every first-order model of $\pi(\Sigma) \cup P$, denoted as $(\Sigma, P) \models \alpha$.

We provide an alternative semantics based on DL interpretations as in [19] rather than on first-order interpretations. However, both semantics are compatible as indicated in [23]. For (Σ, P) and an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of Σ we extend $\cdot^{\mathcal{I}}$ for N_P and $\mathcal{H}_P$ such that for unary predicates $p \in N_P$, $p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, for binary predicates $f \in N_P$, $f^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for $o \in \mathcal{H}_P$; such an extended interpretation is, by definition, an interpretation of (Σ, P) . Furthermore, we impose the unique name assumption such that if $o_1 \neq o_2$, then $o_1^{\mathcal{I}} \neq o_2^{\mathcal{I}}$, for elements $o \in cts(\Sigma, P)$. A *binding* for an interpretation $\mathcal{I}$ of (Σ, P) is a function $\sigma : vars(P) \cup cts(\Sigma, P) \rightarrow \Delta^{\mathcal{I}}$ with $\sigma(o) = o^{\mathcal{I}}$ for $o \in cts(\Sigma, P)$; it maps constants/nominals and variables to domain elements. A unary atom $a(s)$ is then true w.r.t. σ and $\mathcal{I}$ if $\sigma(s) \in a^{\mathcal{I}}$, and a binary atom $f(s, t)$ is true w.r.t. σ and $\mathcal{I}$ if $(\sigma(s), \sigma(t)) \in f^{\mathcal{I}}$. A rule r is satisfied by $\mathcal{I}$ iff for every binding σ w.r.t. $\mathcal{I}$ that makes the atoms in the body true, the head is also true. An interpretation of (Σ, P) is a model if it is a model of Σ and it satisfies every rule in P . Query answering $(\Sigma, P) \models \alpha$ amounts then to checking whether for every (DL) model $\mathcal{I}$ of (Σ, P) , the ground atom α is true in $\mathcal{I}$.

In [17], $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ satisfiability checking is reduced to CLP satisfiability checking. Here we reduce query answering w.r.t. $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ extended with DL-safe rules to query answering w.r.t. ECLPs. We first provide some intuition with an example. Take a knowledge base $\Sigma = \{\exists \text{manuf\_in.Co} \sqcap \exists \text{has\_price} \sqsubseteq \text{Product}\}$, expressing that if something is manufactured in some country and it has a price then it is a product. We have some facts in a DL-safe program P about the world we are considering:

$$\begin{array}{ll} is\_product\_of(p, c_1) \leftarrow & manif\_in(p, japan) \leftarrow \\ is\_product\_of(p, c_2) \leftarrow & Co(japan) \leftarrow \end{array}$$

saying that p is a product of company c_1 and company c_2 , that p is manufactured in Japan and that Japan is a country. Those facts are vacuously DL-safe since they do not contain variables. Additionally, we have a DL-safe rule in P saying that if a product is a product of 2 companies then those companies are competitors⁷, $r_1 : competitors(C_1, C_2) \leftarrow Product(P), is\_product\_of(P, C_1), is\_product\_of(P, C_2)$. Note that this is indeed a DL-safe rule since every variable occurs in a $is\_product\_of$ atom, which is a non-DL-atom in the body of the rule. The only DL-atom in the rule is $Product(P)$. A possible model $\mathcal{I}$ of (Σ, P) would be $\mathcal{I} = (\{japan, c_1, c_2, p, x\}, \cdot^{\mathcal{I}})$ ⁸ with $\cdot^{\mathcal{I}} : Co^{\mathcal{I}} = \{japan\}, Product^{\mathcal{I}} = \{p\}, manif\_in^{\mathcal{I}} = \{(p, japan)\}, has\_price^{\mathcal{I}} = \{(p, x)\}, is\_product\_of^{\mathcal{I}} = \{(p, c_1), (p, c_2)\}$ and $competitors^{\mathcal{I}} = \{(c_1, c_2)\}$.

We translate (Σ, P) now to an ECLP: the DL axiom is translated to the constraint $\leftarrow (\exists manif\_in.Co \sqcap \exists has\_price)(X), not\ Product(X)$, where we introduce predicates corresponding to the concept expressions. Furthermore, we define these predicates by the rules

$$\begin{array}{l} (\exists manif\_in.Co \sqcap \exists has\_price)(X) \leftarrow (\exists manif\_in.Co)(X), (\exists has\_price)(X) \\ (\exists manif\_in.Co)(X) \leftarrow manif\_in(X, Y), Co(Y) \\ (\exists has\_price)(X) \leftarrow has\_price(X, Y) \end{array}$$

such that if an answer set contains $(\exists manif\_in.Co \sqcap \exists has\_price)(x)$, then, by minimality of answer sets and the first rule, $(\exists manif\_in.Co)(x)$ and $(\exists has\_price)(x)$ are in the answer set, and, by the second and third rule, there must be a y_1 and a y_2 such that $manif\_in(x, y_1)$, $Co(y_1)$, and $has\_price(x, y_2)$ are in the answer set. The opposite direction is also valid, i.e. if $manif\_in(x, y_1)$, $Co(y_1)$, and $has\_price(x, y_2)$ are in the answer set then $(\exists manif\_in.Co \sqcap \exists has\_price)(x)$ is in the answer set since rules need to be satisfied. This kind of behavior exactly mimics the DL semantics of the corresponding constructs. Furthermore, we introduce the concept and role names by means of free rules, indicating that a domain element (or a pair of domain elements) is of a certain type or not.

$$\begin{array}{l} Product(X) \vee not\ Product(X) \leftarrow \\ Co(X) \vee not\ Co(X) \leftarrow \\ manif\_in(X, Y) \vee not\ manif\_in(X, Y) \leftarrow \\ has\_price(X, Y) \vee not\ has\_price(X, Y) \leftarrow \end{array}$$

This concludes the CLP part of the translation of (Σ, P) . The ground DLP part consists of the same facts as in the DL-safe part; it also contains the grounding of the rule r_1 in P with constants $\{japan, p, c_1, c_2\}$, e.g. the rule

$$r_2 : competitors(c_1, c_2) \leftarrow Product(p), is\_product\_of(p, c_1), is\_product\_of(p, c_2)$$

⁷ Actually, to correspond entirely with the desired semantics, we would need to indicate that C_1 and C_2 are different companies. This seems to be not possible with the DL-safe rules in [23], however, it is with ECLPs using $\neq$.

⁸ We take $o^{\mathcal{I}} = o, o \in cts(\Sigma, P)$, for ease of notation.

Since DL-safe rules have a first-order interpretation one may have that $(c_1, c_2) \in \text{competitors}^{\mathcal{I}}$ for a model $\mathcal{I}$ of (Σ, P) without any justification in $\mathcal{I}$, i.e. the body of r_1 in P does not need to be satisfied in order to have $(c_1, c_2) \in \text{competitors}^{\mathcal{I}}$. The answer set semantics however only deduces $\text{competitors}(c_1, c_2)$ in an answer set if e.g. the body of r_2 is satisfied in that answer set, since otherwise the answer set would not be minimal (one could omit $\text{competitors}(c_1, c_2)$ and still have an answer set).

To solve this, we introduce for each head a of a rule in the ground DLP part, a free rule $a \vee \text{not } a \leftarrow$, e.g. $\text{competitor}(c_1, c_2) \vee \text{not competitor}(c_1, c_2) \leftarrow$ such that one has always a motivation for $\text{competitor}(c_1, c_2)$, mimicking the first-order semantics.

We refer to [17] for the definition of the closure $\text{clos}(\Sigma)$ of a $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ , but basically, for a concept expression D in Σ it includes the subconcepts of D . Formally, we define the CLP $\Phi_1(\Sigma, P)$ for a $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ and a DL-safe program P as the program containing for every concept expression $D \in \text{clos}(\Sigma)$ the rules in Table 2. Furthermore, for every concept

Table 2. CLP Translation $\Phi_1(\Sigma, P)$

$\neg D(X) \leftarrow \text{not } D(X)$	$D \sqcap E(X) \leftarrow D(X), E(X)$
$D \sqcup E(X) \leftarrow D(X)$	$D \sqcup E(X) \leftarrow E(X)$
$\exists R.D(X) \leftarrow R(X, Y), D(Y)$	$\forall R.D(X) \leftarrow \text{not } \exists R.\neg D(X)$
$R \sqcup S(X, Y) \leftarrow R(X, Y)$	$R \sqcap S(X, Y) \leftarrow R(X, Y), S(X, Y)$
$R \sqcup S(X, Y) \leftarrow S(X, Y)$	$(\leq n R.D)(X) \leftarrow \text{not } (\geq n + 1 R.D)(X)$
$(\geq n R.D)(X) \leftarrow R(X, Y_1), \dots, R(X, Y_n), D(Y_1), \dots, D(Y_n), Y_1 \neq Y_2, \dots$	

name A and role name Q in Σ , we add the free rules $A(X) \vee \text{not } A(X) \leftarrow$ and $R(X, Y) \vee \text{not } R(X, Y) \leftarrow$. Nominals o in Σ are handled by introducing predicates $\{o\}$ with facts $\{o\}(o) \leftarrow$ in $\Phi_1(\Sigma, P)$, such that we can only have that $\{o\}(x)$ is in an answer set if $x = o$. $\Phi_1(\Sigma, P)$ is not a local ECLP, but due to the fact that the body of a rule becomes structurally smaller one can transform it to a local ECLP while preserving satisfiability[17].

We define $\Phi_2(\Sigma, P)$ as the ground DLP $P_{\text{cts}(\Sigma, P)}$, i.e. P grounded with all constants and nominals in Σ and P , together with free rules $\text{head}(r) \vee \text{not head}(r) \leftarrow$ for each $r \in P_{\text{cts}(\Sigma, P)}$.

Theorem 6. *For an $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ knowledge base Σ and a DL-safe program P , we have $(\Sigma, P) \models \alpha$ iff $\Phi_1(\Sigma, P) \cup \Phi_2(\Sigma, P) \models \alpha$.*

In [23] the $\mathcal{SHOIN}(\mathbf{D})$ DL is considered instead of $\mathcal{ALCHOQ}(\sqcup, \sqcap)$, which extends and at the same time restricts the type of allowed constructors. DL-safe rules allow for variables, however, this does not make them more expressive than ground DLP programs: [23] proves that $(\Sigma, P) \models \alpha$ iff $(\Sigma, P^g) \models \alpha$ where P^g is the grounding of P w.r.t. constants and nominals in (Σ, P) . Moreover, using ECLPs instead of a DL knowledge base with DL-safe rules on top has the further advantage of nonmonotonicity by

means of negation as failure in both the CLP part and the grounded DLP part, whereas both DLs and DL-safe rules are monotonic (DL-safe rules are Horn clauses and thus do not allow for negation as failure).

5 Related Work

We highlight some of the current research trends on the application of nonmonotonicity to the Semantic Web and refer the reader for further related work on the combination of (not necessarily nonmonotonic) rules and ontologies to [17].

In [2], one builds a nonmonotonic rule system on top of the ontology language DAML+OIL[6], a predecessor of OWL. More specifically, they use *defeasible logic*[24] to express rule-based knowledge and argue its use for E-commerce applications on the Semantic Web. Another approach combining DAML+OIL with rules can be found in [15], where *situated courteous logic programs* in the rule markup language RuleML[1] provide for the nonmonotonic rule system.

[10] combines the expressive $\mathcal{SHOIN}(\mathbf{D})$, i.e. OWL DL, with ASP reasoning by considering the DL knowledge base as a black box that can be queried from the rules. Moreover, inferences made by rules can serve as input to the DL knowledge base as well, leading to a bidirectional flow of information. A disadvantage of this approach, as was remarked in [23], is that, since one considers only consequences of the DL knowledge base, i.e. atoms that are true in all models, some more fine-grained inferences will not be made by the rules. Since reasoning with CLPs can be reduced to finite ASP, it can be trivially reduced to the approach in [10] with an empty DL knowledge base. In [11] the approach of [10] was adapted for the well-founded semantics instead of the answer set semantics.

[14] explains how reasoning with SWRL[20], i.e. OWL extended with Datalog in RuleML, can be done by iteratively calling the DL reasoner RACER[16] and the rule-based reasoner Jess[12], each feeding the other with the inferences it made. Since SWRL is undecidable, and such an iterative procedure is thus incomplete, it shows that intractable worst-case complexity (or even undecidability) should not hold one back to device practical and useful combined reasoners. A similar iterative angle is taken in [22] where SWRL is extended with negation as failure and equipped with an answer set semantics, resulting in a nonmonotonic but undecidable system.

6 Conclusions and Directions for Further Research

We extended CLPs with a finite set of arbitrary ground DLP rules, and showed that reasoning with the resulting ECLPs can be reduced to finite answer set programming. We established an upper complexity bound and simulated reasoning in a DL equipped with DL-safe rules.

The upper 3-NEXPTIME bound for reasoning with ECLPs is rather bad, however, encouraged by practical algorithms for highly intractable DL algorithms, we believe that, using heuristics, one can also implement practical reasoners for ECLPs. This is subject for further research.

References

1. The Rule Markup Initiative. <http://www.ruleml.org>.
2. G. Antoniou. A Nonmonotonic Rule System using Ontologies. CEUR Proceedings, 2002.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
4. F. Baader and U. Sattler. Number Restrictions on Complex Roles in Description logics. In *Proc. of KR-96*, pages 328–339, 1996.
5. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
6. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not Enough. In *Proc. of the First Semantic Web Working Symposium (SWWS'01)*, pages 151–159. CEUR, 2001.
7. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. <http://www.w3.org/TR/owl-ref/>, 2004.
8. A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.
9. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
10. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with DLs for the Semantic Web. In *Proc. of KR 2004*, pages 141–151, 2004.
11. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-Founded Semantics for Description Logic Programs in the Semantic Web. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 81–97. Springer, 2004.
12. E.J. Friendman-Hill. Jess homepage. <http://herzberg.ca.sandia.gov/jess/>.
13. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of ICLP'88*, pages 1070–1080. Cambridge, Massachusetts, 1988. MIT Press.
14. C. Golbreich. Combining Rule and Ontology Reasoners for the Semantic Web. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 6–22. Springer, 2004.
15. B. N. Grosz and T. C. Poon. SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proc. of WWW 2003*, pages 340–349. ACM Press, 2003.
16. V. Haarslev and R. Moller. Description of the RACER System and its Applications. In *Proc. of Description Logics 2001*, 2001.
17. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Semantic Web Reasoning with Conceptual Logic Programs. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 113–127. Springer, 2004.
18. I. Horrocks and P. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *J. of Web Semantics*, 2004. To Appear.
19. I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of WWW 2004*. ACM, 2004.
20. I. Horrocks, P. F. Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean. SWRL: A Semantic Web Rule language Combining OWL and RuleML, May 2004.
21. N. Leone, W. Faber, and G. Pfeifer. DLV homepage. <http://www.dbai.tuwien.ac.at/proj/dlv/>.
22. J. Mei, S. Liu, A. Yue, and Z. Lin. An Extension to OWL with General Rules. In *Proc. of RuleML 2004*, number 3323 in LNCS, pages 6–22. Springer, 2004.

23. Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. In *Proc. of ISWC 2004*, number 3298 in LNCS, pages 549–563. Springer, 2004.
24. D. Nute. Defeasible Logic. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 3)*, pages 353–395. Clarendon Press, 1994.
25. P. Simons. Smodels homepage. <http://www.tcs.hut.fi/Software/smodels/>.
26. M. Smith, C. Welty, and D. McGuinness. OWL Web Ontology Language Guide. <http://www.w3.org/TR/owl-guide/>, 2004.

Product Information Meta-search Framework for Electronic Commerce Through Ontology Mapping

Wooju Kim¹, Dae Woo Choi², and Sangun Park³

^{1,2}Department of Information Industrial Engineering, Yonsei University,
134 Shin-Chon Dong, Seoul 120-749, Korea
wkim@yonsei.ac.kr, qorwkr@nate.com

³Department of Management Engineering, KAIST,
207-43 Cheongryang, Seoul 130-012, Korea
mascon@kgsn.kaist.ac.kr

Abstract. The Semantic Web and Web services provide many opportunities in various applications such as product search and comparison in electronic commerce. We implemented an intelligent meta-search and comparison system for products through consideration of multiple attributes by using ontology mapping and Web services. Under the assumption that each shopping mall offers product ontology and a product search service with Web services, we proposed a meta-search framework to configure a customer's purpose, make and dispatch proper queries to each shopping mall, evaluate search results from malls, and show the customer the product list with a ranking. Ontology mapping is used for generating proper queries for malls that have different taxonomies of product categories. Also we implemented an inference based search engine using ontology and Web services for each mall.

1 Introduction

The Semantic Web and Web services provide many opportunities in various applications using the Internet. We expect that product search and comparison in electronic commerce will be one of the killer applications of such technologies. So far most product comparison systems over multiple shopping malls depend on a keyword-based search or manual collection of comparison information [13]. The meta-search engine [6, 10, 16] is one of those systems. However, shopping malls are offering their own search and directory service for their products in their Web pages, so the keyword-based meta-search for those malls may become redundant. Moreover, it brings inaccurate results because of difficulties of natural language processing [9].

Another problem of the keyword-based search is that it is very hard to configure a customer's exact preference for the product. For example, keywords are not enough to describe a customer's priorities and criteria for a product attribute such as 'a television whose monitor size is larger than 15 inches'. Therefore, it is difficult to provide an exact matching product to the customer because of incomplete representations of a customer's preference.

One more problem is that we can hardly evaluate and compare products with a keyword-based search from the customer's perspective. There have been increasing

efforts [1, 2, 4] in industrial and academic areas to overcome the above problems, but they still have limitations. For example, Mysimon.com categorizes and sorts products by various attributes in order to help customers easily understand product information. But it is difficult for a customer to sort and compare products based on multiple attributes at the same time.

We want to note that Web services can provide a new framework of the meta-search engine. Amazon.com is offering Web services that allow a variety of functions from retrieving information about products to adding an item to a shopping cart. As more shopping malls are expected to offer Web services, the meta-search engine will be able to use Web services to collect structured product information. Especially when the customer wants to compare products based on various attributes rather than simple price, Web services will be the most appropriate alternative for the meta-search infrastructure.

However, there are still some problems in making and sending a proper query that represents customer preference of a product for each shopping mall, integrating and comparing the results from various malls with Web services. These problems result from different taxonomies of shopping malls for product category names and product attributes. To overcome the problem of different taxonomy, the meta-search system needs to support semantic interoperability among malls. While the standardization and integration of ontology are still progressing, it seems impossible that a single taxonomy will be used in every mall. Therefore, we employed ontology mapping techniques [3, 5, 8] to assure semantic interoperability among malls. Ontology mapping is the process in which we start with two source ontologies and generate a new ontology that includes and reconciles all the information from the two source ontologies according to those semantic relations [12].

In this paper, we will suggest a meta-search and comparison framework under which we can consider multiple attributes of a customer's decision making when buying. This framework uses Web services of shopping malls to acquire structured product information, and compare products based on multiple attributes. Also we use ontology mapping to handle different taxonomies. We named the meta-search and comparison system Intelligent Product Information Search (in short, IPIS).

Section 2 briefly describes the architecture of IPIS. Section 3 explains the intelligent product search and comparison procedure using IPIS. Section 4 shows the prototype and search results. The last section provides conclusions.

2 Overview of the IPIS System

The objective of our research is to develop a meta-search framework capable of configuring a customer's preference and capable of finding and evaluating products based on the customer's preference from various shopping malls. To achieve this goal, the IPIS system should guarantee not only syntactic interoperability but also semantic interoperability between IPIS and shopping malls. The structure is as follows:

2.1 Syntactic Interoperability with Web Services

A keyword-based search cannot thoroughly interpret the contents of Web pages due to the limitation of natural language processing. Therefore it is quite hard to get

structured product information by a keyword-based search. But Web services can provide a method to gather structured information represented with XML from various malls. Syntactic interoperability in our approach means that we can get structured product information by sending a structured query to each shopping mall. Also we can easily construct a meta-search engine by using Web services because it is easy to send queries to and receive responses from multiple shopping malls with Web services. But, syntactic interoperability does not deal with matters of semantic interpretation [18]. Therefore even if we get structured product information, it does not guarantee that we can interpret the content when shopping malls use different taxonomies. With this regard, semantic interoperability is required to interpret the product information.

2.2 Semantic Interoperability with Ontology Mapping

Semantic interoperability is “the ability of information systems to exchange information on the basis of shared, pre-established and negotiated meanings of terms and expressions” [18]. Ontology mapping is one of the techniques for semantic interoperability. The difference of taxonomies among shopping malls brings about the following problems.

The first problem is the difference of product category names. For example, in the case of ‘television’, it is represented as ‘television’ in Open Directory Project taxonomy [14], while represented as ‘TV & HDTV’ in Amazon.com. The second problem is the difference of product attribute names. In addition to product names, product attributes are different in each shopping mall. The third one is the difference of data types and units of product attributes. For the same product attribute, each shopping mall may use different data types or units.

The purpose of using ontology mapping in our approach is to interpret different product category names, attribute names, data types and units of attributes. Also we are going to compare products based on that interpretation. Therefore, ontology mapping is separated into product category name mapping and attribute name mapping in the IPIS system.

2.3 Architecture of the IPIS System

IPIS System architecture (see Figure 1) consists of four key components. First, the *Interface* represents a customer’s search purpose in a structured format. The customer can input the desired product, attributes, and priorities of attributes in the *Interface*. The *Query Generator* transforms the customer’s configuration into queries for each shopping mall by using ontology mapping because shopping malls have different taxonomies. Shopping malls process the queries to search for proper products satisfying the customer’s input and return detailed information of the products to the *Product Evaluation Agent* through Web services. The *Product Evaluation Agent* evaluates and compares the results from shopping malls and shows the customer the list of products with a resulted ranking and attributes. Detailed functions of each module will be described in section 3.

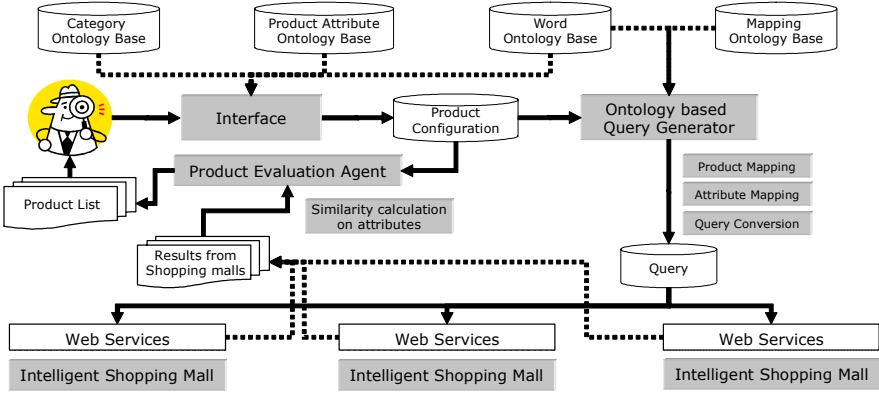


Fig. 1. Architecture of the IPIS System

2.3.1 Interface

The *Interface* is used for the precise configuration of a customer’s preference for a product. In order to represent a customer’s configuration, we used the Open Directory Project as the category ontology base and product attribute ontology base. The Open Directory Project (in short, ODP) is the largest, most comprehensive human-edited directory of the Web. It is constructed and maintained by the vast, global community of volunteer editors [14]. Figure 2 shows an example of product categories and attributes related to ‘Consumer Electronics’ in ODP. The customer can select the desired product in a given product category hierarchy. The customer can also input product attributes, constraints, and priorities by using the *Interface*.

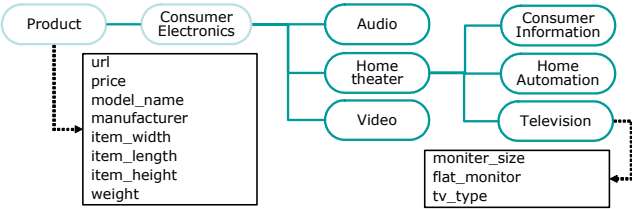


Fig. 2. Product Category and Attribute Example of the Open Directory Project

2.3.2 Query Generator

The *Query Generator* is the most important module in the IPIS system because an appropriate query for each shopping mall can give an appropriate answer. To make an appropriate query, the IPIS system transforms an original query generated from the customer’s configuration into individual queries for each mall because they are using different taxonomies for a product category. Figure 3 shows an example of original query and modified queries for each mall which are represented with RDQL [15].

The mapping ontology base is a mapping table constructed with matching product category names and product attribute names of shopping malls for the words of the ODP ontology. The purpose of the mapping ontology base is to enhance ontology mapping speed. The word ontology base is used to extend product category names

and attributes in ontology mapping. We constructed word ontology with the WordNet [11] ontology.

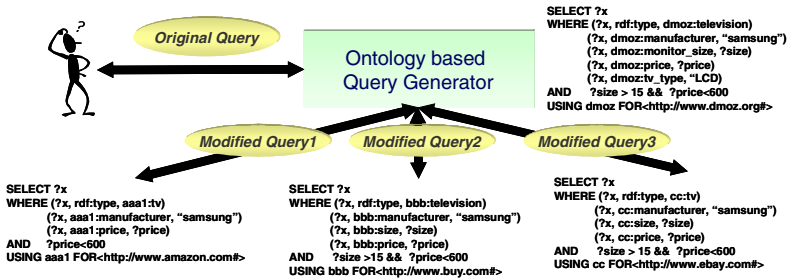


Fig. 3. An Example of Interpreted Queries for Various Malls

2.3.3 Product Evaluation Agent

The *Product Evaluation Agent* analyzes the product list from shopping malls, evaluates the products based on the customer’s configuration, and makes the sorted product list for the customer. The *Product Evaluation Agent* consists of two modules. The first module extracts the values of product attributes from the product list, and the second module evaluates the products by calculating similarities of extracted attributes based on the customer’s configuration.

2.3.4 Intelligent Shopping Mall

The *Intelligent Shopping Mall* receives queries from the *Query Generator* and returns the search results to the *Product Evaluation Agent* after query execution. The search system of the *Intelligent Shopping Mall* exploits an inference based search engine as one of the Semantic Web search engines that can perform various searches on product properties like a database query.

Figure 4 shows the architecture of an intelligent shopping mall using Web services. The inference based search engine executes the query received from the IPIS system. The category ontology base and the product ontology base store product category ontology and product attribute ontology respectively. Rules for inference which describe the relation between categories and attributes are stored in the rule base which consists of Jena rules for OWL axioms for reasoning [7, 17]. The product data base stores detailed product information of the shopping mall.

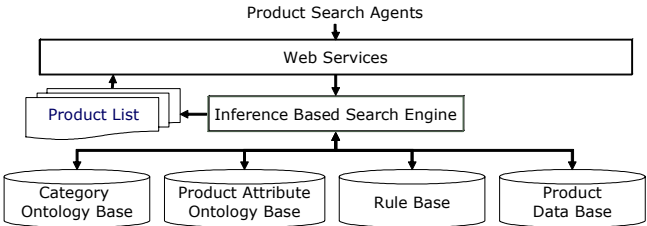


Fig. 4. Architecture of an Intelligent Shopping Mall using Web Services

3 Intelligent Product Information Search Procedure with Examples

This section describes the product search procedure of the IPIS system. The IPIS procedure consists of four steps: configuration of a customer’s purpose, query generation, query execution, and product evaluation.

For a better understanding of the procedure, let us demonstrate the whole process with a consistent example of product search. In our example, the customer’s search purpose is to find a television whose price is less than \$600, monitor size is larger than 15 inches, TV type is ‘LCD’ and manufacturer is ‘Samsung’.

3.1 The Configuration of a Customer’s Search Purpose

Let us describe the configuration process. First, a customer selects the desired product from the category ontology base, and then inputs attributes and his own priorities of attributes. For example, a customer selects ‘television’ from category ontology and selects ‘monitor size’, ‘manufacturer’, ‘tv type’, and ‘price’ as concerned attributes for decision-making. Then, the customer inputs the criteria of attributes such as ‘> 15 inches’ for ‘monitor size’ and ‘< \$600’ for ‘price’. Finally, the customer assigns his personal priority to attributes with numbers between 1 and 10 such as 8 for ‘price’ and 10 for ‘monitor size’ which means that ‘monitor size’ is more important to him than ‘price’. We employ a semantic tree structure to represent the customer’s configuration including product category, names, attributes, criteria, and priorities of attributes in structured form. Figure 5 shows an example of the customer’s configuration with a semantic tree structure.

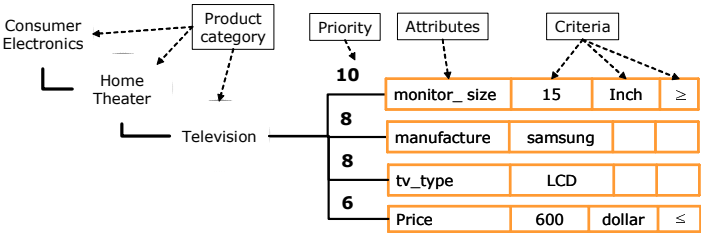


Fig. 5. An Example of a Semantic Tree for Customer Configuration

3.2 Query Generation using Ontology Mapping

An RDQL query can be generated from the semantic tree that represents the configuration of a customer’s search purpose as follows.

```
SELECT ?x
WHERE (?x, rdf:type, dmoz:television)
      (?x, dmoz:manufacturer, "samsung")
      (?x, dmoz:monitor_size, ?size)
      (?x, dmoz:price, ?price)
      (?x, dmoz:tv_type, "LCD")
AND   ?size > 15 && ?price < 600
USING dmoz FOR <http://www.dmoz.org#>
```

The purpose of the query is to ask shopping malls about product information with the customer's configuration. But each shopping mall may not give an appropriate answer to the above query because it uses a different taxonomy from ODP. Therefore, it is necessary to interpret the current query into the form that each shopping mall can understand.

To solve the above problems, the query generation procedure consists of three modules: *Product Mapping*, *Attribute Mapping*, and *Query Conversion* as shown in the flow chart of Figure 6. Let us describe the detailed procedure of *Query Generation*.

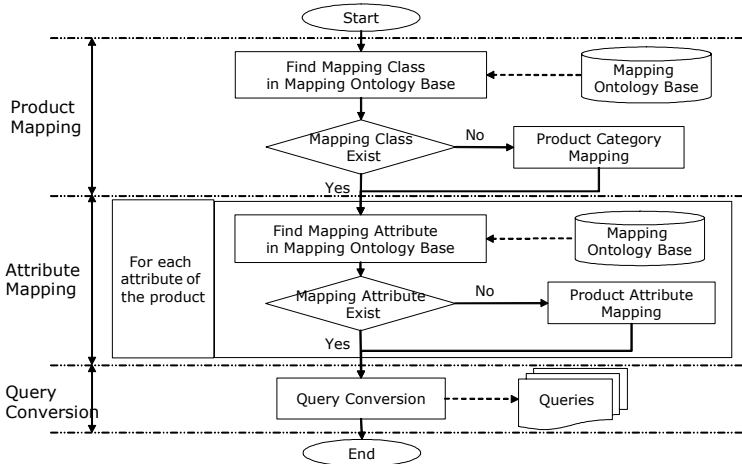


Fig. 6. A Flow Chart for the Query Generation Procedure

3.2.1 Product Mapping

The first step of *Product Mapping* searches for the matching product category name of each shopping mall in the mapping ontology base. If a matching product exists in the mapping ontology base, the time and cost of product mapping can be saved. In our example, there was no exact matching name for ‘television’ in the mapping ontology base. If there is no exact match like in the example, the module looks up the matching product category name from each shopping mall’s ontology in real time.

The procedure of searching for a similar product category name cannot be completed with simple string matching because each shopping mall has a different taxonomy from that of ODP. For example, the product represented as ‘television’ in ODP is represented in Amazon.com as ‘TV & HDTV’. In this case, we can solve the problem by using synonyms. But unconditional use of synonyms and similar words can cause a risk of selecting unexpected products because a word may have different meanings. For example, there are three different meanings of ‘television’ in WordNet. The extension of the product category name uses synonyms and coordinate terms in WordNet. But synonyms and coordinate terms can also be changed as the meaning of the product category name is changed. Therefore we need to determine the exact meaning of the product category name in the configuration and extend the product category name with appropriate synonyms and coordinate terms.

Product Mapping consists of detailed steps like analysis, extension, searching, and choosing. Let us describe these detailed steps.

(1) Analysis and extension of the selected product category name: We used hypernyms of the product category from WordNet to analyze the exact meaning of the product category name. In the example of ‘television’, to find out which one is right for the customer’s configuration among three meanings, we compared a serial category hierarchy that starts from the product category to the root category in ODP to the hypernym hierarchies of three different senses of the product category name in WordNet. Figure 7 shows the category hierarchy of ‘television’ in ODP and three category hierarchies of different meanings for ‘television’ in WordNet.

In Figure 7, WordNet hierarchy 2 has the same word ‘electronic’ in the middle level ‘electronic equipment’ with ‘Consumer electronics’ in the ODP hierarchy. In this manner, the analysis algorithm chooses WordNet hierarchy 2 as the exact meaning of ‘television’.

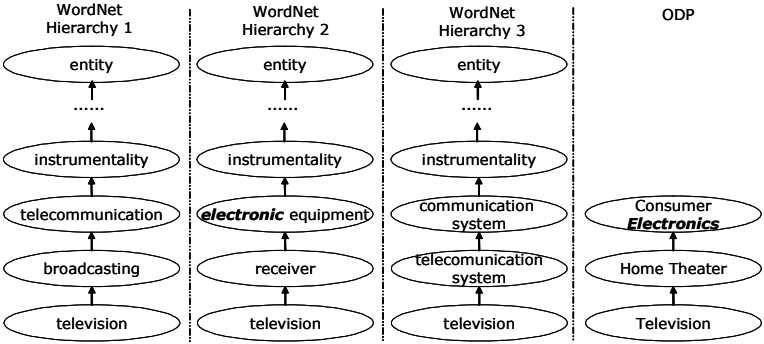


Fig. 7. Comparison between Category Hierarchies of WordNet and ODP

Once the exact sense of the product category name is determined, it extracts the coordinate terms of the selected meaning as an extension set of the product category name from WordNet. For example, the module brings {television receiver, television, television set, tv, tv set, idiot box, boob tube, telly, goggle box} from the WordNet ontology as an extension set of ‘television’.

(2) A Search for Matching Product Category Name: A search starts on each mall’s ontology with an extension set of ‘television’ that was generated in the previous step. We got ‘TV & HDTV’, ‘Internet TV’, and ‘All TVs’ from Amazon.com ontology. After the completion of search for category names, we need to delete redundant category names for the product. To do this, the algorithm generates serial hierarchies for category names by extracting all upper categories starting from the obtained category to the root category. We generated three hierarchies in our example as follows:

/Product/Electronics/Gadgets/Internet Appliances/Internet TV
 /Product/Electronics/Audio & Video/TV & HDTV
 /Product/Electronics/Audio & Video/TV & HDTV/All TVs

From the above result, we can identify that ‘All TVs’ is a subcategory of ‘TV & HDTV’. In this case, we delete the subcategory from the list to avoid a redundant search.

(3) Choice of Matching Product Category Name: After the completion of the search for matching product category names, we should decide which product category names are appropriate for the original product name. We compared the hierarchy of ‘television’ in ODP with the hierarchies of product category names in Amazon.com as in Figure 8 to choose proper names.

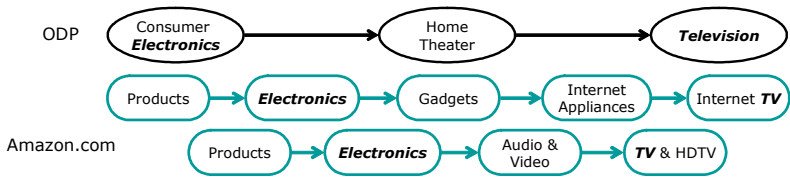


Fig. 8. A Comparison between Hierarchies of Product Category Names and ODP

With the direct comparison of words in the upper categories, we may not decide which one is more appropriate because both hierarchies have ‘electronics’. To find a more appropriate name, we extended upper category names by using synonyms and coordinate terms in WordNet ontology. From the result of upper category name extension, we found that ‘Audio & Video’ in the Amazon.com ontology is similar to ‘Home Theater’ in ODP. So, we selected ‘Home Theater’ as the final matching product category name. We constructed two measures - co-occurrence and order consistency - to estimate the similarity of the product category name to the product name in ODP. Co-occurrence is the calculation of all the similarities of product names by using the similarity of each category name in each category hierarchy. Order consistency is the calculation of the similarity based on comparison between a category order of the ODP hierarchy and category orders of the two hierarchies in Amazon.com. Even if two hierarchies have the same categories, they may have a different similarity along their category order.

Once *Product Mapping* is completed, the matching product category name is stored in the mapping ontology base to enhance next search and avoid the repetition of the same procedure for the same product category name.

3.2.2 Attribute Mapping

After the completion of *Product Mapping*, the *Attribute Mapping* module searches for matching attribute names in a similar manner.

The *Attribute Mapping* procedure is almost the same as the *Product Mapping* procedure except *Attribute Mapping* does not consider the hierarchy of attribute names because there is no attribute hierarchy in the shopping mall ontology. Therefore the similarity is calculated based on only the attribute name. First of all, the attribute mapping module tries to find a matching attribute name from the mapping ontology base. If it fails, the module starts the main attribute mapping procedure by finding matching attribute names in each shopping mall’s ontology. If it also fails, then it extends the attribute name by using the WordNet ontology like *Product Mapping* as

shown in section 3.2.1, and searches for similar attribute names in each shopping mall's ontology with synonyms and coordinate terms of the original attribute name in the WordNet ontology. If an appropriate attribute name was found, it stores the name in the mapping ontology base for the next search. For example, some attributes like 'price' and 'manufacturer' can be directly found in the Amazon.com ontology. But the attribute 'monitor_size' of ODP can be matched to 'screen_size' of the Amazon.com ontology by using attribute name extension with synonyms and coordinate terms.

If the *Attribute Mapping* module cannot find a matching attribute name from a shopping mall's ontology, then it deletes the parts of the query concerning the attribute name from RDQL query. In our example, the attribute 'tv_type' was deleted because there was no matching attribute in Amazon.com.

There are additional steps of *Attribute Mapping* in order to solve the difference of data types and units. In the data type transformation step, we classified data type into two types - string type and numeric type. Next we converted each type to the opposite type if it is needed. For example, if the attribute 'screen_size' is the string type in Amazon.com and the corresponding part of the query in the customer's configuration is 'monitor_size', then we convert the type screen_size to numeric type and make a new query for Amazon.com.

When the unit of an attribute is different, we applied pre-built unit changing rules to change the unit of the attribute. For example, one shopping mall may use 'lbs' as the unit of weight though a customer uses 'kg'. In this case, the unit can be transformed to the other unit by using pre-built unit changing rules.

3.2.3 Query Conversion

As the last step of query generation, the *Query Generator* generates queries for each shopping mall after the completion of all mappings. Figure 9 shows the original query from the customer's configuration and the interpreted query for Amazon.com after the completion of *Product Mapping* and *Attribute Mapping*. The product name was converted to 'TV & HDTV' from 'television' and the attribute 'monitor_size' was changed to 'screen_size'. The attribute 'tv_type' and its value 'LCD' are deleted from the query because there was no matching attribute name to 'tv_type' in the Amazon.com ontology.

<pre> SELECT ?x WHERE (?x, rdf:type, dmoz:television) (?x, dmoz:manufacturer, "samsung") (?x, dmoz:monitor_size, ?size) (?x, dmoz:price, ?price) (?x, dmoz:tv_type, "LCD") AND ?size > 15 && ?price < 600 USING dmoz FOR <http://www.dmoz.org#> </pre>		<pre> SELECT ?x WHERE (?x, rdf:type, amazon:TV&HDTA) (?x, amazon:manufacturer, "samsung") (?x, amazon:screen_size, ?screen_size) (?x, amazon:price, ?price) AND ?screen_size > 15 && ?price < 600 USING amazon FOR <http://www.amazon.com#> </pre>

Fig. 9. An Example of Query Conversion

After generating queries for each shopping mall, the *Query Generator* sends modified queries to corresponding malls.

3.3 Query Execution in Shopping Malls

Query Execution is performed in Web services based shopping malls that receive queries from the *Query Generator*. The search procedure of *Query Execution* depends on each shopping mall's policy and system. For example, some shopping malls may use a keyword-based search to answer the query, and other shopping malls may convert the RDQL query to a database query and get the answer from a database. As one of the alternatives for search engines, we constructed an inference-based product search engine that performs a search by using the product category and attribute ontology, a rule base, and product data base. We used Jena API [7] to implement the inference based product search engine.

The objective of the inference-based product search engine is to perform a query execution based on various attributes like a data base query execution. To achieve this objective, we translated the content of the product data base into an RDF file and executed an RDQL query on the translated RDF file. But we cannot perform accurate query execution based on attributes with direct translation of the data base. For example, if required information is separately stored in two tables instead of one table, we should integrate the information of two tables by joining them. In order to join the two tables, we need rules about the relation between the two tables like the foreign key constraint in DBMS. The rule base stores such rules that are required to generate an RDF file on which an RDQL query is executable. Those rules define the relationship between product categories and attributes. Figure 10 shows the architecture of the inference based product search engine. The *Rule Reasoner* translates the contents of the product data base into an RDF file by using the ontology base, rule base, and the product data base. The *Query Reasoner* performs the given RDQL query on the RDF document and other ontology bases, and completes the search on various attributes. The search result is returned to the IPIS system through Web services.

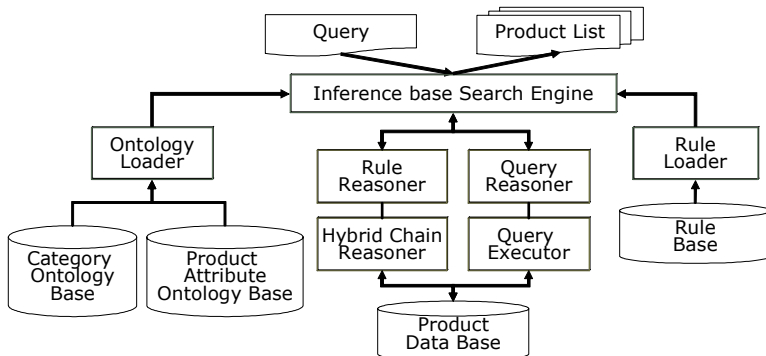


Fig. 10. Architecture of Inference Based Product Search Engine

3.4 Evaluation of Products

The *Products Evaluation* procedure consists of a product information analysis and similarity calculation. In the product information analysis step, the module receives the product information through Web services from shopping malls. Then it analyzes

the product information and extracts attribute names and values of each product. The module uses automatically generated RDQL queries to extract product information because the messages through Web services are represented with OWL. As the last step of the product information analysis, we calculated the distances of every attribute between the customer's configuration and the tracked product. The distance of each attribute is calculated in consideration of the attribute name, value, and criteria. The algorithm calculates the distance for attribute name and value by using the difference of strings and numbers. For criteria, the distance is 0 if the value satisfies a criterion of the attribute, 1 if not. When a unit is different, the module changes the unit with translation rules to calculate the distance. The distance of each attribute becomes normalized between 0 and 1.

In the next step, the similarity calculation module integrates every distance of the attributes to calculate the similarity of each product. We used a general similarity calculation method to evaluate products. The customer's priorities for each attribute are changed to attribute weights. We calculated the weighted sum of distances with the distances and weights of every attribute. Finally, we ranked products by their similarities to the customer's configuration. After the completion of *Product Evaluation*, it prints an ordered product list for the customer, as illustrated in Figure 11.

4 Implementation and Results

We developed an IPIS prototype which consists of an *Interface*, *Query Generator*, and *Product Evaluation Agent*. We also constructed two prototypes of intelligent shopping malls using Web services by implementing an inference based search engine and ontology with the information from Amazon.com and Buy.com. Therefore, users can search for and compare products of Amazon.com and Buy.com by using our prototype.

4.1 Implementation

Figure 11 shows the example of product search results and the functions of each small window in the IPIS system which is implemented using Java. In Figure 11, part number 1 is the window where a customer can choose a product category from a tree formed hierarchy. The hierarchy for the selected category appears in part number 2. The customer can input product attribute names, values, priorities, types and units in part number 3. The attributes that the customer can select are determined by the selected product category. For example, if a customer selects 'television' as product category, the attributes that the customer can select for 'television' are 'manufacturer', 'price', 'monitor_size', etc. Part number 4 shows a product list with the rank that is found with the configuration in part 3 from Amazon.com and Buy.com.

If a customer wants to add new criteria on the attribute 'monitor_size' to the configuration, the customer can pop up an edit window by clicking the 'Edit' button in part 3. Figure 12 shows the edit window to set an attribute's name, value, type and priority that represent the criterion 'monitor_size >= 15 (inch)'. The customer can assign priority with a number between 1 and 9.

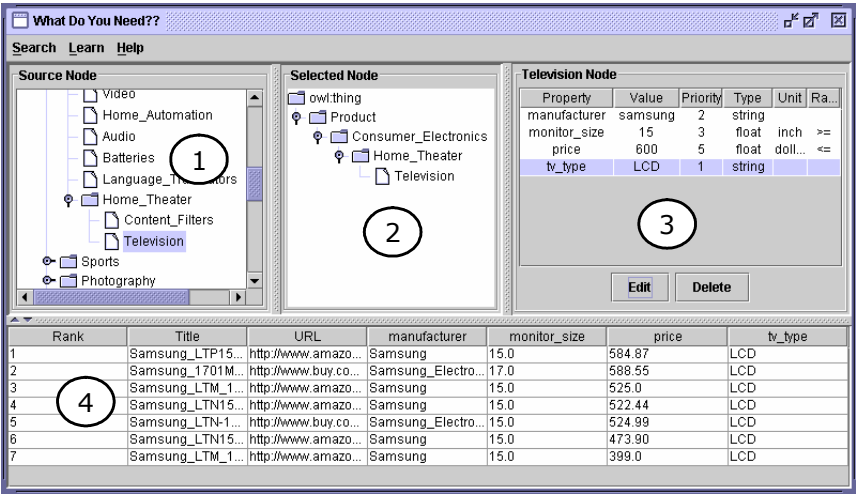


Fig. 11. Product Search Results in the IPIS System

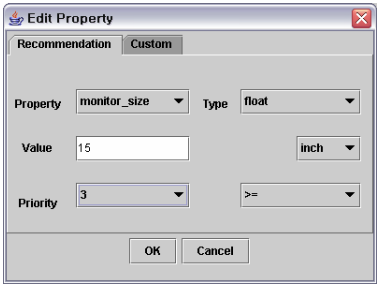


Fig. 12. An Edit Window for Attribute Input

4.2 An Example Comparing the IPIS System and the Keyword-Based Search

Table 1 shows an example that compares search results from the IPIS system and keyword-based search results from Amazon.com. The target product of the search is ‘television’ whose ‘manufacturer’ is ‘Samsung’, ‘tv_type’ is ‘LCD’, price is less than \$600, and ‘monitor_size’ is larger than 15 inches. From the experiment, we were able to find five products satisfying all criteria from Amazon.com with the IPIS system.

In the keyword-based search of Amazon.com, we made keywords as ‘television Samsung LCD 600 15’ to represent the target product. However there was no retrieved product when we included the desired price in the keywords as noted in the second and third row of Table 1. Therefore we excluded the price value and the monitor type value from the keywords, and we could get 19 products. In order to examine the accuracy of the keyword-based search, we manually checked each product to see if the product satisfied the customer’s configuration. We found 4 satisfactory products among them. When we added monitor size shown as the fifth row of Table 1, we got

3 satisfactory products among 4 products in total. In this example, the IPIS system was more accurate than the keyword-based search. Therefore we expect that customers using the IPIS system can save the time required for various keyword-based search trials to acquire accurate results.

Table 1. Product Search Results in the IPIS system and Amazon.com

Result of IPIS	Keyword Based Search Results of Amazon.com		
# of products	Keywords	# of products	# of products satisfying criteria
5	television samsung LCD 600 15	0	0
	television samsung LCD 600	0	0
	television samsung LCD	19	4
	television samsung LCD 15	4	3

4.3 Discussion

One of limitations in our system is that we assumed every shopping mall uses Web services. In a commercialized system, the IPIS system should support traditional shopping malls which are based on HTTP. Also, most shopping malls do not provide their own ontologies for product category names and attribute names. Actually we constructed each shopping mall's ontology by collecting required product information from their Web pages.

In addition, we did not prove that our IPIS system was more accurate and efficient than the traditional keyword-based meta-search engine because the experiment was limited. We just tried to show an example of a better case. We are planning to conduct a more precise experiment to prove the advantages of the IPIS system.

5 Conclusion

We have designed and implemented an intelligent product information search framework using ontology mapping and Web services which has a taxonomy free architecture for meta-search and comparison. Although shopping malls of our systems have the same inference system, our architecture can be easily extended to heterogeneous systems by using the Semantic Web and Web services. We expect that our system will show the opportunities and challenges of the Semantic Web and Web services in electronic commerce.

Web services can be actively utilized in the area where information should be exchanged repeatedly and periodically such as in B2B exchange. However the difference in ontologies can be an obstacle of information exchanges. Ontology mapping is one of the techniques that solve the semantic difference. We expect that the integration of Web services and ontology mapping will be a powerful solution for automatic information exchanges between software programs and agents.

Acknowledgements. This work has been funded by the University Fundamental Research Program of the Ministry of Information & Communication in Korea

References

1. Ackerman, M., Billsus, D., Gaffney, S., Hettich, S., Khoo, G., Kim, D.J.: Learning Probabilistic User Profiles. *AI Magazine*, Vol. 18. No. 2 (1997) 47-56.
2. Aridor, Y., Carmel, D., Lempel, R., Soffer, A., Maarek, Y. S.: Knowledge Agents on the Web. *Lecture Notes in Computer Science*, No. 1860 (2000) 15-26.
3. Benetti, H., Beneventano, D., Bergamaschi, S., Guerra, F., Vincini, M.: An Information Integration Framework for E-Commerce. *IEEE Intelligent Systems*, Vol. 17. No. 1 (2002) 18-25.
4. Chen, Z., Meng, X., Zhu, B., Fowler, R.H.: WebSail: From On-line Learning to Web Search. *Knowledge and Information Systems*, Vol. 4. No. 2 (2002) 219-227.
5. Ehrig, M., Sure, Y.: Ontology Mapping - An Integrated Approach. *Lecture Notes in Computer Science*, No. 3053 (2004) 76-91.
6. Howe, A. E., Dreilinger, D.: Savvy Search: A Metasearch Engine that Learns which Search Engines to Query. *AI Magazine*, vol. 18. no. 2 (1997) 19-25.
7. Jena 2 Inference Support. <<http://jena.sourceforge.net/inference/index.html>>.
8. Kalfoglou, Y., Schorelmmmer, M.: Ontology mapping: the state of the art. *The Knowledge engineering review*, Vol.18. No.1 (2003) 1-32.
9. Lawrence, S., Giles, C.L.: Accessibility of Information on the Web. *Nature*, Vol. 400 (1999)107-109.
10. Lawrence, S., Giles, C.L.: Context and Page Analysis for Improved Web Search. *IEEE Internet Computing*, Vol. 2. No. 4 (1998) 38-46.
11. Miller, G. A., "WordNet a Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, 1995, pp. 39-41.
12. Noy, N.F., Musen, M.A.: The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, Vol. 59. No. 6 (2003) 983-1024.
13. O'Keefe, R. M., McEachern, T.: Web Based Customer Decision Support Systems. *Communications of the ACM*, Vol. 41. (1998) 71-78.
14. Open Directory Project. <<http://www.dmoz.com>>.
15. Seaborne, A.: RDQL - A Query Language for RDF, W3C Member Submission. (<http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/>) (2004).
16. Selberg, E., Etzioni, O.: The MetaCrawler Architecture for Resource Aggregation on the Web. *IEEE Expert*, Vol. 12. No. 1 (1997) 11-14.
17. Smith, M.K., Welty, C., McGuinness, D.: OWL Web Ontology Language Guide, W3C Recommendation. <<http://www.w3.org/TR/owl-guide/>> (2004).
18. Veltman, K. H.: Syntactic and Semantic Interoperability: New Approaches to Knowledge and the Semantic Web. *New Review of Information Networking*, Vol. 7 (2001) 159-184.

Multiple Vehicles for a Semantic Navigation Across Hyper-environments

Irene Celino and Emanuele Della Valle

CEFRIEL – Politecnico of Milano,
Via Fucini 2, 20133 Milano, Italy
celino@cefriel.it, dellavalle@cefriel.it

Abstract. The Web, but also for example a large extra-net such as a digital library, has an intricate topology that makes navigation through resources a tricky task. The mere introduction of Semantic Web technologies won't automatically solve this task, because the Semantic Web only promotes machine understandability of Web resources by explicitly providing a thick bunch of annotations, thus leaving their interpretation and use to the application. In this paper, we refer to such difficult-to-navigate open information systems as *hyper-environments* and we compare the interaction with a hyper-environment to a *journey* in which users are travelers and the aim of the journey is to find useful information.

We therefore propose a Semantic Navigation Engine that aims at helping Web travelers in traversing hyper-environments by giving them proper tools (called *vehicles* in the travel metaphor) that can help them in getting oriented and fulfilling their aims.

1 Introduction

The existing Web, the so-called *syntactic Web*, is a boundless environment in which Web users navigate, searching for useful information to achieve their goals.

Navigating in the syntactic Web, following links to move from a Web resource to another, has been compared (see [1], [2]) to the concept of *travel* in the physical world: a Web user acts like a traveler who tries to get oriented in an unfamiliar environment and often has to re-start and take different paths before finding the right way to the desired destination.

Adding a bit of semantics to the syntactic Web provides machines with a bunch of annotations, but we believe that this does not automatically imply that navigating the *Semantic Web* is any simpler.

In this paper, we name *hyper-environment*, in accordance with the travel metaphor, any open information system in which resources are described in a machine processable way and we introduce the concept of *vehicle* as the necessary tool to navigate effortlessly across a hyper-environment and to follow the most opportune path to reach the needed information.

We therefore propose a Semantic Navigation Engine which, building upon semantic annotations attached to resources, is able to provide users with proper

vehicles, which support each part of their journey and suggest them the most suitable path to fulfil their specific tasks. In a few words, our Semantic Navigation Engine aims at providing end-users with *multiple vehicles for traveling across hyper-environments*.

In particular, for demonstrative purposes, we focus our attention on the medical general practice: if we apply the travel metaphor to this field, travelers are General Practitioners, the hyper-environment in which they move is a result set of a search on distributed and heterogeneous healthcare repositories, and their aim is to deepen their personal medical education in order to better manage their patient visits. In this paper we show how our Semantic Navigation Engine is able to provide General Practitioners with a customized vehicle that makes their travel across resources easier and more effective.

This paper is structured as follows: in section 2 we present the navigation problem and in section 3 the state-of-the-art approaches to solve it, both in the syntactic and in the Semantic Web fields; the concept of multiple vehicles to travel across hyper-environments is introduced in section 4, while the implementation of our Semantic Navigation Engine and our test-beds are described in sections 5 and 6 respectively.

2 Getting Lost in Hyper-environments

The fact that traveling across hyper-environments is possible does not mean that navigation is also easy or obvious to be undertaken. Below, we present the problem setting together with a possible usage scenario in the medical general practice.

2.1 The Problem

In the early days of the Web the lack of navigation plainness was considered as the *navigation problem*: users can get lost in a hyperspace and this means that, when users follow a sequence of links, they tend to become disoriented in terms of the goal of their original query and in terms of the relevance to their query of the information they are currently browsing [3]. The navigation problem has been long investigated in the hypermedia community and in particular some proposals tried to use the physical world as a model (e.g. Dillon in [2], Darken in [1]).

Following the physical world's metaphor, we notice that getting lost in a hypertext can be compared to getting lost in an unstructured space like a wood. In a structured space, for example when you get lost in a city, you can get oriented because streets have names and you just need to figure out your position on the two-dimensional structure of the city's map. On the contrary, when you get lost in a wood, you don't know where you are, how to reach your destination, and, often, you don't even know how to return to your original place. In a hypertext, like in a wood, you have to remember many trails and the way in which they are interconnected because most of them look just the same. As Nielsen writes in

his famous book “Designing Web Usability: The Practice of Simplicity” [3], the navigation problem is still *the* unresolved problem in Web site usability.

The current effort to add semantics to the Web suggests augmenting user-intended resources with machine-readable information, by means of metadata defined by ontologies. Considering also that all this additional information is provided with re-use in mind (thus authors are invited to put as much information as possible), the topology of the resulting hyper-text is much more complex than the already intricate topology of the Web. This is the reason why we name it a *hyper-environment*.

We believe that addressing the navigation problem in a hyper-environment is challenging but feasible, because semantic annotations provide machines with the ability to access what readers normally consider shared contextual information together with the information which is hidden in the resource.

2.2 A Usage Scenario in the Medical General Practice

General Practitioners are end-users that cannot afford to get lost. They are the category of physicians which is more exposed to medical errors. If they were well-informed about each novelty concerning diagnostics or treatments in the healthcare sector, they could considerably reduce their patients’ risks. But being up-to-date is often a hard task, and probably it’s impossible for General Practitioners to be informed about every single human pathology, so they generally prefer to prescribe an additional examination or to refer the patient to a specialist for a visit. They would spend time to deepen their medical knowledge only if they could be sure to reach easily and effortlessly the most suitable information, for example if they could come directly to the appropriate clinical guideline for a given pathology and access information regarding the available medical services, technologies and medications, their efficiency and side effects, possibly even relevant case studies or some specialist advice.

The following usage scenario describes the interaction between the end user (i.e. the General Practitioner) and the Semantic-based Healthcare Information Portal (named SHIP) we are conceiving in the COCOON project¹, which provides a uniform, single point of access to heterogeneous and highly distributed medical information sources. This information may include articles from medical journals, scientific publications from specialized search engines and university libraries, electronically available clinical guidelines, as well as free text documents provided by each regional public health authority to local General Practitioners:

1. Geena, a General Practitioner, has come across an article in the British Medical Journal mentioning new breast cancer symptoms apparently discovered in the population of female smokers over the age of 40. Intrigued by the article, and having relevant population among her patients, Geena decides to use SHIP, in order to collect additional information regarding this topic which may be useful in her future practice.

¹ <http://www.cocoon-health.com>

2. Geena submits a query on early detection of breast cancer and an overwhelming amount of documents, together with their semantic annotations, is retrieved, but SHIP hides them behind a multifaceted display of the search results that includes a ranked search results list, relevant ontology concepts and documents associated with them. For each document SHIP provides the title, the publication date, the relevant keywords and a short excerpt.
3. Geena inspects the search results. She finds a document that she considers useful for her practice and follows the hyperlink to the original document text. SHIP offers her not only the document, but also hyperlinks to other documents thematically related to the selected one and the keywords used to semantically describe the document.
4. Geena follows the link to a keyword and a small bunch of related terminology is presented. She finds in this terminology a more specific term she did not think of when submitting the query, so she follows its hyperlink. SHIP provides her again with a small bunch of the terms and with links to the documents in the result set that are labeled with such keywords. This time Geena does not have to look for other documents, because she finds what she was looking for.

3 State of the Art and Trends

Over the last few years, many different research trends have tried to solve the navigation problem. We briefly summarize some of the most relevant ones, in order to frame the background of our solution.

3.1 Syntactic Web Approaches

Among the approaches which try to solve the navigation problem in the syntactic Web, an important position is occupied by trail engines. A *trail engine*² is a sort of search engine, but it differs from a standard one because, in reply to a query expressed as a conjunction of keywords, a search engine returns a ranked list of pages (possibly containing all the keywords), whereas a trail engine returns a graph of pages connected by links, normally named trail (so that the user is advised to follow a trail across a set of interconnected pages that represents a path via hyperlinks among the requested keywords).

The idea behind many trail engines is to model the Web as a network of nodes labeled with keywords. The Best-Trail Algorithm [4] models the Web via a finite automaton called Hypertext Finite Automaton (HFA). The states of a HFA represent Web pages, while transitions represent links. In this way, in a HFA, a valid trail leads the user to follow existing links from one node to the others in the trail. A slightly more complex way to model the Web was proposed in [5]. It consists in extending HFA by attaching probabilities to state transitions. These

² see e.g. <http://trails.navigationzone.com/>

probabilities can denote either the result of frequent user behavior patterns or the result of some calculation (e.g. average) over the relevancies of the pages in the trail.

In short, trail engines provide an affordable way to search for a trail in a set of already interconnected Web resources. In a way they try to cope with the lack of explicit semantics in the Web using models of the user behaviors that are supposed to partially capture what each page is about. Unfortunately, the known approaches don't seem to scale up.

3.2 WWW Conceptual Model Approaches

An alternative category of solutions to trail engines is provided by *World Wide Web Conceptual Modeling approaches*, or shortly the WWWCM approaches. They are characterized by the common goal of modeling a Web application at the conceptual level, in order to automatically implement it. Some successful attempts, such as WebML [6], W2000 [7], OOHDM [8], belong to this category. They show that a data-intensive Web application can be easily developed by separately modeling the domain information space, the navigation, the access and, in recent attempts, also the operations.

In the attempt to sketch out the least common denominator among the cited WWWCM approaches, we formulate the following definitions we will refer to in the rest of the paper:

- The *domain information model* describes the organization of the information managed by the Web application, in terms of the pieces of content that constitute its information base and of their semantic relationships. The schema of this model provides a shared understanding of the Web content that does not change or only changes very slowly over the time.
- The *navigation models* concern the facilities for moving across the application content; they represent the heterogeneous inter and intra-object navigation facilities the users can employ in traversing the information space of the Web application. These models are not necessarily shared among all users, but they are jointly employed by homogeneous categories of users.
- The *access models* concern the facilities for accessing information, i.e. the available access paths to objects in the information space. Access models specify the way in which the information described by the domain model is accessed: multiple access-models can be attached to the same domain information model, in order to specify different access semantics for different purposes. Each access model consists of collections of not strictly homogeneous objects.

Summing up, the WWWCM approaches are typically top-down, therefore they provide excellent solutions to manage the life cycle of a complex Web application, but this is mainly done on the strong assumption that all the information is under the control of the organization responsible for developing the Web application and that such application can be built from scratch. So, we might

make to the WWWCM approaches the criticism that they address, and probably solve, the navigation problem in a closed context, but they do not address the navigation problem on the open Web. In a way, they break the principle according to which the Web is to be a universal, hence open, system.

3.3 Semantic Web Approaches

The lack of explicit semantics in the Web was perceived as an addressable problem by the knowledge representation community in the late '90s. SHOE [9] and OntoBroker [10] are successful attempts to show that such added semantics could prove to be very useful in solving the navigation problem. The idea is to define the terminology top-down (through ontologies) and use it bottom-up to annotate and wrap Web documents.

We believe that, nowadays, the major trend in solving the navigation problem is represented by the approaches based on the Semantic Web, whose common principle can be named *ontology-supported and ontology-driven conceptual navigation*. According to this principle, resources and links should be considered separately. *Resources* are self-contained items whose content can even be difficult to process automatically. *Links* are machine-processable descriptions (known as metadata) of the resources. These descriptions are provided by the authors in accordance with the terms described in one or more ontologies. In this way, every resource lies in a context made of semantic descriptions of terms and other resources that a machine can access and process. So the common situation experienced in the Web, whereby a reader can get lost if the author's intention does not match with the reader's intention, might be less common in the Semantic Web, because authors provide reusable descriptions while machines, being able to manage these descriptions, can adapt the interface to meet the readers' intentions. Therefore, complex role-based and integrated navigation structures can be built bottom-up on the fly, as long as each resource is described by the terms defined by shared ontologies, provided top-down.

Below, we provide a short description of some of the leading efforts in the Semantic Web approach:

- *COHSE* - a Conceptual Open Hypermedia Service [11] is an ontological reasoning service and Web-based open hypermedia link service integrated to form a conceptual hypermedia system, to enable documents to be linked via metadata describing their contents;
- *SEAL* (SEmantic portAL) and *SEAL-II* [12] show how ontologies can power information retrieval, greatly contributing to the combined goals of low-effort information integration and user-friendly information presentation;
- *OntoWebber* [13] supports the creation of reusable specifications of Web sites, by explicitly modeling the Web site via ontologies and employing semi-structured data technology for data integration;
- *ODESeW* [14] is a rapid development tool for building ontology-based Web portals, which allows to configure the visualization of ontology-based information for different kinds of users;

- *SOIP-F* [15] describes a framework for developing Organization Information Portals that deal with a small-scaled organizational Semantic Web, where resources are augmented with semantic annotations.
- *OntoViews* [16] is a Semantic Web portal tool for publishing RDF content on the Web; it combines the multi-facet search paradigm, developed within the information retrieval research community, with Semantic Web RDFS ontologies and extends the search service with a Semantic browsing facility based on ontological reasoning.

4 Our Concept: Multiple Vehicles for Traveling Across Hyper-environments

In this section we introduce our approach to the navigation problem, using a vivid metaphor to explain how users move across hyper-environments and how we can support and facilitate their navigation.

4.1 The Travel Metaphor

We believe that the physical world metaphor eases the problem of conceiving an open solution to the navigation problem, because users in traversing the hyperspace need the sort of information which is normally required to traverse the physical space. According to [17], users require the following information:

- *Orientation information*, necessary to find one's place within a body of interlinked resources. In designing a hypertext, one should therefore give an appropriate answer to the question "What can be done to orient users and help them to navigate efficiently and pleasantly?"
- *Navigation information*, necessary to make one's way through resources. In designing a hypertext, one should therefore give an appropriate answer to the question "While accessing a particular resource, how can users be informed about where the links related to that resource lead?"
- *Exit or departure information*, necessary to inform the users that they are leaving a given context. In designing a hypertext, one should therefore give an appropriate answer to the question "How can users retrace their steps in their going-on path?"
- *Arrival or entrance information*, necessary to inform the users that they are entering a given context. In designing a hypertext, one should therefore give an appropriate answer to the question "While accessing a new resource, how can users be assisted to feel "at home" in the new context?"

Goble et al. in [18] have moved further in the direction of introducing the notion of travel and mobility on the Web, to improve the accessibility of using the physical world as a model. They define *travel* as the *confident navigation and orientation with purpose, ease and accuracy within an environment*. This means that they extend navigation to include orientation, environment, mobility and purpose of the journey:

- *Orientation* is the knowledge of the basic relationships between objects within the environment, and between the objects and the traveler.
- *Environment* is the context which the traveler traverses and includes the way in which the landscape is rendered and perceived.
- *Mobility* is the ease and confidence at which travel can be accomplished.
- *The purpose of the journey* is the reason why the traveler has chosen to undertake the journey.

Following this trend, Yesilada, Stevens and Goble introduce in [19] the concept of *travel objects* in order to describe how the hypertext environment is rendered and perceived by the travelers. In fact, travelers use or may need to use such environmental features or elements in order to make a successful journey, meaning that, when following a trail of information, they need to keep a sense of direction and they need a high mobility in terms of the goal of their original query and of the relevance to the query of the information they are currently browsing.

4.2 The Vehicle Metaphor

When we browse the Web, we are already accustomed to following links to move from one page to another. But, simply carrying out the action of clicking on a hyperlink, not only do we go ahead in our travel across Web resources, but we also make a decision about the direction of our trail in order to reach the most relevant information we are looking for. Furthermore, in different navigations, we follow different paths, either because we are looking for different information or because our task is more specific or more generic, so the granularity of the information we require is narrower or broader.

Following the travel metaphor, in this paper we introduce the idea that Web users need different vehicles to travel across resources on different occasions. A *vehicle* must support user navigation, suggesting the most relevant trail among all the possible paths a user can undertake, according to the purpose of the journey itself. A vehicle must provide valuable information about the environment, enabling orientation and supporting mobility in order to reach easily and effortlessly the travel destination. Moreover, in order to make his/her journey useful, a user needs the most appropriate vehicle to travel across the environment to achieve the specific purpose of his/her journey. Thus, he/she will request different vehicles that satisfy different needs not only during different travels, but also within the same journey: for example, when a Web traveler enters a hyperenvironment for the first time, he needs at the beginning a vehicle that helps him to get oriented and understand the spatial relationships among the resources; afterwards, when he/she feels familiar within the new environment, the same traveler needs another vehicle to move on in his/her journey, to deepen his/her knowledge about a particular section of that hyperspace, looking for more detailed information.

What users need in their navigation is the most appropriate view on resources, meaning that, in every step of their travel, they don't need all the available information, but only a part of it, the part related to the information they are looking for.

Thus, when conceiving the vehicle metaphor, we asked ourselves what kind of view on resources a vehicle must provide the user with. To answer this question in the most precise and effective way, we noticed that in our everyday experience of navigation through the Web, we can look at the term *view* under, at least, two different aspects; we can therefore give two different meanings to a view on a particular resource:

1. View as *presentation* of a subset of all the available information regarding the resource; if we divide all the knowledge about a particular item in *travel objects*, i.e. atomic bunches of information, we can build a view by composing together these elementary “bricks”; in this meaning, two views on the same resource differ in the set of travel objects employed in giving information about a resource (a generic description instead of a detailed presentation, a certain set of aspects or features instead of another, and so on).
2. View as *navigation* from that resource to another one following hyperlinks; the view can suggest different paths to cross information, which can be related to the particular phase of the travel; in this meaning, two views on the same resource differ in the possible directions they suggest the user for the continuation of his/her journey.

5 Our Implementation

Our work tries to solve the navigation problem building vehicles to support the users’ journey across resources and building upon semantics attached to resources in order to make traveling significant and effective to attain the users’ tasks.

Starting from an early prototype we already described in [15], we refine the browsing-time support of SOIP-F, a framework that supports the building of Semantic Organizational Information Portals, introducing the Semantic Navigation Engine by adding a presentation model, including a way to handle travel objects and a simple way to describe a model in term of navigation, access and presentation models. As a result we obtain the more powerful version of SOIP-F we describe below.

5.1 A Technical Overview of SOIP-F

At first glance, SOIP-F might appear as a radical new departure in Web portal design, but it is not. SOIP-F is implemented bringing together existing and well understood technologies:

- it uses a *Web Framework* that implements the well-known Model-View-Controller (MVC) design pattern,
- it follows the *WWW Conceptual Model approach* in separately modeling domain information space, navigation, access and presentation,
- it requires portal administrators to specify conceptual models using *ontologies* written in OWL-DL,

- it manages and stores RDF encoded *metadata* that describe the resources in a machine processable way.

An important requirement taken in consideration during the design of SOIP-F is the strong decoupling between the portal and the information sources, because the aim of an organization portal is to provide a single and user-tailored point of access to all organizational content sources.

SOIP-F, in fact, takes from the WWWCM the idea of separately modeling domain information space, navigation, access and presentation. Following such an approach in modeling portals independently of the domain, SOIP-F proposes a *portal ontology* that includes portal-dependent terminology: structural terms such as **entity** or **component**, navigation terms such as **contains** or **related_to**, access terms such as **next** or **down** and presentation terms like **title**, **text-box** or **image**.

This strong requirement for decoupling between the portal and the information sources expects a portal built using SOIP-F to be just one of the many applications accessing content sources; for this reason, SOIP-F doesn't require the information to be structured in any particular way. So, differently from most WWWCM approaches, SOIP-F proposes to model navigation, access and presentation by *mapping* the domain terminology into the portal terminology, creating, in a bottom-up approach, a relation between domain-dependent terms and portal-dependent terminology.

In particular, as we anticipated, modeling a vehicle for our Semantic Navigation Engine in order to travel across a portal built using SOIP-F implies the operation of mapping, through the composition of navigation, access and presentation models. We explain the meaning of these models in SOIP-F as follows:

- The *navigation model* takes into account the possible paths across homogeneous resources (for example, it states which relationships must be underlined and emphasized); it can be shared among many users with similar aims.
- The *access model* takes into account the possible paths across heterogeneous resources which share some meaning (for example an access model can suggest an ordered list of resources to be navigated serially or a set of “most visited” or “recently added” pages); it is specific of the aim but it can be built by querying the domain information model.
- The *presentation model* takes into account the composition of different parts of information on the page, the order and the layout of the presentation (for example, a presentation model states number, type and position of the “boxes” that set up a page); it can include both shared travel objects (that are provided for general purposes) and user specific travel objects (that are provided to support the user in a specific part of his/her travel).

5.2 A Structural Overview of SOIP-F

Our Semantic Navigation Engine builds upon the fact that SOIP-F is an extendable J2EE application framework that can be configured using a set of OWL ontologies.

The framework itself is not a portal, because it needs at least a domain ontology and a set of “content sources” to be crawled. Once this information is provided, the content sources become browsable using a low-level vehicle we name “metadata-vehicle”. Starting from a resource, this vehicle shows in a table all the RDF triples that involve the selected resource. Each subject, property and object is provided as a link that the end-user can follow in order to select another resource.

In the same way, other vehicle descriptions can be provided. Each vehicle description includes a navigation model, an access model and a presentation model. In particular, the presentation model describes which travel objects are displayed for each type of resource. Each travel object is either responsible for presenting the information carried by some metadata or for providing a link the user can follow to move to other parts of the hyper-environment.

Each travel object is made up of two parts: a decorator and a template. A decorator is a Java class that is responsible for querying the reasoner and for extracting the information that the travel object will show, whereas a template describes the visual appearance of the information provided by the decorator.

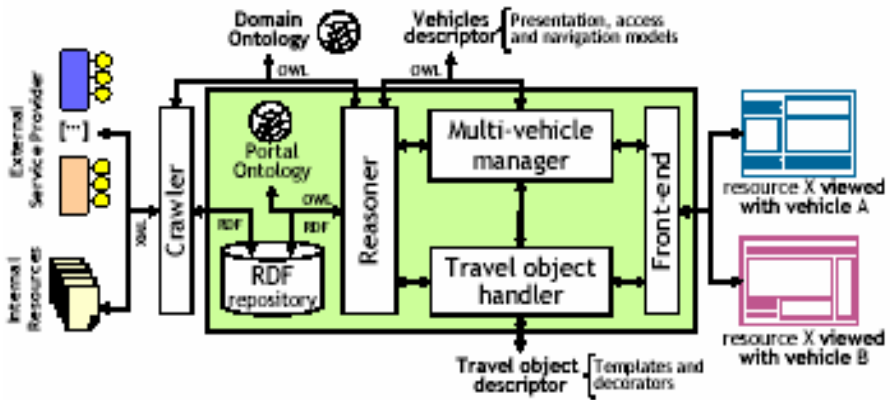


Fig. 1. SOIP-F structural overview

SOIP-F (see figure 1) is composed of five logical components:

- the *reasoner*, which manages both the terminologies and the assertions (except for literals). If no configuration information is provided, it only contains the portal ontology. It also offers query facilities for the other components. It is based on RACER³ and extends JRacer APIs;

³ <http://www.sts.tu-harburg.de/~r.f.moeller/racer/>

- the *RDF repository*, which stores the metadata describing the resources; it offers the reasoner query facilities for selecting literals and it offers the crawler query facilities for inserting, deleting and updating metadata. It is based on Jena framework⁴ and can use either the file system or a database as permanent storage facility.
- the *multi-vehicle manager*, which enables end-users to select a vehicle and/or to change it during the navigation. It uses the facilities provided by the reasoner in order to mount the most appropriate navigation, access and presentation model.
- the *travel object handler*, which uses, according to the current vehicle selected, the appropriate decorators to query the reasoner and the right templates to produce a HTML-based presentation of each travel object.
- the *front-end*, which is made up of several Java Servlets that share a Velocity template engine⁵. Together they implement a MVC design pattern in a J2EE environment.

These components can be extended with another one: during the building of a portal with SOIP-F, a crawler can be developed to extract data, properly annotated with metadata described by one or more domain-specific ontologies, from content sources and to make them available to be presented and navigated within the portal.

6 Our Test-Beds

To prove our approach we built some test-portals on the top of our SOIP framework (see <http://seip.cefriel.it>). We briefly propose the most significant one, which (tries to) solve the navigation problem in the healthcare domain, and then we present other test-implementations demonstrating our approach.

6.1 Aiding General Practitioners in Navigating a Healthcare Hyper-environment

As introduced in §2.2, in the COCOON project we have to satisfy the requirements of a General Practitioner who would like to be informed and up-to-date about every medical information that could be useful to better manage his/her visits, in order to provide the most suitable treatment or to prescribe the most useful examination to his/her patients.

A Semantic-based Healthcare Information Portal (SHIP, as abbreviated before) could be very useful during the general practitioner's search for information, supporting his/her queries, returning the most interesting results and suggesting different significant paths to navigate across the result set.

We built a prototype of SHIP on the top of our SOIP-framework, designing a portal to help general practitioners to travel, easily and effortlessly, in a

⁴ <http://jena.sourceforge.net/>

⁵ <http://jakarta.apache.org/velocity/>

healthcare hyper-environment. We based our search facilities on the Web services provided by Entrez PubMed⁶, called e-utilities, that allow to search in PubMed, MeSH and other medical databases. We developed an ad-hoc component, an invoker that, querying Entrez e-utilities through pre-formatted queries (called search strategies), is able to retrieve information about medical articles; Entrez services return results in XML-format that our component translates, through an XSL transformation, to RDF-format described by a simple OWL ontology.

The result set is therefore made up of a set of resources described with meta-data: each article presents a bunch of information about the document itself (such as authors, title, medical journal, abstract, the link to the actual document, etc.) together with cues about its *semantics*, i.e. the medical keywords that describe the meaning of the article. This terminology comes from MeSH⁷ (Medical Subject Headings), a thesaurus of medical terms that are semantically interconnected (for example, a term is linked to a broader or narrower one, or a term about a pathology is linked to the term of the affected body-part); these terms can be exploited to put in relation articles sharing the same *semantics*.

Once the result set is available to be crawled and accessed via the portal, our Semantic Navigation engine presents the results to the final user, giving him/her a vehicle to travel across them. This vehicle is able to suggest more than one path to navigate through resources: besides the path that offers access to the results as in a ranked ordered list and that lets the user navigate from an article to the subsequent one just following a “next article” link, SHIP offers the possibility of navigating through resources following the shared keywords. While in the first case the General Practitioner just follows a list (even if the order of the retrieved documents can be altered to take into account the user’s preferences and interests), in the latter case the final user can navigate through keywords as well as through documents and can exploit this information to better understand the retrieved results and, if necessary, to refine his/her search strategy using the most appropriate keywords to re-query Entrez databases.

Readers wishing to try SHIP can do so on <http://seip.cefriel.it/ship>.

6.2 Other Demonstrative Portals

Some other test portals that we built to demonstrate our approach are available on line. These demonstrative portals illustrate the possibilities and potentialities of our Semantic navigation Engine in SOIP-framework and we introduce them briefly.

Virtual Museum of Contemporary Art portal – it aggregates data (in Italian) about artworks and artists from different real museums, letting virtual visitors

⁶ <http://www.ncbi.nlm.nih.gov/Entrez/index.html>

⁷ MeSH is a thesaurus developed by US National Library of Medicine since 1954; further information can be obtained in <http://www.nlm.nih.gov/mesh/meshhome.html>

travel across resources with different vehicles: the portal offers a thematic trail vehicle (a guided tour across artworks of a particular artistic movement or period), but also a detailed trail vehicle (that allows to investigate the work of a particular artist).

Semantic Web Virtual Lesson – it’s a portal built as a unifying view on different material taken from some presentations about fundamentals, technologies and applications of Semantic Web; it allows end-users to move from a presentation to another just following links to semantically related slides, as if it was a single lesson.

CEFRIEL organization portal – CEFRIEL’s information about units, people, projects, research fields is enriched with semantic annotations that allow the Semantic Navigation Engine to present resources to different users in different ways: there are various views centered on units, on ongoing projects or on research fields.

7 Conclusions

Our proposal for a Semantic Navigation engine is a joint attempt to solve the navigation problem bringing together Semantic Web technologies, the WWCM approach and various studies on users’ habits and needs in browsing the Web. We identify the core problem in navigating through a hyper-environment with the difficulty of moving across different resources, maintaining a good sense of orientation and reaching the desired destination while covering a path that makes the user/traveler enrich his/her knowledge and attain his/her aim.

We believe that the key innovations introduced by our proposal for a Semantic Navigation engine are the following:

- loose coupling between domain information model (captured by the organizational ontology) and the various navigation, access and presentation models;
- bringing in the Semantic Web community the efforts of the WWCM approach in defining a (top-down) terminology for navigation, access and presentation;
- building navigation, access and presentation models in an indirect way, by mapping domain information terminology to navigation, access and presentation terminology (as in a bottom-up approach).

To achieve this goal, our Semantic Navigation Engine builds upon the definition of vehicle as a composition of navigation, access and presentation models; for each Semantic Navigation Engine, we can define a set of vehicles. In fact, different vehicles might be useful in different parts of the hyper-environment, in the same way as real vehicles (bicycles, cars, trains, planes) are used to take different kinds of journeys in the real world. In this way, we make a step forward in the direction of uncoupling domain knowledge from the way to access it.

Acknowledgments

The research has been partially supported by the COCOON Integrated Project (IST FP6-507126) while the implementation of SOIP-F has been partially funded by Engineering as part of the activities of the XVI Master in Information Technology of CEFRIEL – Politecnico of Milano. We thank Maurizio Brioschi, Stefano Ceri and Nahum Korda for their precious contributions.

References

1. R.P. Darken, J.L. Sibert: Wayfinding Strategies and Behaviors in Large Virtual Worlds. In: Proc ACM CHI'96. (1996) 142–149
2. A. Dillon, M.W. Vaughan: “It’s the Journey & the Destination”: shape and the emergent property of genre in evaluating digital documents. *The New Review of Hypermedia and Multimedia* **3** (1997) 91–106
3. J. Nielsen: *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Thousand Oaks, CA, USA (2000)
4. R. Wheeldon, M. Levene: The Best Trail Algorithm for Assisted Navigation of Web Sites. In: WWW2003, Budapest, Hungary. (2003)
5. M. Levene, G. Loizou: A Probabilistic Approach to Navigation in Hypertext. *Information Sciences* **114** (1999) 165–186
6. S. Ceri, P. Fraternali, A. Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks (Amsterdam, Netherlands: 1999)* **33** (2000) 137–157
7. L. Baresi, F. Garzotto, P. Paolini, S. Valenti: HDM2000: The HDM Hypertext Design Model Revisited. Tech. report, Politecnico di Milano (Jan. 2000)
8. D. Schwabe, G. Rossi, S.D.J. Barbosa: Systematic hypermedia application design with OOHDM. In ACM, ed.: Hypertext '96, Washington, DC, March 16–20, 1996: the Seventh ACM Conference on Hypertext: Proceedings, New York, NY, USA, ACM Press (1996) 116–128
9. J. Heflin, J. Hendler, S. Luke: SHOE: A Blueprint for the Semantic Web. In: *Spinning the Semantic Web*. MIT Press, Cambridge, MA (2003)
10. D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, A. Witt: On2broker: Semantic-based access to information sources at the WWW. In: *WebNet (1)*. (1999) 366–371
11. L. Carr, W. Hall, S. Bechhofer, C.A. Goble: Conceptual linking: ontology-based open hypermedia. In: *World Wide Web*. (2001) 334–342
12. A. Hotho, A. Maedche, S. Staab, R. Studer: SEAL-II - The Soft Spot between Richly Structured Unstructured Knowledge. *Journal of Universal Computer Science* **7** (2001) 566–590
13. Y. Jin, S. Xu, S. Decker, G. Wiederhold: Managing Web Sites with OntoWebber. *Lecture Notes in Computer Science* **2287** (2002) 766
14. O. Corcho, A. Gomez-Perez, A. Lopez-Cima, V. Lopez-Garcia, M. Suarez-Figueroa: ODESeW. Automatic Generation of Knowledge Portals for Intranets and Extranets. In D. Fensel, ed.: *The Semantic Web, ISWC 2003, LNCS 2870*. (2003) 802–817
15. E. Della Valle, M. Brioschi: Toward a framework for Semantic Organizational Information Portal. In proceedings of first European Semantic Web Symposium, ESWS2004 (2004)

16. E. Makelä, E. Hyvönen, S. Saarela, K. Viljanen: OntoView – A Tool for Creating Semantic Web Portals. In: Proceedings of the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan (2004)
17. G.P. Landow: Hypertext 2.0: The Convergence of Contemporary Critical Theory and Technology. Johns Hopkins University Press (1997)
18. C.A. Goble, S. Harper, R. Stevens: The travails of visually impaired web travellers. In: UK Conference on Hypertext. (2000) 1–10
19. Y. Yesilada, R. Stevens, C. Goble: A Foundation for Tool Based Mobility Support for Visually Impaired Web Users. In: Proceedings of the Twelfth International World Wide Web Conference. (2003)

Activity Based Metadata for Semantic Desktop Search

Paul Alexandru Chirita, Rita Gavriloiu, Stefania Ghita,
Wolfgang Nejdl, and Raluca Paiu

L3S Research Center / University of Hanover,
Deutscher Pavillon, Expo Plaza 1, 30539 Hanover, Germany
{chirita, gavriloiu, ghita, nejdl, paiu}@l3s.de

Abstract. With increasing storage capacities on current PCs, searching the World Wide Web has ironically become more efficient than searching one's own personal computer. The recently introduced desktop search engines are a first step towards coping with this problem, but not yet a satisfying solution. The reason for that is that desktop search is actually quite different from its web counterpart. Documents on the desktop are not linked to each other in a way comparable to the web, which means that result ranking is poor or even inexistent, because algorithms like PageRank cannot be used for desktop search. On the other hand, desktop search could potentially profit from a lot of implicit and explicit semantic information available in emails, folder hierarchies, browser cache contexts and others. This paper investigates how to extract and store these activity based context information explicitly as RDF metadata and how to use them, as well as additional background information and ontologies, to enhance desktop search.

1 Introduction

The capacity of our hard-disk drives has increased tremendously over the past decade, and so has the number of files we usually store on our computer. It is no wonder that sometimes we cannot find a document any more, even when we know we saved it somewhere. Ironically, in quite a few of these cases nowadays, the document we are looking for can be found faster on the World Wide Web than on our personal computer.

Web search has become more efficient than PC search due to the boom of web search engines and due to powerful ranking algorithms like the PageRank algorithm introduced by Google [16]. The recent arrival of desktop search applications, which index all data on a PC, promises to increase search efficiency on the desktop. Still, these search applications are weaker than their web counterparts as they cannot rely on PageRank-like ranking mechanisms which have revolutionized web search. Unfortunately, they also fall short of utilizing desktop specific characteristics, especially context information. Some of these missed opportunities include:

- *Email context* is not utilized by the existing search algorithms, even though this clearly drops useful information. For example, one email might contain a question describing the object one is looking for, and another email in the same thread might include the answer to that question in the form of an attached document.

- Email attachments lose all contextual information as soon as they are stored on the PC, even though emails usually include additional information about their attachments, such as sender, subject, comments. We might discuss a paper with a colleague during a brainstorming session, and then afterwards send her the electronic version via email, together with a few helpful comments. After a while, our colleague might not remember details about the paper itself, but rather recall with whom she discussed it or which question was raised in the discussion and included as comment in the email. It would be helpful to find the stored paper not only based on its content, but also associatively based on that context¹.
- *Folder hierarchies* are barely utilized by the search algorithms, even though we might have spent considerable time to build sophisticated classification hierarchies for the documents we store. For example, pictures taken in Hanover are probably stored in a directory entitled "Germany", "Lower Saxony" or "Hanover", and it would be nice if we could utilize this information when we search for the pictures.
- *Browser caches* include all information about user's browsing behaviour, which are useful both for finding relevant results (for example, if we remember how to find the project's home page, but not the corresponding API specification), and for providing additional context for results. It would also be very useful if our search application not only returns one specific scientific paper we downloaded from the CiteSeer repository, but all the referenced and referring papers which we downloaded on that occasion as well.

As studies have shown that people tend to associate things to certain contexts [9], all this information should be utilized during search. So far, however, neither has this information been collected, nor have there been attempts to use it.

In this paper we discuss how to enhance and contextualize desktop search based on semantic metadata collected from different contexts available and activities performed on a personal computer. We explore three important contexts: electronic mail, folder hierarchies, and web cache. Analogously, other contexts might be exploited as well. We describe the semantics of these different contexts by appropriate ontologies and show how to extract and represent the corresponding context information as RDF metadata which can be used by a search application together with a full text index of our documents.

The next section gives an overview over existing approaches which try to exploit metadata in search algorithms, and classifies them according to how they use metadata to enhance search. Section 3 then shows how to describe contexts and their corresponding metadata by means of appropriate ontologies and association rules, and how to use these metadata in four different search scenarios where a simple full text index employed by current desktop search engines fails to find the information we are looking for. Finally, section 4 describes the architecture of our semantic desktop search environment, as well as our prototype.

¹ Desktop Search is in fact "a search into our past", and it should therefore exploit the associative functionality of the human memory.

2 Using Semantic Metadata in Search: A Classification

2.1 Using Metadata to Enrich Search Results

One of the most interesting semantic search efforts is probably being performed in the TAP project [8]. TAP builds upon the TAPache module, which provides a platform for publishing and consuming data from the Semantic Web. Its knowledge base is updated with the aid of the onTAP system, which includes 207 HTML page templates, being able to read and extract knowledge from 38 different high quality web sites. The key idea in TAP is that for specific searches, a lot of information is available in catalogs and backend databases, but not necessarily on Web pages crawled exhaustively by Google. The semantic search based results are independent of the results obtained via traditional information retrieval technologies and aim to augment them.

While searching for musicians and other well-known entities like cities, countries and others can draw upon the fact that a lot of information about them is available in backend databases, whose data sets can be joined based on the ID of that entity, the situation is different in the educational context, where topic classification is the most important characteristics of a page. This latter approach is used in our personal reader system [3], which finds additional pages related to the pages contained in a course, and again provides these as additional information to the core information presented.

2.2 Using Metadata to Connect and Visualize Information

In "The Social Semantic Desktop" [2], the authors envision that the next step towards communication is a desktop application based on the Semantic Web, which could draw connections between all the types of data people interchange. For example, an entry in an agenda would be correlated with the author of an article or to the context associated to an email. Altogether, the entire information existing in a social network would be connected to each desktop. Such a structure would then help people organize and find information, due to the enhancement brought by metadata into the system.

The Fenfire project [5] proposes a solution to interlink any kind of information on one's desktop. That might be the birthday with the person's name and the articles she wrote, or any other kind of information. The idea is to make the translation from the current file structure to a structure that allows people to organize their data closer to the reality and to their needs, in which making comments and annotations would be possible for any file.

Haystack [17] pursues similar goals as Fenfire. One important focus is on working with the information itself, not with the programs it is usually associated with. For example, only *one* application should be enough to see both a document, and the email address of the person who wrote it. Therefore, a user could build her own links to Semantic Web objects (practically any data), which could then be viewed as thumbnails, web pages, taxonomies, etc.

A third project building an information management environment for the desktop is Gnowsis [19]. The main idea behind applications in this environment is the use of a central information server which allows users to administer and directly access all the information on their computer (for example the author of a file, her email address,

etc.). Gnowsisis envisions appropriate ontologies at four levels. The first one is used on the server, as it needs custom formats for the internal operation data and for its configuration files. The second one is for each application and the data stored by it. For example, in Outlook Express the types of data that can be found are emails, contacts and appointments. On the third level we have public ontologies, created by others to describe people, projects or documents (e.g. Dublin Core or FOAF). On the uppermost level, the user can create user-specific ontologies to fit her needs. For each level, only general architectural information is given, but no specific details or examples about the proposed ontologies, though.

In the context of another interesting prototype, the interface proposed by [21] improves image search by providing and using faceted metadata. Users can add flat or hierarchical categories of information to images, and then use them for filtering search results. Again, the idea is to provide an enhanced access to information, based on the different kinds of collected metadata.

2.3 Using Context Metadata to Find Information

[15] describes a very interesting approach for exploiting additional metadata for retrieving pictures. Their main idea is to rely on mostly automatically generated metadata (location, time and other digital photo metadata) and some manual annotations (events etc.) and to enhance these metadata automatically to provide information about actual light status (night, day, dawn, dusk), weather status and temperature, and additional aspects on the events, and then use these metadata to find stored images.

Another semantic search algorithm is proposed by [18]. It debuts with a classical text-based search on the metadata, whose output is then extended using the RDF network induced by the relations between semantic concepts, and finally reordered with techniques adapted from information retrieval.

[20] presents a new approach to content-based image retrieval. To improve the retrieval performance, the authors use a self-adjustable meta-database, which records the optimized relevance feedback information, representing the results obtained from previous queries from users that give a feedback on the relevance of the retrieved pictures. This kind of information partitions the images into classes denoting relevant images for future queries. The features taken into account by the algorithm are only low-level ones, though, such as HSV color-histograms or directional histograms.

3 Integrating Context Metadata Within Desktop Search

3.1 How Do Users Search?

Now how can we enhance desktop search with additional metadata? Clearly, if we know how users search, we can support their queries in an appropriate way. Recent studies of user web search behavior [4] have shown that the user goals can be classified into three main categories:

- *Navigational*: the user is searching for a specific web site, whose URL she forgot.
- *Informational*: the user is looking for information about a topic she is interested in.
- *Resource Seeking*: the user wants to find a specific resource (e.g. lyrics of a song, a program to download, a map service, etc.).

On our computer we are mainly interested in navigational queries, i.e. the user knows she stored a resource somewhere on the PC and now wants to find it again. Other less frequent, but possible, search goals are resource seeking (for example when searching for a previously installed application which plays MPEG-4 movies) and the close-directed subclass of informational queries [4] (searching for a resource annotated with a given description, such as "introduction to logic programming"). The other types of informational queries are almost inexistent on the desktop, as one generally has at least a vague picture of what is stored, and thus knows whether resources on a specific topic do exist on the PC or not².

Now clearly, when searching for something on our desktop we want to be able to exploit as much additional context as possible. In the following sections, we will discuss which context information is available for desktop search, how we can describe this context information using appropriate ontologies and how we can represent this information by explicit or inferred RDF metadata.³

After a brief presentation of current conventional approaches to desktop search (Section 3.2), we will analyze three important contexts which can be exploited to enhance desktop search: emails in Section 3.3, directory structures in 3.4, and the web cache in 3.5 and 3.6. For each context, we describe ontologies representing the available context information, and discuss both explicitly available metadata, as well as metadata that can be inferred and materialized using appropriate association rules.

3.2 Current Approaches to Desktop Search

The difficulty of accessing information on our computers has prompted several first releases of desktop search applications during the last months. The most prominent examples include Google desktop search [7] (proprietary, for Windows) and the Beagle open source project for Linux [6]. Yet they include *no* metadata whatsoever in their system, but just a regular text-based index. Nor does their competitor MSN Desktop Search [14]. Finally, Apple Inc. promises to integrate an advanced desktop search application (named *Spotlight Search* [1]) into their upcoming operating system, Mac OS Tiger. Even though they also intend to add semantics into their tool, only explicit information is used, such as file size, creator, last modification date, or metadata embedded into specific files (images taken with digital cameras for example include many additional characteristics, such as exposure information or whether a flash was used). While this is indeed an improvement over regular search, it still misses contextual information often

² If she knows that "something" is there, then the search becomes "navigational" or "resource seeking". If she knows there is nothing stored on the given topic, she would not search for it on her desktop.

³ Note, that even inferred metadata have to be materialized in order to enable efficient search.

resulting or inferable from explicit user actions or additional background knowledge, as discussed in the next sections.

In the following we will introduce four important search contexts, each with a small scenario, where ordinary full-text search fails, but additional context metadata provide the necessary information for finding the document we search for. For each context we will describe RDFS ontologies defining the metadata relevant for that context, as well as association rules and possible background knowledge which infer and materialize additional metadata.

3.3 Exploiting E-Mail Context

Scenario. Alice is interested in distributed page ranking, as her advisor asked her to write a report to summarize the state of the art in this research area. She remembers that during the last month she has discussed with a colleague about a distributed PageRank algorithm, and also that the colleague sent her the article via email. Though the article does not mention distributed PageRank, but instead talks about distributed trust networks, it is basically equivalent to distributed PageRank as her colleague remarked in this email. Obviously she should be able to find the article based on this additional information.

Context and Metadata. There are several aspects relevant to our email context. Sender and receiver fields of the email are clearly relevant pieces of information. Further information can be captured if we analyze the date of the email or the “reply_to” field, which gives thread information and is useful to determine social network information in general, for example which people discussed which topic etc.

Metadata should be generated automatically while the user works. For example, when an email is received, the system automatically generates email RDF metadata, instantiating e.g. “To”, “From” and “Comment” metadata from the email fields, and associating them to the document(s) attached to this email.

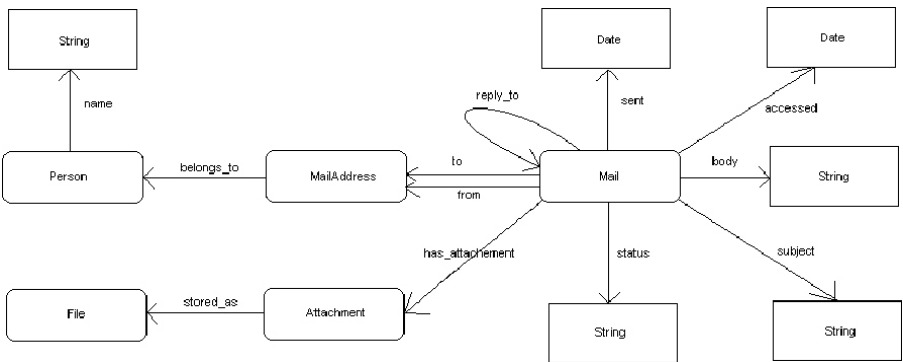


Fig. 1. Email prototype

Useful RDFS Ontologies. Basic properties for this context are properties referring to the date when an email was sent or the date it was accessed, the subject of the email and the email body. The status of an email can be described as seen/unseen or read/unread. We also have a property of the type *reply_to* which represents thread information. The *has_attachment* property describes a 1:n relation because a mail can have one or more attachments. The *to* and *from* properties connect to Class *MailAddress* which connects to Class *Person*. A *Person* is usually associated to more than one *MailAddress* instances. For attachments we keep the connection to the email it was saved from, because when we search for an attachment we want to use all attributes originally connected to the email it was attached to. The *stored_as* attribute is the inverse relation of the *File:stored_from* property we will see later.

Corresponding Association Rules. Association rules infer and materialize additional metadata information. For example, when creating the annotations, for each stored file we also associate a subject, derived from the subject of the email the file was attached to. The corresponding association rule, written in Datalog style, looks as follows:

$$\begin{aligned} \text{subject}(\text{File}, \text{Subject}) \leftarrow & \text{stored\_as}(\text{Attachment}, \text{File}), \\ & \text{has\_attachment}(\text{Mail}, \text{Attachment}), \\ & \text{subject}(\text{Mail}, \text{Subject}). \end{aligned}$$

Similarly, we also associate date and body text to the attached documents:

$$\begin{aligned} \text{accessed}(\text{File}, \text{Date}) \leftarrow & \text{stored\_as}(\text{Attachment}, \text{File}), \\ & \text{has\_attachment}(\text{Mail}, \text{Attachment}), \text{accessed}(\text{Mail}, \text{Date}). \\ \text{body}(\text{File}, \text{Body}) \leftarrow & \text{stored\_as}(\text{Attachment}, \text{File}), \\ & \text{has\_attachment}(\text{Mail}, \text{Attachment}), \text{body}(\text{Mail}, \text{Body}). \end{aligned}$$

as well as the name of the sender of the original email:

$$\begin{aligned} \text{from}(\text{File}, \text{Name}) \leftarrow & \text{stored\_as}(\text{Attachment}, \text{File}), \\ & \text{has\_attachment}(\text{Mail}, \text{Attachment}), \\ & \text{from}(\text{Mail}, \text{MailAddress}), \\ & \text{belongs\_to}(\text{MailAddress}, \text{Person}), \text{name}(\text{Person}, \text{Name}) \end{aligned}$$

In email threads connected through the *reply_to* relationship, we also inherit email subjects and bodies in addition to the original email subject / body:

$$\begin{aligned} & \text{subject}(\text{Mail}, \text{Subject}). \\ \text{subject}(\text{Mail}, \text{Subject}) \leftarrow & \text{reply\_to}(\text{Mail}, \text{Mail}_1), \text{subject}(\text{Mail}_1, \text{Subject}). \\ \text{body}(\text{Mail}, \text{Body}). \\ \text{body}(\text{Mail}, \text{Body}) \leftarrow & \text{reply\_to}(\text{Mail}, \text{Mail}_1), \text{body}(\text{Mail}_1, \text{Body}). \end{aligned}$$

Note that these association rules generate and materialize the appropriate metadata before the query is evaluated, and thus materialized metadata can be used directly during search, similar to the full text of the file / document. In our example, we can retrieve the correct document by using body text and sender information associated to this document, inherited from the original email.

3.4 Exploiting File Hierarchy Context

Scenario. In our second scenario, Alex spent his holiday in Hanover, Germany, taking a lot of digital pictures. He usually saves his pictures from a trip into a folder named after the city or the region he visits. However, he has no time to rename each image, and thus their file names are the ones used by his camera (for example "DSC00728.JPG"). When he forgets the directory name, no ordinary search can retrieve his pictures, as the only word he remembers, "Germany", does neither appear in the file names, nor in the directory structure. It would certainly be useful if an enhanced desktop search with "pictures germany" would retrieve his Hanover pictures.

Context and Metadata. In this example we need to consider file type and directory name information, and we need to be able to go beyond simple keyword search, taking part-of relationships and synonyms into account. To enrich the context metadata provided by file and directory names, we use WordNet [13], a lexical reference system which contains English nouns, verbs, adjectives and adverbs organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. In our case, we use the following additional relationships:

- *Hypernym*: Designates a class of specific instances. X is a hypernym of Y if Y is a (kind of) X.
- *Holonym*: Designates the superset of an object. A is a holonym of B if B is a part of A.
- *Synonyms*: A set of words that are interchangeable in some context. X is a synonym of Y if Y can substitute X in a certain context without altering the meaning.

Useful RDFS Ontologies. Obviously, our context metadata for files include the basic file properties like date of access and creation, as well as the file owner. File types can

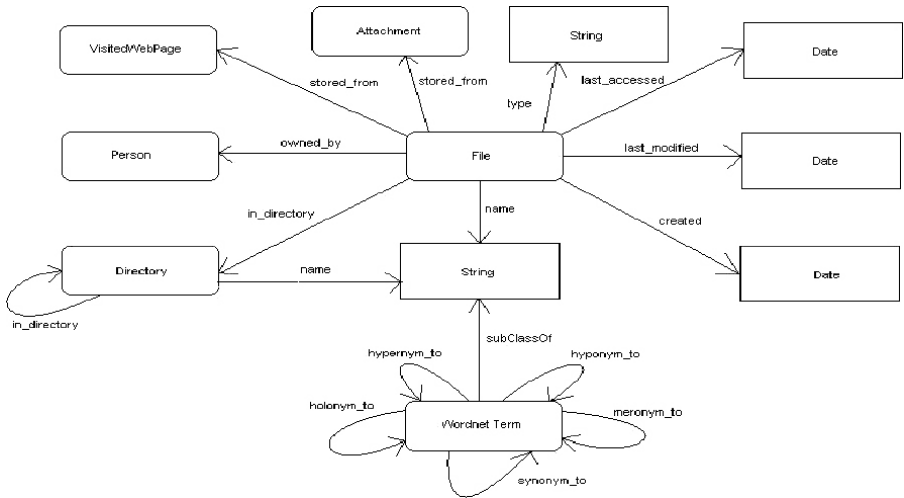


Fig. 2. File prototype

be inferred automatically, and provide useful information as well (in our case, the file is of type “JPEG image data”). Additionally, a file might be a visited web page which we stored on our computer or an attachment saved from an email. This *stored_from* property is of great importance because this represents information that current file systems miss, the provenance of information. We also keep track of the whole file path, including the directory structure. Finally, we extend the strings used in name and type metadata using WordNet information: synonyms, hypernyms, and holonyms. For each term we add the information provided by WordNet in order to enrich the context of the stored file.

Corresponding Association Rules. The use of WordNet induces the following association rules:

$$\begin{aligned} name(File, String_1) &\leftarrow name(File, String_2), synonym\_to(String_2, String_1). \\ name(File, String_1) &\leftarrow name(File, String_2), holonym\_to(String_2, String_1). \\ name(File, String_1) &\leftarrow name(File, String_2), hypernym\_to(String_2, String_1). \end{aligned}$$

Furthermore, we associate directory names as additional names to the contained files as well. The rules allow us to add explicit part-of information (“Hanover is part of Germany”), as well as synonym information (“picture” is a synonym to “image”), and enable us to successfully solve the search problem discussed in our scenario.

3.5 Exploiting the Web Cache Context for Visualization

Scenario. Even though Web search engines are providing surprisingly good results, they still need to be improved to take user context and user actions into account. Consider for example Paul, who is looking for the Microsoft internships web page, which he has previously visited, coming from the Microsoft main home page. If he does not remember the right set of keywords to directly jump to this page, it certainly would be nice if enhanced desktop search, based on his previous surfing behavior, would support him by returning the Microsoft home page, as well as providing the list of links from this page he clicked on during his last visit.

Context and Metadata. The context we have to use here can be extracted from Paul’s web cache, so we want to annotate each cached web page with additional information both for its basic properties (URL, access date, etc.), as well as more complex ones such as the used in-going and out-going links to other neighboring pages, reflecting Paul’s surfing behavior. This way, when browsing a certain cached page, enhanced desktop search can also provide information about the context in which that document has been useful for the user, i.e. how it was reached or which links were followed from there.

Useful RDFS Ontologies. Correspondingly, the central class in this scenario’s ontology is the class *VisitedWebPage*. Upon visiting a web page, the user is more interested in the links she has used on that page, rather than every possible link which can be followed from there. Thus, the metadata contains only the hyperlinks *accessed* for each stored web page:

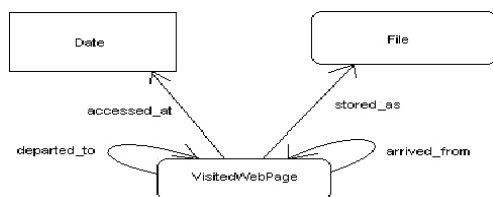


Fig. 3. WebPage prototype

- *departed_to* is a relation of the type one to many (as the user could have accessed many pages from a web page) which shows the hyperlinks the user clicked on the current web page;
- *arrived_from* is a relation representing the page(s) the user came from.

Also here, we have added properties related to the time of access and place of storage in the hard disk cache. For specific scenarios we can define subclasses of this base class, which include scenario specific attributes, for example recording the browsing behavior in CiteSeer, which we will discuss in the next section.

Corresponding Association Rules. There are no specific association rules materializing inferred metadata we need for our scenario. Instead we use our metadata for enriching search results. Displaying context information for enhanced browsing under this scenario uses a similar layout as the TAP search screen, with the web pages or documents from the cache provided in the main window, and an additional frame to display the context information using the *departed_to* and *arrived_from* relations.

3.6 Exploiting the Web Cache Context to Enrich Search Results

Scenario. If we have more information about the web pages visited, we can provide even better context information. Suppose that Alice browses through CiteSeer for papers on a specific topic, following reference links to and from appropriate papers, and downloads the most important documents onto her computer. Now as soon as they are stored in one of her directories, her carefully selected documents are just another bunch of files without any relationships. They have completely lost all information present in CiteSeer, in this case which paper references specific other papers or is referenced by another paper, and which papers Alice deemed important enough not only to look at but also to download. It is the task of a semantic desktop search environment to preserve that information and make it available as explicit metadata.

Context and Metadata. As discussed, stored files on today's computers do not tell us whether they were saved from a web page or from an email, not to mention the URL of the web page, out-going or in-going visited links and more specific information inferable from this information and a model of the web page context browsed, as discussed in our scenario. All this information should be covered by our metadata to connect the stored files to their original contexts, and thus allow the user to exploit all the previous knowledge and context she gathered around them.

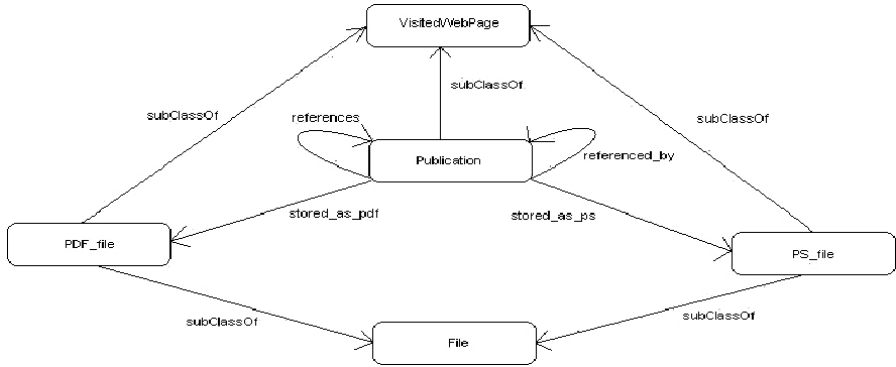


Fig. 4. Publication prototype

Useful RDFS Ontologies. In our scenario we make use of additional knowledge about how CiteSeer pages are connected. We therefore create a subclass of *VisitedWebPage* called *Publication*, and add suitable properties as described in figure 4. The *Publication* class represents a CiteSeer document web page. It records the CiteSeer traversed links from that page using the *references* property and the CiteSeer documents which the user visited before using the *referenced_by* property. It is easy to notice that these pages represent a subset of the metadata captured by the *departed_to* and *arrived_from* relations. *PDF_file* and *PS_file* are subclasses of *File*, and are connected to *Publication* with subproperties of “*stored_as*”, namely “*stored_as_pdf*” and “*stored_as_ps*”.

Corresponding Association Rules. In our semantic desktop search environment we use these metadata to enrich the search results by displaying the context of the document found in the form of downloaded papers referencing that document, or downloaded papers referenced by the document. This can be expressed for example by an association rule such as the following one:

$$\text{downloaded\_references}(\text{Document}, \text{File}) \leftarrow$$

$$\begin{aligned} &\text{stored\_as}(\text{Publication}_1, \text{Document}), \\ &\text{references}(\text{Publication}_1, \text{Publication}_2), \\ &\text{stored\_as}(\text{Publication}_2, \text{File}). \end{aligned}$$

4 Desktop Search Architecture and Prototype

4.1 Generating Input Metadata

Event Triggered Metadata Generation. The main characteristic of our desktop search architecture is metadata generation and indexing on-the-fly, triggered by modification events generated upon occurrence of file system changes. This relies on notification functionalities provided for example by the kernel. Events are generated whenever a new file is copied to hard disk or stored by the web browser, when a file is deleted or

modified, when a new email is read, etc. Much of this basic notification functionality is provided on Linux by an inotify-enabled Linux kernel, which we use in our prototype.

Metadata Generator Applications. Depending on the type and context of the file / event, metadata generation is then performed by appropriate metadata generator applications, as described in the next paragraphs. These applications build upon an appropriate RDFS ontology as described in the previous sections describing the RDF metadata to be used for that specific context. Generated metadata are either extracted directly (e.g. email sender, subject, body) or are generated using the appropriate association rules plus possibly some additional background knowledge (e.g. the WordNet ontology in our prototype). All of these metadata are exported in RDF format, and added to a metadata index, which is used by the search application together with the usual full-text index.

The architecture of our prototype environment is depicted in Figure 5. It includes three prototype metadata generator applications, based on the scenarios described in the previous section 3. We will shortly describe them in the following paragraphs.

Email Metadata Generator. Our current email prototype is built on top of the Java-Mail API [10]. It processes the incoming emails into a separate class, derived from the *Message* class defined in JavaMail. The associated metadata is easily generated according to figure 1, as the *Message* class already provided helpful methods in this direction (e.g. "*getTo*", "*getRecipients*", "*getSubject*" and "*getSentDate*"). Further metadata are generated when attachments are stored in the file system. Metadata are stored as RDF using the Jena toolkit [11]. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF and RDFS, including a rule-based inference engine which we use to implement our association rules.

File Metadata Generator. Upon creation of a new file, its path is decomposed into a sequence of tokens, one for each level of the directory tree existing on the hard disk. Each of these tokens is added as metadata description to the file, together with the usual file attributes, as described in section 3.4. We use WordNet to add additional metadata (WordNet senses) both to the file name and to each token of the path, thus capturing all meaningful information implicitly available through the file and folder names. Our file prototype is again implemented in Java, and uses the JWNL API [12] to access the WordNet relational dictionary. As in the previous module, we also use the Jena API to generate the RDF file that contains the annotation corresponding to the file structure, in which each indexed file is a resource.

As future work, we intend to extract additional specific information stored in several widely used file types. For example, many image formats provide specific additional metadata, such as exposure information. Another possible improvement for this generator is to use additional background knowledge about seasons etc., as well as to let the user manually add more annotations to files or directories. We could then search for the pictures we took during the last winter in Germany, or during a special event in our life, like a birthday.

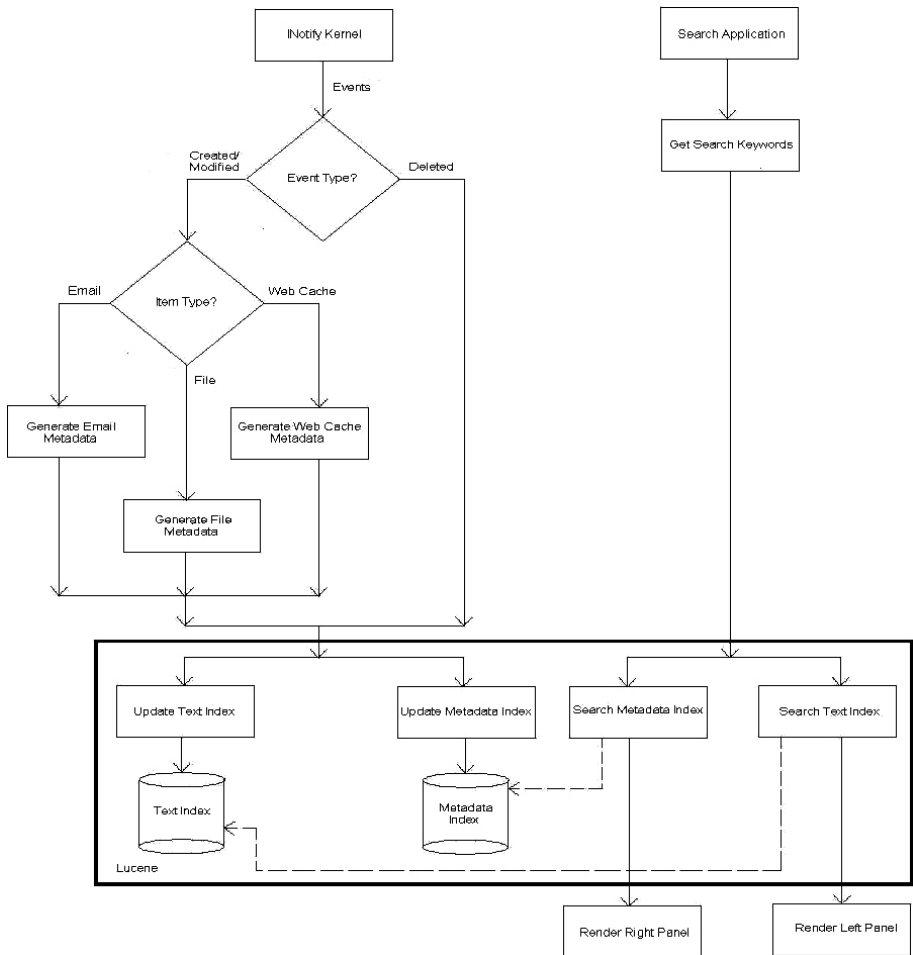


Fig. 5. Prototype Application Architecture

Web Cache Metadata Generator. In the web cache prototype, the annotation of the cached web pages is triggered by browsing web pages which were not previously stored in the local cache. Generation starts with the basic annotations for each web page (e.g. access date) and then proceeds with the annotations representing the connections between web pages (for example from which page did the user arrive at the current one, or which hyperlinks of the current page are traversed). Again, we use the Jena API to export the annotations in RDF format.

For specific sites, the metadata generator uses additional ontologies. In our prototype this is done when using the CiteSeer repository. These ontologies then trigger additional metadata generation, which can be used during search, as well as for enriching search results.

4.2 Displaying and Enriching Search Results

Enhanced semantic desktop search provides a search service similar to its web sibling. However, rather than searching only one through the full-text index, it also searches the additional metadata index, with each metadata item linked to the resource it has been derived from.

The regular search interface is as simple as the one provided by Google, i.e. an input text box for the searched terms and a search button. This type of search looks for the keywords in both indexes automatically. Results are then presented as in TAP [8]: the left side of the output window displays the hits matched from the text index, and the right side contains additional information provided through the metadata associated with the chosen result document.

The items displayed on the right hand side obviously depend on the type of the result document. For the web cache scenario this allows us to show not only the previously browsed pages, but also the entire context in which they have been used and accessed (for example where did the user go from each page, or which referenced papers the user downloaded related to a found document). This helps a lot, as the user now has all the orienteering steps [9] right in front of her.

An additional advanced search interface allows the user to restrict her search to one of the two indexes. Moreover, she can define filters, by choosing where to search (only in the emails, files or the web cache), each category also allowing other additional filters according to its ontology (e.g. *To*, *From*, *Reply To*, *Subject*, *Attachment*, etc. in the email scenario).

5 Conclusions and Further Work

Advanced desktop search needs semantics and metadata. Applying search engine technology on the desktop is useful, but not sufficient, because sophisticated heuristics and algorithms like PageRank, which are very successful on the web, do not work on the desktop. On the other hand, our personal desktop environment provides a lot of context not available on the web, which can be used to implement sophisticated semantic search functionalities on our desktop surpassing those possible on the web.

This paper presents concept, architecture and prototype for a semantic desktop search environment, which promises to exploit the information present in these contexts, accumulated by user activities and additional background knowledge. Our search environment relies on ontologies describing appropriate metadata for different contexts relevant on the desktop and uses these semantic annotations to both extend search functionalities and enrich search results.

The semantic desktop search environment contains two distinct modules. The first one is *the index*, which consists of a metadata repository including all metadata associated to each resource on the desktop, as specified in the appropriate context ontologies, and a regular search engine full-text index of these resources. The second module is *the search module*, which combines keyword search on the full-text index with semantic search on the metadata repository to provide both improved functionalities for finding information on our PC, as well as enriching the search results and visualizing existing contexts using the additional knowledge stored in the metadata repository.

Comparing the possibilities for a semantic desktop search environment to semantic search on the web, we believe that semantic web technologies might ultimately be more important on the desktop than on the web. This is because, first, our desktop environment is “limited” in the sense that we will be able to describe most relevant contexts rather easily, and thus will be able to provide more complete ontologies / metadata specifications for the desktop environment than for the web in general. Second, even with 200GB hard disks in our computers, the amount of data and metadata itself is limited compared to the information available on the web, so more sophisticated algorithms for using semantic annotations are feasible on the desktop than on the web.

We are currently working on integrating our metadata repository and tools into one of the existing approaches to desktop search, Gnome Beagle [6], where we can re-use their conventional infrastructure for full-text search on the desktop. Additionally, we are extending our available context ontologies and metadata generation functionalities beyond the current status as described in this paper, in conjunction with several user surveys meant to capture both the requirements of a larger set of users, as well as to measure the improvements provided by adding semantic annotations to desktop search.

References

1. Apple spotlight search. <http://developer.apple.com/macosx/tiger/spotlight.html>.
2. Stefan Decker and Martin Frank. The social semantic desktop. In *DERI Technical Report 2004-05-02*, 2004.
3. P. Dolog, N. Henze, W. Nejdl, and M. Sintek. Personalization in distributed elearning environments. In *Proceedings of the 13th World Wide Web Conference*, 2004.
4. Rose D. E. and Levinson D. Understanding user goals in web search. In *Proc. of WWW 2004, May 17-22, 2004, New York, USA*, 2004.
5. Benja Fallenstein. Fentwine: A navigational rdf browser and editor. In *Proceedings of 1st Workshop on Friend of a Friend, Social Networking and the Semantic Web*, 2004.
6. Gnome beagle desktop search. <http://www.gnome.org/projects/beagle/>.
7. Google desktop search application. <http://desktop.google.com/>.
8. R. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the twelfth international conference on World Wide Web*, pages 700–709. ACM Press, 2003.
9. Teevan J., Alvarado C., Ackerman M. S., and Karger D. R. The perfect search engine is not enough: A study of orienteering behavior in directed search. In *In Proc. of CHI*, 2004.
10. Javamail api. <http://java.sun.com/products/javamail/>.
11. Jena api. <http://jena.sourceforge.net/>.
12. Jwnl api. <http://sourceforge.net/projects/jwordnet>.
13. G.A. Millet. Wordnet: An electronic lexical database. *Communications of the ACM*, 38(11):39–41, 1995.
14. Msn desktop search application. <http://beta.toolbar.msn.com/>.
15. Mor Naaman, Susumu Harada, Qian Ying Wang, Hector Garcia-Molina, and Andreas Paepcke. Context data in geo-referenced digital photo collections. In *Proceedings of the 12th annual ACM International Conference on Multimedia*, 2004.
16. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.
17. Dennis Quan and David Karger. How to make a semantic web browser. In *Proceedings of the 13th International WWW Conference*, 2004.

18. Cristiano Rocha, Daniel Schwabe, and Marcus Poggi de Aragao. A hybrid approach for searching in the semantic web. In *Proceedings of the 13th International World Wide Web Conference*, 2004.
19. Leopold Sauermaann. Using semantic web technologies to build a semantic desktop. Master's thesis, TU Vienna, 2003.
20. Yimin Wu and Aidong Zhang. Category-based search using metadatabase in image retrieval. In *IEEE International Conference on Multimedia and Expo*, 2002.
21. Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the conference on Human factors in computing systems*, 2003.

An Ontology-Based Information Retrieval Model

David Vallet, Miriam Fernández, and Pablo Castells

Universidad Autónoma de Madrid

Campus de Cantoblanco, c/ Tomás y Valiente 11, 28049 Madrid

{david.vallet, miriam.fernandez, pablo.castells}@uam.es

Abstract. Semantic search has been one of the motivations of the Semantic Web since it was envisioned. We propose a model for the exploitation of ontology-based KBs to improve search over large document repositories. Our approach includes an ontology-based scheme for the semi-automatic annotation of documents, and a retrieval system. The retrieval model is based on an adaptation of the classic vector-space model, including an annotation weighting algorithm, and a ranking algorithm. Semantic search is combined with keyword-based search to achieve tolerance to KB incompleteness. Our proposal is illustrated with sample experiments showing improvements with respect to keyword-based search, and providing ground for further research and discussion.

1 Introduction

The use of ontologies to overcome the limitations of keyword-based search has been put forward as one of the motivations of the Semantic Web since its emergence in the late 90's. While there have been contributions in this direction in the last few years, most achievements so far either make partial use of the full expressive power of an ontology-based knowledge representation, or are based on boolean retrieval models, and therefore lack an appropriate ranking model needed for scaling up to massive information sources.

In the former case, ontologies provide a shallow representation of the information space, equivalent in essence to the taxonomies and thesauri used before the Semantic Web was envisioned [3,6,7,15]. Rather than an instrument for building knowledge bases, these light-weight ontologies provide controlled vocabularies for the classification of content, and rarely surpass several KBs in size. This approach has brought improvements over classic keyword-based search through e.g. query expansion based on class hierarchies and rules on relationships, or multifaceted searching and browsing. It is not clear though that these techniques alone really take advantage of the full potential of an ontological language, beyond those that could be reduced to conventional classification schemes.

Other semantic search techniques have been developed that do exploit large knowledge bases in the order of GBs or TBs consisting of thousands of ontology instances, classes and relations of arbitrary complexity [1,2,4,12]. These techniques typically use boolean search models, based on an ideal view of the information space as

consisting of non-ambiguous, non-redundant, formal pieces of ontological knowledge. In this view, the information retrieval problem is reduced to a data retrieval task. A knowledge item is either a correct or an incorrect answer to a given information request, thus search results are assumed to be always 100% precise, and there is no notion of approximate answer to an information need. This model makes sense when the whole information corpus can be fully represented as an ontology-driven knowledge base, so that search results consist of ontology entities.

However, there are limits to the extent to which knowledge can or should be formalized in this way. First, because of the huge amount of information currently available to information systems worldwide in the form of unstructured text and media documents, converting this volume of information into formal ontological knowledge at an affordable cost is currently an unsolved problem in general.

Second, documents hold a value of their own, and are not equivalent to the sum of their pieces, no matter how well formalized and interlinked. The replacement of a document by a bag of information atoms inevitably implies a loss of information value: the thread of thought behind the order of the sentences in free text, the choice of the words, etc., are a valuable, relevant, and necessary part of the conveyed message. Therefore, although it is useful to break documents down into smaller information units that can be reused and reassembled to serve different purposes, it is yet often appropriate to keep the original documents in the system.

Third, wherever ontology values carry free text, boolean semantic search systems do a full-text search within the string values. In fact, if the string values hold long pieces of free text, a form of keyword-based search is taking place in practice beneath the ontology-based query model since, in a way, unstructured documents are hidden within ontology values, whereby the “perfect match” assumption starts to become arguable, and search results may start to grow in size. While this may be manageable and sufficient for small knowledge bases, the boolean model does not scale properly for massive document repositories where searches typically return hundreds or thousands results. Boolean search does not provide clear ranking criteria, without which the search system may become useless if the search space is too big.

In this paper we propose an ontology-based retrieval model meant for the exploitation of full-fledged domain ontologies and knowledge bases, to support semantic search in document repositories. In contrast to boolean semantic search systems, in our perspective full documents, rather than specific ontology values from a KB, are returned in response to user information needs. The search system takes advantage of both detailed instance-level knowledge available in the KB, and topic taxonomies for classification. To cope with large-scale information sources, we propose an adaptation of the classic vector-space model [16], suitable for an ontology-based representation, upon which a ranking algorithm is defined.

The performance of our proposed model is in direct relation with the amount and quality of information within the KB it runs upon. The latest advances in automating ontology population and text annotation are promising [5,9,11,14]. While, if ever, ontologies and metadata (and the Semantic Web itself) become a worldwide commodity, the lack or incompleteness of available ontologies and KBs is a limitation we shall likely have to live with in the mid term. In consequence, tolerance to incomplete KBs

has been set as an important requirement in our proposal. This means that the recall and precision of keyword-based search shall be retained when ontology information is not available or incomplete.

We have implemented our model and done some low-scale experimentation with real documents and data from a digital news archive from a local Spanish newspaper. The experiments build upon previous work in the Neptuno project [1], where an ontology and a knowledge base were built for the description of archive news.

The rest of the paper is organised as follows. An overview of related work is given in Section 2. After this, our scheme for semantic annotation is described. Section 4 explains the retrieval and ranking algorithms. Some initial experiments with our techniques are reported in Section 5. The strengths, weaknesses, and significance of our approach are summarized in Section 6, after which some conclusions are given.

2 State of the Art

Our view of the semantic retrieval problem is very close to the latest proposals in KIM [11,14]. While KIM focuses on automatic population and annotation of documents, our work focuses on the ranking algorithms for semantic search. Along with TAP [8], KIM is one of the most complete proposals reported to date, to our knowledge, for building high-quality KBs, and automatically annotating document collections at a large scale. In their latest account of progress [11] a ranking model for retrieval is hinted at but has not been developed in detail and evaluated. In fact, KIM relies on a keyword-based IR engine for this purpose (indexing, retrieval and ranking). Our work complements KIM with a ranking algorithm specifically designed for an ontology-based retrieval model, using a semantic indexing scheme based on annotation weighting techniques.

TAP [8] presents a view of the Semantic Web where documents and concepts are nodes alike in a semantic network, whereby the separation of contents and metadata is not as explicit as we propose here. The two main problems addressed by TAP are a) the development of a distributed query infrastructure for ontology data in the Semantic Web, and b) the presentation of query execution results, augmenting query answers with data from surrounding nodes. These issues are complementary to the ones addressed in this paper. However the expressive power of the TAP query language is fairly limited compared to ontology query languages such as RDQL, RQL, etc. The supported search capability is limited to keyword search within the “title properties” of instances, and no ranking is provided.

Mayfield and Finin [13] combine ontology-based techniques and text-based retrieval in sequence and in a cyclic way, in a blind relevance feedback iteration. Inference over class hierarchies and rules is used for query expansion, and extension of semantic annotations of documents. Documents are annotated with RDF triples, and ontology-based queries are reduced to boolean string search, based on matching RDF statements with wildcards, at the cost of losing expressive power for queries. We share with Mayfield et al the idea that semantic search should be a complement of keyword-based search as long as not enough ontologies and metadata are available. Also, we

believe that inferencing is a useful tool to fill knowledge gaps and missing information (e.g. transitivity of the *locatedIn* relationship over geographical locations).

Semantic Portals [1,2,4,12] typically provide simple search functionalities that may be better characterised as semantic data retrieval, rather than semantic information retrieval. Searches return ontology instances rather than documents, and no ranking method is provided. In some systems, links to documents that reference the instances are added in the user interface, next to each returned instance in the query answer [4], but neither the instances, nor the documents are ranked. Maedche et al do provide a criterion for query result ranking in the SEAL Portal [12], but the principles on which the method is based – a similarity measure between query results and the original KB without axioms, is not clearly justified, and no testing of the method is reported.

The ranking problem has been taken up again in [19], and more recently [15]. Rocha et al propose the expansion of query results through arbitrary ontology relations starting from the initial query answer, where the distance to the initial results is used to compute a similarity measure for ranking [15]. This method has the advantage of allowing the user to express information needs with simpler, keyword-based queries but, from our perspective, it introduces an unnecessary loss of precision, since a more accurate result expansion can be achieved by including ontology relations explicitly in a structured query. From our point of view, Rocha's techniques would be appropriate in a more browsing-oriented information seeking context. Stojanovic et al propose a sentence ranking scheme based on the number of times an instance appears as a term in a relation type, and the derivation tree by which a sentence is inferred [19]. Whereas these works are concerned with ranking query answers (i.e. ontology instances), we are concerned with ranking the documents annotated with these answers. Since our respective techniques are applied in consecutive phases of the retrieval process, it would be interesting to experiment the integration of the query result relevance function proposed by Stojanovic et al into our document relevance measures.

3 Knowledge Base and Document Base

In our view of semantic information retrieval, we assume a knowledge base has been built and associated to the information sources (the document base), by using one or several domain ontologies that describe concepts appearing in the document text. The concepts and instances in the KB are linked to the documents by means of explicit, non-embedded annotations to the documents.

While we do not address here the problem of knowledge extraction from text [4,5,9,10,11,14], we provide a vocabulary and some simple mechanisms to aid in the semi-automatic annotation of documents, once ontology instances have been created (manually or automatically). These are described in Subsection 3.2. Our system can work with any arbitrary domain ontology with essentially no restrictions, except for some minimal requirements that are explained next.

3.1 Root Ontology Classes

Our system requires that the knowledge base be constructed from three main base classes: *DomainConcept*, *Taxonomy*, and *Document*. *DomainConcept* should be the root of all domain classes that can be used (directly or after subclassing) to create instances that describe specific entities referred to in the documents. For example, in the Arts domain, classes like *Artist*, *Sculptor*, *ArtWork*, *Painting*, and *Museum* should be defined as (probably indirect) subclasses of *DomainConcept*. A small set of upper-level open-domain classes like *Person*, *Building*, *Event*, *Location*, etc., is included in the base concept ontology, to be extended for specific domains.

Document is used to create instances that act as proxies of documents from the information source to be searched upon. Two subclasses, *TextDocument* and *MediaContent*, are provided, which can be further subclassed, if appropriate for a particular application domain, to provide for different types of documents, such as *Report*, *News*, *PurchaseOrder*, *Invoice*, *Message*, etc., with different fields (e.g. *title*, *date*, *subject*, *price*, *sender*). The class *MediaContent* is provided in anticipation of future extensions for multimedia retrieval, which we have not developed yet. *Document* has a *location* property that holds a dereferenceable physical address (in our current implementation, a URL) from which the actual document contents can be retrieved.

Taxonomy is the root for class hierarchies that are merely used as classification schemes, and are never instantiated. These taxonomies are expected to be used as a terminology to annotate documents and concept classes, using them as values of dedicated properties. For instance, in a KB for news, classes like *Culture*, *Politics*, *Economy*, *Sports*, etc. (after the IPTC Subject Reference System standard¹), could be used as values of a (probably multivalued) *topic* property of the *News* class. Furthermore, concept classes like *Athlete* and *Tournament* could also have the *topic* property, in this case with the value *Sports*, i.e. concepts can also be classified under the same scheme as documents. Several separate taxonomies can be used simultaneously on the same documents, thus providing for multifaceted classification.

The distinction between the three root classes *DomainConcept*, *Taxonomy*, and *Document*, arises from our own experience in previous Semantic Web projects [1,2], and many other observed information systems where this (or a similar distinction) seems to be natural, useful and recurrent (see e.g. [17]). In our system, we exploit taxonomies for multifaceted search, and to solve word ambiguities, as will be described later.

3.2 Document Annotation

The predefined base ontology classes described above are complemented with an annotation ontology that provides the basis for the semantic indexing of documents with non-embedded annotations. In many respects, our scheme for semi-automatic annotation is similar to the one recently reported in [11].

Documents are annotated with concept instances from the KB by creating instances of the *Annotation* class, provided for this purpose. *Annotation* has two relational

¹ <http://www.iptc.org/NewsCodes>

properties, *instance* and *document*, by which concepts and documents are related together. Reciprocally, *DomainConcept* and *Document* have a multivalued *annotation* property.

Annotations can be created manually by a domain expert, or semi-automatically. The subclasses *ManualAnnotation* and *AutomaticAnnotation* are used respectively, to differentiate each case. We have found this distinction useful for the system at least because a) manual annotations are more reliable than automatic ones, and when available should prevail, and b) while automatic annotations can be deleted for recalculation, manual annotations should be preserved.

Our system provides a simple facility for semi-automatic annotation, which works as follows. *DomainConcept* instances use a *label* property to store the most usual text form of the concept class or instance. This property is multivalued, since instances may have several textual lexical variants. Close equivalents of our *label* property are used in systems like KIM [11] and TAP [8]. In our current experiments, the value of this property is set by hand by the ontology designer, but it could be set by automatic means, if an external instance generation system was plugged to our system. Similarly to KIM, instance labels are used by the automatic annotator to find potential occurrences of instances in text documents. Whenever the label of an instance is found, an annotation is created between the instance and the document. In our system, documents can be annotated with classes as well, by assigning labels to concept classes.

This basic mechanism is complemented with heuristics to cope with polysemia, i.e. label coincidence between different instances or classes. First the system always tries to find the longest label, e.g. “Real Madrid” is preferred to “Madrid”. Second, classification taxonomies are used as a source of semantic scope for disambiguation: a similarity measure is defined to compare the respective classification of the document and candidate synonym instances for annotation, so that the instance that has the closest classification to the document is chosen. For example, the word “Irises” in a document classified under *Arts* would be linked to an instance of *Painting* that represents Van Gogh’s famous work, rather than a subclass of *Flower*, provided that the painting instance exists in the knowledge base and has been correctly classified under *Arts*, or a taxonomic subclass thereof, and assuming that *Flower* is classified under a different taxonomic branch such as *Botany* or the like. Of course, if the *Painting* instance does not exist, our system fails because it would incorrectly annotate the document with the botanic sense.

Our semi-automatic annotation mechanisms can be further improved, but this is out of the extent of our undergoing research. More sophisticated annotation techniques, as have been reported in the literature [5,9,11,14], would be complementary and beneficial to our system.

3.3 Weighting Annotations

The annotations are used by the retrieval and ranking module, as will be explained in Section 4. The ranking algorithm is based on an adaptation of the classic vector-space model [16]. In the classic vector-space model, keywords appearing in a document are assigned weights reflecting that some words are better at discriminating between

documents than others. Similarly, in our system, annotations are assigned a weight that reflects how relevant the instance is considered to be for the document meaning. Weights are computed automatically by an adaptation of the TF-IDF algorithm [16], based on the frequency of occurrence of the instances in each document. More specifically, the weight $w_{i,j}$ of instance I_i for document D_j is computed as:

$$w_{i,j} = \frac{freq_{i,j}}{\max_k freq_{k,j}} \times \log \frac{N}{n_i}$$

Where $freq_{i,j}$ is the number of occurrences of I_i in D_j , $\max_k freq_{k,j}$ is the frequency of the most repeated instance in D_j , n_i is the number of documents annotated with I_i , and N is the total number of documents in the search space.

The number of occurrences of an instance in a document is primarily defined as the number of times the label of the instance appears in the document text, if the document is annotated with the instance, and zero otherwise. We realised in our first experiments that quite a number of occurrences were missed in practice with this approach, since pronouns, periphrasis, metonymy, and other deixis abound in regular written speech. Finding all the references to an individual in free text is a very complex natural language processing problem far beyond the scope of our current research. Nonetheless we have achieved significant improvements by extending our labeling scheme and exploiting class hierarchies, as follows.

First, further instance occurrences are found by adding more labels to instances. However, the proliferation of labels tends to introduce further polysemic ambiguities that lead to incorrect annotations. To avoid this negative effect, our system provides a separate *keyword* property to be used, in addition to *label*, for instance frequency computation, but not for automatic annotation. As a general rule, *label* should be reserved to clearly instance-specific text forms, leaving more ambiguous ones as *keywords*. Since instance occurrences are only computed in the presence of an annotation, very few or no ambiguities are caused in practice.

Also, synecdoche is a frequent rhetoric figure used to avoid repetition, where an individual is referred to by its class (e.g. “the painter”), after the individual (e.g. “Picasso”) has already appeared in the text. To cope with this, the list of textual forms (labels and keywords) of an instance is automatically expanded (just for the computation of occurrences) with the textual forms of its direct and indirect classes. This introduces a slight occurrence counting imprecision when more than one instance of the same class are annotating the same document, because the same class references are counted once for each instance. However, in our experiments the improvements obtained with this technique outweigh the effect of the imprecision.

4 Processing Queries

Our approach to ontology-based information retrieval can be seen as an evolution of classic keyword-based retrieval techniques, where the keyword-based index is replaced by a semantic knowledge base. The overall retrieval process is illustrated in Fig. 1. Our system takes as input a formal RDQL query. This query could be gener-

ated from a keyword query, as in e.g. [8,15,18], a natural language query [4], a form-based interface where the user can explicitly select ontology classes and enter property values [1,11,12], or more sophisticated search interfaces [7]. A number of research works have undertaken the construction of easy to use user interfaces for ontology query languages, and we do not address this problem here. The RDQL query is executed against the knowledge base, which returns a list of instance tuples that satisfy the query. Finally, the documents that are annotated with these instances are retrieved, ranked, and presented to the user.

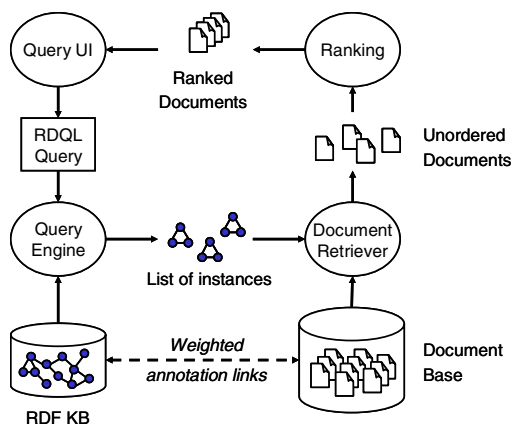


Fig. 1. Our view of ontology-based information retrieval

4.1 Query Encoding and Document Retrieval

The RDQL query can express conditions involving domain ontology instances, document properties (such as *author*, *date*, *publisher*, etc.), or classification values. E.g. “cultural articles published by the Le Monde newspaper about European movies with Canadian actors in the cast.”

In classic keyword-based vector-space models for information retrieval, the query keywords are assigned a weight that represents the importance of the concept in the information need expressed by the query. Analogously, in our model, the variables in the SELECT clause of the RDQL query can be weighted to indicate the relative interest of the user for each of the variables to be explicitly mentioned in the documents. For instance, in the previous example, the user might be interested that both the movies and the Canadian actors are mentioned in the articles, or have a higher priority for either the movies or the actors. The weights can be set explicitly by the user, or be automatically derived by the system, e.g. based on frequency analysis, personalisation techniques, or other strategies [6].

Our system uses inferencing mechanisms for implicit query expansion based on class hierarchies (e.g. organic pigments can satisfy a query for colorants), and rules such as one by which the winner of a sports match might be inferred from the scoring.

In fact, in our current implementation, it is the KB which is expanded by adding inferred statements beforehand.

The query execution returns a set of tuples that satisfy the query. It is the document retriever's task to obtain all the documents that correspond to the instance tuples. If the tuples are only made up of instances of domain concepts, the retriever follows all outgoing annotation links from the instances, and collects all the documents in the repository that are annotated with the instances. If the tuples contain instances of document classes (because the query included direct conditions on the documents), the same procedure is followed, but restricted to the documents in the result set, instead of the whole repository.

4.2 Ranking Algorithm

Once the list of documents is formed, the search engine computes a semantic similarity value between the query and each document, as follows. Let $O = \{I_i\}_{i=1}^M$ be the set of all instances in the ontology, and $\{D_i\}_{i=1}^N$ be the set of all documents in the search space. Let $(v_1, \dots, v_k)$ be the weights of the variables in the SELECT clause of the RDQL query Q , and let $T = \{T_i\}_{i=1}^n$ be the list of tuples in the query result set, where $T_i = \{T_{i,j}\}_{j=1}^k$, with $T_{i,j} \in O$.

We define the *document vector* of D_i as $\vec{d}_i = (d_{i,1}, \dots, d_{i,M})$, where $d_{i,j}$ is the weight of the annotation of document D_i with I_j , if such annotation exists, and zero otherwise. We define the *extended query vector* as $\vec{q} = (q_1, \dots, q_M)$, where $q_l = \sum_{\exists i I_l = T_{i,j}} v_j$, i.e. the

query vector element corresponding to I_l is added the variable weight v_j if I_l is a value of the variable j in some tuple T_i that satisfies the query Q . Note that the sum rarely has more than one term, since this would mean that the same instance appears more than once in the same result set tuple. If I_l does not appear in any tuple, $q_l = 0$.

Now the similarity measure of D_i for the query Q is computed as:

$$\text{sim}(D_i, Q) = \frac{\vec{d}_i \cdot \vec{q}}{|\vec{d}_i| \times |\vec{q}|}$$

Because of the way $\vec{q}$ is constructed, $|\vec{q}|$ is usually quite large, and as a consequence the values of the similarity function are too low. For example, if the user queries for special offers for summer holidays in the Aegean Islands, a document that shows one such offer will get a similarity value in the order of $1/n$, where n is the total number of registered offers in the knowledge base that match the query. Only a document that displays nearly all offers could get close to similarity 1. To compensate for this, in practice we use the following normalization factor instead of $|\vec{q}|$:

$$\sqrt{\sum_{j=1}^k (v_j^2 \times \max_i \#T_j^i)}, \text{ where } T_j^i = \{I \in O \mid I \text{ annotates } D_i \wedge \exists i, I = T_{i,j}\}.$$

If the knowledge in the KB is incomplete (e.g. there are documents about travel offers in the knowledge source, but the corresponding instances are missing in the KB), the semantic ranking algorithm performs very poorly: RDQL queries will return less

results than expected, and the relevant documents will not be retrieved, or will get a much lower similarity value than they should. As limited as might be, keyword-based search could perform better in these cases. To cope with this, our ranking model combines the semantic similarity measure with the similarity measure of a keyword-based algorithm. The final value for ranking is computed as $t \times \text{sim}(D_i, Q) + (t - 1) \times \text{ksim}(D_i, Q)$, where *ksim* is computed by a keyword-based algorithm. We have taken $t = 0.5$, which seems to perform well in our experiments. As a further adjustment, if *ksim* returns 0, we take $t = 1$, and if *sim* returns 0, we take $t = 0.2$. For further testing, we have implemented a user interface where this parameter can be freely set by the user with a slider after the search has been executed, so that the user can see dynamically how the results are reranked as the value of t is moved.

The keywords for the *ksim* algorithm could be extracted directly from the user query, if a keyword-based or even natural language interface was used. In our current implementation we extract the keywords from the RDQL query, which is suitable for testing, and would be appropriate for a form-based interface as well. More specifically, the value of the *label* property of a) the class of all query variables for which a *rdf:type* clause is included in the query, and b) any instances explicitly appearing within the RDQL query, are taken as query keywords. In practice, since the *label* property can be multivalued, a separate property is used, which stores one of the label values, designated as a unique most common lexical form. For example, the query:

```
SELECT ?player ?team
WHERE (?player rdf:type sports:Player)
      (?player sports:sport sports:Basketball)
      (?player general:nationality geog:USA)
      (?player sports:playsIn ?team)
      (?team rdf:type sports:SportsTeam)
      (?team geog:locatedIn geog:Catalonya)
```

would yield the query keywords “player”, “basketball”, “USA”, “team”, “Catalonia”.

In sum, our method improves keyword-based search (actually outperforms it, as is shown in the next section) when the relevant information is available in the KB, and relies on keyword-based search otherwise.

5 Early Experiments

We have tested our system with a document base taken from an online newspaper archive [2]. For this application, the document class hierarchy includes *News* (subclass of *TextDocument*), *Photograph* and *CustomGraphic* (subclasses of *MediaDocument*). Only one classification taxonomy is used, based on the IPTC Subject Reference System, with which all documents and domain classes are classified, as explained in Section 3.1. Our current implementation is compatible with both RDF and OWL.

Building appropriate domain ontologies and a complete KB for a newspaper archive is an enormous undertaking, or would need very advanced semi-automatic knowledge extraction techniques not yet available in current state of the art. However,

as stated in previous sections, our system tolerates incomplete ontologies and KBs. We have built three reduced domain ontologies for testing purposes, corresponding to the Culture, Economy, and Sports domains, with classes such as *Artist*, *Painter*, *Monument*, *Company*, *Bank*, *Sportsman*, *SportsTeam*, *Stadium*, etc., and a few instances of each class. These ontologies were built by reading 200 news articles, and defining classes and instances by hand for concepts found in the documents. In total, 143 domain classes and 1,144 instances were created. We have also manually set labels and keywords for concept classes and instances. Then we have run the automatic annotation and weighting algorithm over a subset of the archive comprising 2,039 news articles, generating 3,471 annotations, of which 349 were manually created.

Once the KB was built, we tested the retrieval algorithm with some examples, and compared it to a keyword-only search, using the Jakarta Lucene library.² We report next the observed results in four examples, showing different levels of performance of our method in different cases. The metrics are based on a manual ranking of all documents for each query, on a scale from 0 to 5. In the experiments, all the query variables were given a weight of 1. The measurements are subjective and limited, yet indicative of the degree of improvement that can be expected, and in what cases, with respect to a keyword-based engine. The retrieval times are too low to draw any significant observation regarding efficiency. The results are shown in Fig.2.

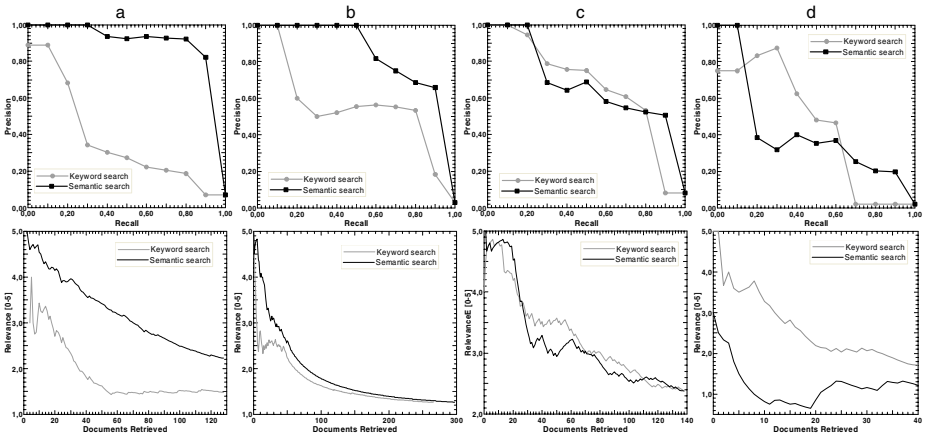


Fig. 2. Evaluation of ontology-based search (combined with keyword-based) against keyword-based only. The performance of both algorithms are shown for four different queries a, b, c, and d. The graphics on top show the precision vs. recall figures (as defined in e.g. [16]), and the graphics below show the average relevance at different document cutoff values, for each query

Query a. “News about players from USA playing in basketball teams of Catalonia.” In this example the semantic retrieval algorithm outperforms keyword-based search

² <http://lucene.apache.org>

because the KB contains many instances of basketball players and teams, some of which match the query. Keyword-based search only recognizes a document as relevant if it contains words like “player”, “USA”, “Catalonia”, whereas the semantic search retrieves news about players and teams as soon as the name of the player or the team are mentioned in the document.

These are typical results when a search query involves a region of the ontology with some degree of completeness in terms of instances and annotations. These cases yield a high precision up to almost maximum recall. On the other hand, the relevance graph shows that here the semantic search gives high ranks to the relevant documents. For instance, the top 20 retrieved documents have a mean relevance value of 4.2 upon 5, versus 2.7 in the keyword search.

However, the KB does not contain *all* teams and players, which explains the collapse of the precision at 100% recall. If more instances were created, precision values would stand at high levels for all the recall values.

Query b. “News about sports team presidents.”

In this example, the ontology KB has only a few instances of sports team presidents, so not all documents relevant to the query are annotated. This causes precision to drop to lower values when recall increases. Although the total recall of semantic search is low, it still has a good precision for the top-ranked documents, which are the few ones annotated with instances in the KB. A few more documents where semantic search alone fails are still given a high ranking thanks to the combination with keyword-search, which shows here a comparable behavior to example a.

Query c. “News about basketball players.”

In this case the performance of the two algorithms is similar. For this example, we have intentionally removed most instances of players from the KB, leaving a relatively low number. Moreover, we have removed all lexical variants in the *label* and *keyword* properties of player instances, except the player’s surname. As a consequence, many annotations are missing. Under these conditions, the semantic model alone performs much worse than keyword-based search. However, the combined search yields a similar final behavior to keyword-based search.

Query d. “News about the European Union.”

This example shows a case where our method fails. The KB contains an instance for the European Union, with all the possible syntactic variants (in Spanish “UE”, “U.E.”, etc.). The problem is that many Catalan sports teams have the word “UE” (acronym for “Sports Union” in Catalan) in their names. If the KB contained these teams, the disambiguation algorithm would solve the problem by favoring the sports interpretation, whenever appropriate, because “UE” is part of a longer matching string (the team name). In other examples where the labels could be totally coincident, the system would use the classification of news and instances as context information for disambiguation. But because many such teams are missing in the KB, the automatic annotator incorrectly annotates the sports news with the European Union concept, and the retriever returns them. So far, the keyword-based search behaves similarly. But

the semantic ranking places these wrong documents in a top position, whereas the keyword-based model does not rank them particularly higher than the correct documents.

It can be seen that it is the automatic annotator, and not the retrieval system, which is failing here in the absence of the appropriate instances needed to solve ambiguities. One way to reduce the negative impact of incorrect annotations would be to introduce a factor in the automatic weighting algorithm that accounts for the proximity of the respective classifications of the documents and the instances. In this example, although it is difficult to avoid annotating with the European Union concept the news about Catalan teams whose name contains “UE” (in fact, some sports news could properly mention the EU), at least the weight of the annotation would be reduced because the classifications (*Geography* and *Politics* vs. *Sports*) do not match. Testing this and other possible improvements to the automatic annotation strategies are one of our planned tasks for the immediate future.

6 Discussion

The added value of semantic information retrieval with respect to traditional keyword-based retrieval, as envisioned in our approach, relies on the additional explicit information – type, structure, relations, classification, and rules, about the concepts referenced in the documents, represented in an ontology-based KB, as opposed to classic flat keyword-based indices. Semantic search introduces an additional step with respect to classic information retrieval models: instead of a simple keyword index lookup, the semantic search system processes a semantic query against the KB, which returns a set of instances. This can be seen as a form of query expansion, where the set of instances represent a new set of query terms, leading to higher recall values. Further implicit query expansion is achieved by inference rules, and exploiting class hierarchies. The rich concept descriptions in the KB provide useful information for disambiguating the meaning of documents.

In summary, our proposal achieves the following improvements with respect to keyword-based search:

- Better recall when querying for class instances. For example, querying for “presidents of the Spanish government” would return documents that mention *José Luis Rodríguez Zapatero* and other former presidents, even if the words “president”, “Spanish” and “government” are not present in the documents.
- Better precision by using structured semantic queries. Structured queries allow expressing more precise information needs, leading to more accurate answers. For instance, in a keyword-based system, it is not possible to distinguish a query for USA players in Catalan basket teams vs. Catalan players in USA teams, which is possible with a semantic query.
- Better precision by using query weights. Variables with low weights are only used to impose conditions on the variables which really matter. For example, the user can search for news about USA players in Catalan teams, regardless of whether the news mention the team at all, or the nationality of the player.

- Better recall by using class hierarchies and rules. For example, a query for *WaterSports* in Spain would return results in *ScubaDiving*, *Windsurf*, and other subclasses, in *Cádiz*, *Málaga*, *Almería*, and other Spanish locations (by transitivity of *locatedIn*).
- Better precision by reducing polysemic ambiguities using instance labels and classifications of concepts and documents.

As explained and shown along this paper, the degree of improvement of our semantic retrieval model depends on the completeness and quality of the ontology, the KB, and the concept labels. For the sake of robustness, the system resorts to keyword-based search when the KB returns poor results.

The combination of keyword ranking and semantic ranking is tricky. We have observed that occasionally a good semantic ranking score is spoiled by a low keyword-based value. A simple solution would be to set a minimum threshold for the keyword-based score to be counted. Anyhow, these cases, albeit infrequent, suggest that more sophisticated methods than the linear combination of both values should be researched to improve our initial results.

7 Conclusion

The research presented here started as a continuation of our previous work on the construction, exploitation, and maintenance of domain-specific KBs using Semantic Web technologies [1,2]. While some basic semantic search facilities were included in these prior proposals, room for improvement was acknowledged, because the level of semantic detail was insufficient, since it was essentially based on types of documents and taxonomic classifications. The aim of our current work is to provide better search capabilities which yield a qualitative improvement over keyword-based full-text search, by introducing and exploiting finer-grained domain ontologies.

Our approach can be seen as an evolution of the classic vector-space model, where keyword-based indices are replaced by an ontology-based KB, and a semi-automatic document annotation and weighting procedure is the equivalent of the keyword extraction and indexing process. We show that it is possible to develop a consistent ranking algorithm on this basis, yielding measurable improvements with respect to keyword-based search, subject to the quality and critical mass of metadata. Our proposal is an adaptation of the vector-based ranking model that takes advantage of an ontology-based knowledge representation.

Our proposal inherits all the well-known problems of building and sharing well-defined ontologies, populating knowledge bases, and mapping keywords to concepts. Recent research on these areas is yielding promising results [5,11]. It is our aim to provide a consistent model by which any advancement on these problems is played to the benefit of semantic search improvements.

Further experimentation, larger KBs, and larger document sets are needed to test and improve our model. For instance, our annotation weighting scheme is not taking advantage yet of the different relevance of structured document fields (e.g. title is more important than body). Annotating documents with statements, besides instances,

is another interesting possibility to experiment with. Also, we are currently extending our model with a profile of user interests for personalised search [6].

Acknowledgements

This research was supported by the European Commission under contract FP6-001765 aceMedia. The expressed content is the view of the authors but not necessarily the view of the aceMedia project as a whole. The authors would like to thank the reviewers for their detailed, accurate and helpful comments.

References

1. Castells, P., Foncillas, B., Lara, R., Rico, M., Alonso, J. L.: Semantic Web Technologies for Economic and Financial Information Management. In: Davies, J., Fensel, D., Bussler, C., Studer, R. (eds.): *The Semantic Web: Research and Applications – 1st European Semantic Web Symposium (ESWS 2004)*. LNCS, Vol. 3053. Springer Verlag, Berlin Heidelberg New York (2004) 473-487
2. Castells, P., Perdrix, F., Pulido, E., Rico, M., Benjamins, V. R., Contreras, J., Lorés, J.: Neptuno: Semantic Web Technologies for a Digital Newspaper Archive. In: Davies, J. et al (eds.): *The Semantic Web: Research and Applications – 1st European Semantic Web Symposium (ESWS 2004)*. LNCS, Vol. 3053. Springer Verlag, Berlin Heidelberg New York (2004) 445-458
3. Christophides, V. et al: Optimizing taxonomic semantic web queries using labeling schemes. *Journal of Web Semantics* 1, Issue 2, Elsevier (2003) 207-228
4. Contreras, J., Benjamins, V. R., Blázquez, M., Losada, S., et al: A Semantic Portal for the International Affairs Sector. In: Motta, E., Shadbolt, N., Stutt, A., Gibbins, N. (eds.): *Engineering Knowledge in the Age of the Semantic Web – 14th Intl. Conference on Knowledge Engineering and Knowledge Management (EKAW 2004)*. Lecture Notes in Computer Science, Vol. 3257. Springer Verlag, Berlin Heidelberg New York (2004) 203-215
5. Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R. et al: A Case for Automated Large Scale Semantic Annotation. *Journal of Web Semantics* 1, Issue 1, Elsevier (2003) 115-132
6. Gauch, S., Chaffee, J., and Pretschner, A.: Ontology-based personalized search and browsing. *Web Intelligence and Agent System* 1, Issue 3-4 (2003) 219-234
7. Guarino, N., Masolo, C., Vetere, G.: OntoSeek: Content-Based Access to the Web. *IEEE Intelligent Systems* 14, Issue 3 (1999) 70-80
8. Guha, R. V., McCool, R., Miller, E.: Semantic search. In *Proc. of the 12th Intl. World Wide Web Conference (WWW 2003)*, Budapest, Hungary, (2003) 700-709
9. Handschuh, S., Staab, S., and Ciravegna, F.: S-cream – Semi-automatic Creation of Metadata. In: A. Gómez-Pérez, V. Richard Benjamins (eds.): *13th Intl. Conf. on Knowledge Engineering and Knowledge Management – Ontologies and the Semantic Web (EKAW'02)*. LNCS, Vol. 2473. Springer Verlag, Berlin Heidelberg New York (2002) 358-372
10. Järvelin, K., Kekäläinen, J., and Niemi, T.: ExpansionTool: Concept-based query expansion and construction. *Information Retrieval* 4, Issue 3-4 (2001) 231-255
11. Kiryakov, A., Popov, B., Terziev, I., Manov, D., Ognyanoff, D.: Semantic Annotation, Indexing, and Retrieval. *Journal of Web Semantics* 2, Issue 1, Elsevier (2004) 47-49

12. Maedche, A., Staab, S., Stojanovic, N., Studer, R., Sure, Y.: *SEmantic portAL: The SEAL Approach*. In: Fensel, D., Hendler, J. A., Lieberman, H., Wahlster, W. (eds.): *Spinning the Semantic Web*. MIT Press, Cambridge London (2003) 317-359
13. Mayfield, J., and Finin, T.: *Information retrieval on the Semantic Web: Integrating inference and retrieval*. In: *Workshop on the Semantic Web at the 26th Intl. ACM SIGIR Conference on Research and Development in Information Retrieval*, Toronto, Canada (2003)
14. Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., Kirilov, A.: *KIM – A Semantic Platform for Information Extaction and Retrieval*. *Journal of Natural Language Engineering* 10, Issue 3-4, Cambridge University Press (2004) 375-392
15. Rocha, C., Schwabe, D., de Aragão, M. P.: *A Hybrid Approach for Searching in the Semantic Web*. In *Proc. of Intl. World Wide Web Conf. (WWW 2004)*, NY (2004) 374-383
16. Salton, G. and McGill, M. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York (1983)
17. Sheth, A., Bertram, C., Avant, D., Hammond, B., Kochut, K., and Warke, Y.: *Managing Semantic Content for the Web*. *IEEE Internet Computing* 6, Issue 4 (2002) 80-87
18. Stojanovic, N.: *On Analysing Query Ambiguity for Query Refinement: The Librarian Agent Approach*. In: Song, I.-Y.; Liddle, S.W.; Ling, T.W.; Scheuermann, P. (eds.): *Conceptual Modeling – ER 2003, 22nd Intl. Conf. on Conceptual Modeling*. LNCS, Vol. 2813. Springer Verlag, Berlin Heidelberg New York (2003) 490-505
19. Stojanovic, N., Studer, R., and Stojanovic, L.: *An Approach for the Ranking of Query Results in the Semantic Web*. In: Fensel, D., Sycara, K. P., Mylopoulos, J. (eds.): *The Semantic Web – ISWC 2003, 2nd Intl. Semantic Web Conf. Lecture Notes in Computer Science*, Vol. 2870. Springer Verlag, Berlin Heidelberg New York (2003) 500-516

Knowledge Sharing by Information Retrieval in the Semantic Web

Neyir Sevilmis¹, André Stork^{1,†}, Tim Smithers³, Jorge Posada³,
Massimiliano Pianciamore², Rui Castro⁴, Ivan Jimenez³, Gorka Marcos³,
Marco Mauri², Paolo Selvini², Bruno Thelen⁵, and Vincenzo Zecchino⁶

¹ Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt

² CEFRIEL, Milan

³ VICOMTech, Donostia / San Sebastián

⁴ Centro de Computação Gráfica, Guimarães

⁵ Schenck Pegasus GmbH, Darmstadt

⁶ Italdesign – Giugiaro SpA, Moncalieri (Torino)

[†]ist-wide@igd.fhg.de

Abstract. Effective and efficient information retrieval, knowledge sharing and combining has become an essential part of more and more professional tasks and work flows in different kinds of projects. Our aim is to investigate the use of emerging Semantic Web technologies, tools, and standards in the support of effective information retrieval in real multi-disciplinary activities, such as innovative product design. This paper presents an approach to knowledge sharing and information support that has been developed and adopted, the information system architecture that is being developed to test both this approach, and the Semantic Web techniques that are used in its implementation, some early results, and a discussion of related work in information systems and Semantic Web techniques and tools.

1 Introduction

Unquestionable, the internet is developing towards the Semantic Web. Semantic Web technology promises to improve on one of the main usages of the internet: knowledge exchange via information retrieval. But how shall a Semantic Web-based system look like to best support different users in retrieving information and knowledge generated by others in the Semantic Web? This was the deriving question, motivating us to design, implement, and test an approach that explores Semantic Web technologies for improving on today's limited search and retrieval possibilities on the internet. The developments have been done in the context of the product development process within the car industry. The domain knowledge, use and test cases have been developed with real data and real users from two companies, namely ItalDesign Giugiaro SpA and Schenck Pegasus GmbH.

[†] Corresponding Author.

The main requirements imposed by the scenario - typical for the Semantic Web - have been to support different kinds of users accessing various heterogeneous information sources in a semantic way. These requirements entailed a number of secondary questions: How to support users in developing complex queries in a terminology that each type of user is familiar with? How to process these queries so they can be 'understood' by different information sources? How to deal with complex queries? What to do with the results from heterogeneous sources? How to present the results in a meaningful way? And from the technological point of view: What kinds of Semantic Web technologies are best suited to answer those questions? How to they perform with real world ontologies and data? Are the current tools appropriate for end-users to model their domain?

This paper tries to answer to those questions by introducing an architecture and describing its implementation based on the experience we gathered in two rounds of user testing. Although there have been comparable attempts to approach the information management issue in heterogeneous environments [15, 16], to our knowledge none of them comprises both: the full scope of the approach that we introduce here and the use of latest Semantic Web technology and tools, e.g. OWL and OWL reasoners such as RACER.

The paper is structured as follows: first the basic concepts are described. Secondly, the role of the components of the architecture is overviewed. The main part of the paper is dedicated to the semantic information retrieval process, made up by seven steps: the graphical interactive user query development, semantic query processing using a domain ontology, planning distributed query execution on heterogeneous information sources, the retrieval from Semantic Information Sources, the result collection and adaptation, the result preparation process of semantically enriched results, and finally the domain knowledge enhanced result presentation.

Afterwards, based on our experience we point out further research needs from a holistic point of view, here we see another contribution of the current paper to the Semantic Web community.

2 Concepts

In this section, we introduce the basic concepts and items/entities involved in our approach. Some of them are related with the architecture, others with the way we are modelling the domain and how we distinguish amongst the role of the components of the system whereas other are definitions useful in describing the various kinds of information and information structures involved.

2.1 3-Layered Architecture

The basic assumption that drove the conception of the architecture is that the near future of the Semantic Web will be determined by a mixture of information sources with various levels of semantic richness in the internet and intranet. Those information sources basically form the lowest level of the architecture – the Content Level (see Figure 1).

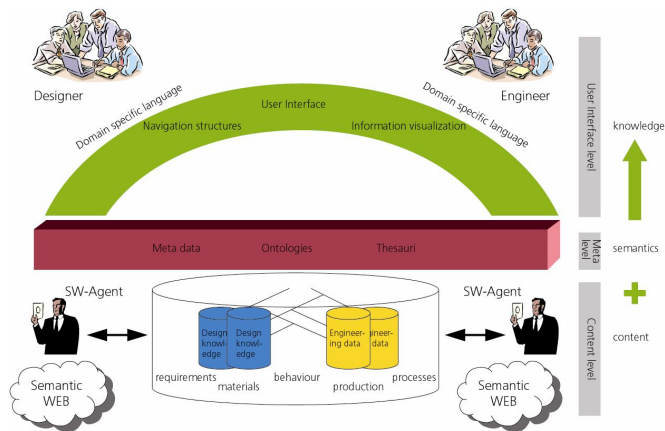


Fig. 1. 3-layered architecture

The second assumption is that domain knowledge and process knowledge shall reside in central components doing the semantic processing of queries and results – the Meta Level.

And thirdly, users of different disciplines along the domain - in our case stylists, designers, engineers involved in the product design process of cars – shall access information and share knowledge via different instances of a User Interface that communicates with the Meta Level and presents retrieved results.

These assumptions immediately entail that the modelling of information sources and the modelling of the domain ontology is performed independently, giving rise to the need of terminology mapping and query adaptation.

2.2 Semantic Information Sources

As already mentioned, we believe that for the near future the developing Semantic Web will provide information sources with different levels of semantic richness.

But, how an ideal Semantic Information Source shall look like? And what kind of functionality shall it contain or provide? Plus what kind of information shall a Semantic Information Source return to enable semantic reasoning?

From a data-centric view, for us a Semantic Information Source contains the following layers: Schema, Annotation and Content.

On the top-most layer a conceptual data schema describes the content that is stored by abstract entities and their relations. This schema appears as a low-level ontology: the provider ontology of the Semantic Information Source. In the middle layer the abstract entities are instantiated in interlinked metadata annotation objects which, in turn, refer to the actual content items on the bottom layer. The content can be of many kinds, e.g. multi-media documents such as pictures, text, 3d models. These content items (instances) are what the user is interested in to retrieve. All the metadata on the schema and annotation level are being used to semantically describe the content and allow for precise and accurate retrieval. The metadata objects in the middle layer

appear as instances of classes while the content items appear as references to URL-accessible stores or lower-level database access components.

Note that a Semantic Information Source does not contain a full domain ontology nor does it represent a knowledge base from our point of view. Instead, the Semantic Information Source models the ‘aboutness’ of documents/information contained in the lowest level. The schema only contains what is needed to appropriately describe the kinds of documents/information contained in the lowest level and about which real world concepts they are talking about.

From a functional point of view a Semantic Information Source should be able to process a query posed in a standard format – we are using RQL [3] in the current implementation. In this context processing comprises, mapping it in a syntactic and semantic way.

Furthermore, it shall provide the results in a standard form, preferably enriched by semantic information/context for further semantic processing, e.g. reasoning on the results for filtering and ranking them before showing them to the user. We are using the W3C suggestion for RDF result representation returning not only the results but also structured context information (for details see below) [12].

2.3 User Queries and System Queries

We aim at allowing the user to formulate his query in an as natural as possible way (ease-of-use) but also as precise as possible (quality of results).

To face the users with standard query languages such as RQL, SQL, etc. is certainly not the best approach in terms of usability. Thus we decided to distinguish between User Queries (UQ) and System Queries (SQ). User Queries are input by the user whereas System Queries are derived from User Queries using domain knowledge by the Meta Level.

3 System Architecture

We designed and implemented a distributed system architecture which is divided in to four basic blocks: the User Interface, the Meta Level, the Agency and the Content Level. Each of these is implemented as independent subsystems, the Agency - a multi agent system - is used to “glue” them together, as shown in Figure 2.

3.1 Role of the User Interface

The User Interface (UI) provides a graphic front end to the user and supports the incremental development of user queries in an alphanumeric or graphic way. By easy to use drag and drop operations the user can successively build up his query. This interactive and incremental query development process is supported by the Meta Level and the domain knowledge contained in the Meta Level. Furthermore, the UI presents the returned results and their relationships (semantics) in a graph-based structure. This graph structure can be navigated by the user in order to explore the returned results and their metadata. Based on the returned graph structure and the corresponding metadata the current user query can be refined or a new one can be developed.

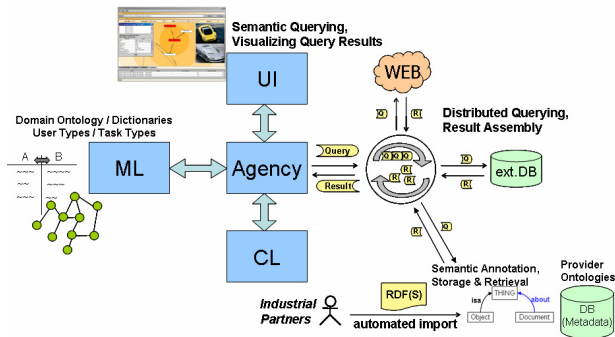


Fig. 2. Distributed system architecture

3.2 Role of the ML

The main purpose of the ML is the semantic processing of user queries into system queries and the semantic processing of the returned results. To do this, the ML uses a domain ontology (for car design), together with a Task Type ontology (knowledge about the different tasks carried out in the domain), User Type ontology (knowledge about the profile of the different users in the domain), and dictionaries of description terms and User Type terms. All these different kinds of knowledge are used to produce the system queries that are then passed to the Agency. The returned results undergo a similar semantic processing as the user queries. They are semantically processed in order to associate them with the appropriate concepts in the domain ontology and finally display them in a meaningful graph structure by the UI.

3.3 Role of the Agency

The Agency subsystem glues the other components together by allowing their inter-operation. It further identifies and locates information sources in the Content Level to which the system queries can be sent to produce effective returns, by addressing and managing issues like distribution and heterogeneity in them. The Agency also provides the system's gateway to the Web, which is also considered part of the Content Level. Essentially, Web sites and Web search engines are treated as weakly structured information sources. Besides the planning and execution of queries, the Agency's responsibilities are also to collect and to transform the results of the heterogeneous information sources into a common result format on which the ML is performing semantic processing and reasoning.

3.4 Role of the CL

The Content Level (CL) consists of different information sources (RDF sources [5, 6], ASAM/ODS [4], relational databases and the web) that vary in their semantic richness. Since those information sources might have a different structure than the ML domain ontology, one of the tasks of the CL is to adapt the system queries to the

query language understood by the information sources (syntactically, terminologically, and structurally). The main purpose of the CL is to answer precisely the system queries in a quick way.

4 The Process of Information Retrieval

We understand the information retrieval process as a kind of design task by firstly recognizing the difference between user stating needs and forming well specified requirements, and secondly properly supporting the incremental development of a complete and consistent requirements specification (search specification, in this case), and the re-use of the knowledge generated in this (sub)process to effectively support the subsequent steps in the process that concludes in a useful set of search results. According to the system architecture presented in Figure 2, the process of information retrieval is composed of seven steps that are presented in Figure 3.

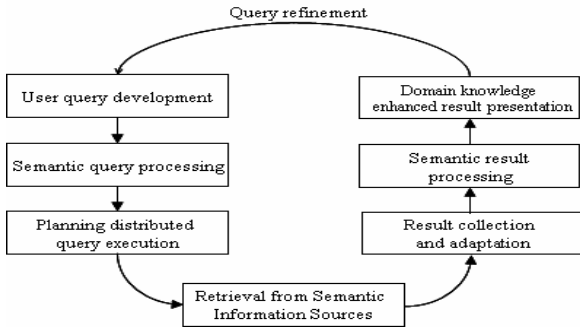


Fig. 3. Seven steps of the information retrieval process

Based on the experienced gathered when developing the system, we are convinced that these are – at least – the steps needed for improved information retrieval on the Semantic Web. According to this, each step of the information retrieval process will be explained in the following sections.

4.1 User Query Development

The UI provides text-based and graphic-based support to specify queries. The graphical version uses domain knowledge (from the ML, suitably selected and presented using the User Type and Task Type specifications) to offer the user a “drag and drop” way of building correct queries. Users can use a combination of both text input and graphical selection to form a query as shown in Figure 4. This is then checked against a BNF grammar [2], for correctness, and passed to the ML. The ML then processes the user query, using its domain ontology, User Type Dictionary, and personal user dictionary, to discover what other concepts it has that are related to the concepts in the UQ.

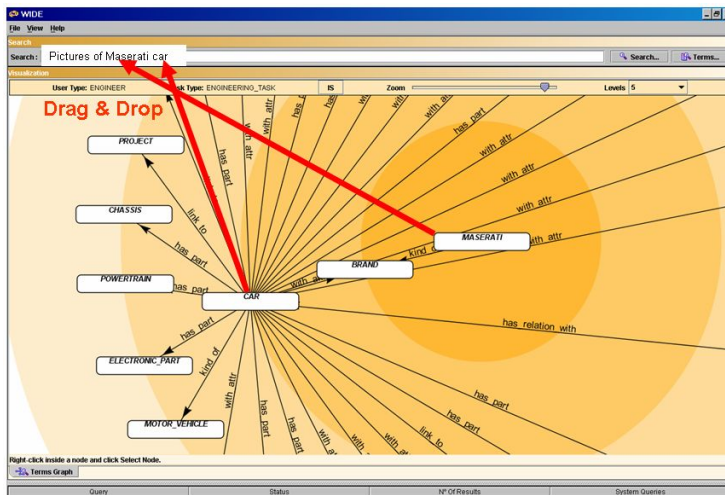


Fig. 4. Graphical interactive user query development

These further domain ontology concepts, and the ways they are related to the user query concepts, are then returned to the UI as an ontology fragment that represents the user query and its immediate conceptual context, where it is graphically presented to the user. This ontology fragment, or query structure, as it is called, supports further navigation of the concepts and properties present, allowing the user to further extend the fragment by including further concepts along selected relations. In this way, a user is able to see how the system understands his or her query, and is supported in further exploring around it, to see how it might be changed, adapted, or extended, to form a more effective query: more precise and/or more complete.

4.2 Semantic Query Processing

When the ML receives the user query from the UI via the Agency, this query is well formed input since it has been successfully parsed by the UI. Thus, this is the starting point of the semantic query processing.

The first task that must be done is to check again the stylized input, but at this occasion from the semantic point of view. The ML uses its knowledge about the domain, but also its knowledge of the User Type and the task context the query is being expressed on, to find out more information about the query.

First of all, the ML uses the dictionaries together with the user profile in order to translate the query into the internal terminology. Next, it tries to find out which of the words that have been classified by the BNF parser as “terms” are known in the domain and which not.

Once the ML has translated the user query into internal terminology, the parser generates an AST (Abstract Syntax Tree) which includes useful information that allows the ML to know the terms within the context of the BNF grammar. For instance,

the ML automatically knows that “OF” is a connector and that “OR” is a logical binary operator that links two items. Thus, the ML is able to classify the relevance of each term depending on its BNF grammar nature. Once the ML acquired this knowledge, it can go through this tree and focusing on the known terms in order to try to find out the existing relationships among them. This is possible due to an inference process of the ML domain ontology. For instance, this process allows the ML to relate two concepts basing on subsumption or indirect relationships.

Once this process has been carried out, the ML has a graph view of the user input, which has been enriched with intermediate concepts and/or inter-concept relationships.

After this first query has been built, the ML, basing on the task context the query is related to, tries to create complementary queries by the expansion of some concepts. For instance, if the user is looking for emission standards in Europe, and the task context is wide enough in order to cover also the test cycles, the system will generate another system query that will retrieve test cycles in Europe.

Once the ML has inferred the sub-domain related to the task context and has created several queries, these queries are formatted in RQL, and then passed to the Agency.

For example, a concept designer might start by asking for

user query 1: Photographs of Maserati cars

In response, the UI (with support from the ML) would show that it understands *photograph* to be a kind of picture, where *drawing*, *image*, and *sketch* are other kinds of *picture* concepts. As a result, the user might then change the query to

user query 2: Pictures of Maserati cars

to be more inclusive of other possible kinds of pictures. This is then transformed by the ML in to the following internal form

user query 3: PICTURE ABOUT MASERATI CAR

where *picture* is a known document type, *about* is the term used to connect the document type to the concept, and *Maserati* and *car* are understood as two terms forming an attribute value qualifying phrase. The ML then expands this user query, based upon its knowledge that *Maserati* is the name of an individual of the concept *brand*, and that *brand* is defined as the range of a *has_property*, whose domain is *car*. The resulting expanded system query thus expressed as an RQL query then looks like:

```
SELECT pt, mc
FROM {pt:$pt} @p {mc:$mc},{rc1} @w_al{c1:$c1},
    {rc2} @w_vl {vl:Literal}
WHERE @p = "has_info_about" AND
    ($p1 = "PICTURE") AND
    $mc = "CAR" AND
    mc = rc1 AND
    @w_al = "with_attr" AND
    $c1 = "BRAND" AND
    c1 = rc2 AND
    @w_vl = "with_value" AND
    vl = "MASERATI"
```


The RQL system query says: select all the pictures and all the cars where picture is a presentation type that has info about the cars and the cars have an attribute named brand, whose value is Maserati.

4.3 Planning Distributed Query Execution

As soon as the Agency subsystem receives the system queries produced by the ML, it sends them to the various information sources, in order to proceed with the search process. The decision about how to distribute the various queries over the available sources is referred to as query execution planning and can be carried out by analyzing the structure of each query. Concerning the structure of a system query expressed in the RQL language, it can easily be seen that the FROM clause can be interpreted as the navigation path inside a proper ontology of concepts bound to one another by means of suitable relationships. As such, it appears evident that one single source might not be able to address the whole system query in its entirety and therefore it is forced to focus on a sub part of it. When this happens, the system query being processed is called a *complex query*, since it can be both logically and physically split up in as many sub-queries as there are sub parts entirely addressable by a single source. We call these latter ones *simple queries*. Using an internal representation in which system queries are made up of conjunctive predicates (reflecting the RDF triples embedded in the RQL queries) the Agency broadcasts the complete sequence of predicates for each query to all the information sources available. The management of the interdependencies between the simple queries is left to the information sources themselves, which may or may not be enabled to do that. If not, they simply consider each complex query as a simple query and try to execute it entirely, possibly returning an empty result in case they cannot understand it or have no data for it. Otherwise, each source decides which simple queries it can address, by looking at the set of mapping rules it has, in order to bridge the semantic and structure gaps between the conceptualization implied in the queries and their own (see below). If no rules exist that have a match with a simple query, then that query is not considered for local execution and the source waits for its results to come from other sources. This is done by means of mobile software agents carrying queries and results back and forth between the core system and the remote sources. Mobile agents are particularly well suited for repetitive tasks like information retrieval, as they realize an asynchronous computing model that adapts well to varying network conditions. In particular, mobile agents improve the fault-tolerance with respect to network communication problems by providing a disconnected operation model: interactions and communication occur at well defined instants, which also generate a more efficient, burst-like, kind of network traffic.

In order to contact the remote sites where to apply the various queries, agents may choose two options: either they move there or they spawn children agents, which are sent on their behalf. Which option to choose depends on factors like the current network traffic, the dimension of the search job in terms of how big is the query and how many information sources must be contacted. Currently, agents choose the first option (move) when there is only one information source to be contacted, otherwise they parallelize the process by choosing the second alternative.

Once all the results are produced, they are assembled and sent back to the system, for final processing and visualization. Result assembling is realized either by pure concatenation of the return fragments obtained, in case the source is not enabled to manage the complex queries. Otherwise results undergo an interdependency-solving process by which join conditions specified in the complex query and binding its constituent simple queries are taken into account.

Query execution at the various information sources requires that all the heterogeneity issues be dealt with, at the syntactic, structural and semantic levels: information pieces, in fact, are likely to be distributed and partially replicated on different repositories, often built using different technologies and modeling techniques. To overcome the syntactic differences a proper algorithm was devised, making use of thesauri on both (on the system and at each source) sides to find matchings and rewrite terms. As for structural differences, they are dealt with by identifying recurrent structural patterns in queries that can be easily rewritten into others. Semantic heterogeneity, on the other hand, has to do with the interpretation given to the stored data along with any relationship binding them. For a machine to be able to communicate and understand what a single piece of information is about, a model describing the semantics of the information has to be provided. Such model is represented by the Provider Ontologies, that are either already available, for semantically rich sources, or can otherwise be created. Semantic mapping, in this respect, has to do with the rewriting of queries, by employing rules associating whole fragments in the modeling of the domain ontology to fragments suitable for the various provider ontologies.

The information sources that we addressed belong to one out of three kinds: an object-relational one served by an ASAM/ODS server, the Google search engine (as an example of an unstructured source), which is accessed by using its Web Services interface and a set of plain relational databases, each one provided with an RDF(S) (provider) ontology that describes its contents and that has a mapping to the (central) domain ontology. To manage each query, execute it and apply the necessary transformations, a *provider agent* was created, whose tasks are to:

- receive and manage the software agents
- transform the incoming RQL query and adapt it to the internal language of the source
- manage the results by formatting them in a suitable and common format to be sent back to the system.

In particular, the latter two points depend from the specific details of the underlying source, even though the interface provided to the outside world should remain unchanged.

4.4 Retrieval from Semantic Information Sources

Our Semantic Information Sources, as described in section 2.2, provide a RQL/RDF(S) interface. The input to such an information source is a RQL query which - from the schema layer point of view - asks for instances of concepts on which there are imposed conditions (in the WHERE clause of RQL queries). The result that is being produced for a system query is a RDF graph fragment which contains RDF

instances required answering the RQL query together with their attributes and direct references to other RDF instances. Within our Semantic Information Source the Sesame RQL engine [9] is used to access the RDF(S) store.

Each RQL query undergoes a 2-step query process. In the first step only the RDF resources are returned that form the core answer to the system query. In the second step, the returned RDF resources are semantically enriched by their metadata to be presented and visualized by the UI. Concerning the metadata, we distinguish between properties and context information of a returned RDF instance. The properties are describing the returned RDF instance itself and shall provide the user additional information. For example, an instance of the concept *car* has the properties *brand* and *segment*. It is possible that in the results several concepts (context information) are present that are not strictly related to what originally contained in the system query coming from the ML. The context information is being described by the neighbouring concepts, because we think that they provide meaningful information to reason on the results by the ML. The neighbouring concepts are the direct sub-classes, direct super-classes and further direct related concepts.

Taking into account the neighbouring concepts the ML can easily find out in which context instances of a concept are returned. All this information are encoded in the RDF result fragments in order to provide both the user additional information that he/she can use to refine the current query in a subsequent search process (see section 4.7) and the ML in order to allow reasoning on the returned results. Thus, the RDF result fragment is produced in combination of RQL resource querying and navigation on the RDF(S) store.

The answering of RQL queries takes the RDF model theory [13] into account, which includes some basic inference over inheritance hierarchies as well as domain and range constraints. In addition, we extended the inference rule engine built in Sesame [9] that works as a production system generating inferred RDF triple facts from explicitly stated ones according to rules and axioms expressed in a proprietary XML format. In particular the transitivity of user defined properties is realized in this way.

4.5 Result Collection and Adaptation

Presenting the retrieved results in an efficient way for the users to access them is one of the basis of any good searching activities. The returns produces are assembled into a document with well-known structure, which follows a proposal discussed at the W3C consortium as a standard for query result formatting [12]. This format is made up of very simple yet efficient structure binding variables with their values.

Along with raw data, each result also contains some special meta-information, helpful to better organize, filter and sort the information before presentation to the user, in a process that also encompasses some rewriting steps much like what happens for the queries as explained above. The additional information is the following:

- the source, which each return comes from
- the context of each return (i.e., similar or related concepts)

- an additional ontology fragment, in case the provider ontology is richer than the domain one and a direct mapping cannot be established.
- the relationship with what was originally asked for in the system query (as heterogeneity could cause this reference to be lost)

The formatting process, besides endowing every result with the same structure, also allows for a much simpler filtering procedure, whose purpose is to drop duplicate answers and evident useless information, evaluated on the basis of a syntactic analysis over the results. The final step in the search process is the logical opposite of the query adaptation and negotiation described above. At this stage all the results are expressed by a common structure and format, but the terms used refer to the related provider ontologies. Hence a final mapping of the results to comply with the system ontology must take place. This mapping can be seen as the counterpart of the one already described above. During this activity, each variable of the result is also tagged using a special prefix to emphasize whether or not that term is known to the system ontology, in order for the Meta Level to take that into account for the subsequent phase.

4.6 Semantic Result Processing

Once the Meta Level has sent a set of system queries to the Agency, it assumes that the search process has been launched. At that moment, the main role of the Meta Level is to retrieve the returns of each one of the queries, evaluate and rank them and prepare the graph of the results that will be presented to the user. The Agency sends the results for each of the system queries produced immediately back to the ML, without waiting to have completed them all to it. As it gets them, the ML is responsible for the following actions: grouping the returns, ranking the returns, and constructing the graphical visualization.

The ML, independently from the information sources the returns come from, groups them depending on the system query they belong to. This is an important process since the visualization of the results will be different depending on the semantics of each system query.

The ML evaluates the returns using the information contained in the results, the terms included in the system query the return belongs to, the concepts involved in the task context the query is related to and the concepts appearing in the user query. The idea is to measure the distance not only to the original query but also to the task context where the query is located. If the returned results contain instances of concepts that are

- (a) not specified in the system query and
- (b) unknown in the ML ontology,

the results are considered as unclassified, and are graphically shown in a different node of the graph.

Using as a start point the graphs that represent each one of the system queries that the returns belong to, the ML builds a result visualization graph where the final results

will be attached. This graph shows not only the concepts involved in the results, but also the relationships among them.

4.7 Domain Knowledge Enhanced Result Presentation

After the semantic processing of the results the ML attaches each set of results to one of the concepts of the graph. Beside this, the ML provides each bunch of results of each system query with a semantic path, which shows the set of concepts each result is related to. Thus, although each result appears under a concrete concept, the user is able to see the semantic contextualization of the result. This is done using the graph information of the system query to which the result belongs.

The query refinement is the subsequent browsing of the results presentation that supports further exploration of how the search specification might be useful further developed to better meet his or her information needs. In this way, from the user perspective, the query refinement effectively merges with the query development (see Figure 4) one to form what can be understood as a kind of design process.

5 Technology Used and Problems Encountered

In our approach we have done experiments and investigations with many Semantic Web tools and technologies. In the current implementation we use:

- Protégé [8] for modelling the domain ontology and provider ontology
- RACER [10] for doing reasoning on the domain ontology
- Sesame [9] for querying and navigating the Semantic Information Sources
- RDF(S) [5, 6] for describing content in the Semantic Information Sources
- RQL [3] for the System Queries (SQ)
- OWL [7] for describing the domain ontology
- W3C result format [12] for transforming the results coming from the heterogeneous information sources into a common result format

Based upon the work done so far, the good experience with respect to the use of these can be summarized as: OWL better supports the knowledge representation work involved in building the domain ontology, and other ML ontologies. RQL offers an effective low level query language. Protégé, with the OWL plug-in, provides a good ontology editor and development environment.

The following problems encountered when using these technologies: Sesame, like other general purpose ontology stores, is currently too slow for the query process and to support the kind of ontology-based inference needed by the ML. In order to increase the performance concerning the query process we implemented a logic-based query optimizer for RQL queries. RACER can provide useful support to ontology development, but becomes too slow for ontologies like the ML domain ontology (with approximately 790 concepts and individuals, and 150 relations). None of the published ontology development methods [11], either do not have a validation step, or are strong enough to support effective validation of realistic sizes of ontologies. The rather toy examples typically used to present these methods also don't help much in

understanding how to apply them to real ontology developments. Furthermore, we explored that the expressiveness of the ontology description language RDF(S) used in the CL are restricted according to our needs. Currently, it is not possible to relate instances and concepts with user defined RDF(S) properties. If this would be possible, query languages need to be extended in order to allow querying those modelling approaches. The user testing sessions showed that the user interface of the ontology editing tool Protégé is not for everybody. It is still too complex for users who are not familiar with it.

6 Future Needs

To support interoperability between heterogeneous semantic web applications, the need comes up to standardize ontologies. Furthermore, high-performance OWL tools that can handle large size ontologies are still missing.

Acknowledgements

This work was funded in part by the European Commission Grant #IST-2001-34417, Semantic Web-based Information Management and Knowledge Sharing for Innovative Product Design and Engineering (WIDE project).

References

- [1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, pp 34–43, May 2001.
- [2] Backus-Naur form (BNF), WIKIPEDIA, <[http://en.wikipedia.org/wiki/Backus-Naur Form](http://en.wikipedia.org/wiki/Backus-Naur_Form)>.
- [3] The RDF Query Language (RQL), FORTH Institute of Computer Science, <<http://athena.ics.forth.gr:9090/RDF/RQL/>>.
- [4] Association for Standardisation of Automation and Measuring Systems (ASAM) Open Data Service (ODS), <http://www.asam.net/01_asam-ev_01.php>.
- [5] Resource Description Framework (RDF), W3C Semantic Web Activity, Technology and Society Domain, <<http://www.w3.org/RDF/>>.
- [6] RDF Vocabulary Description Language 1.0: RDF Schema, W3C Technical Reports and Publications, <<http://www.w3.org/TR/rdf-schema/>>.
- [7] OWL Web Ontology Language Overview, W3C Technical Reports and Publications, <<http://www.w3.org/TR/owl-features/>>.
- [8] The Protégé Ontology Editor, Stanford Medical Informatics, Stanford University School of Medicine, <<http://protege.stanford.edu/>>.
- [9] J. Broekstra and A. Kampman and F. van Harmelen, "Sesame: A generic architecture for storing and querying RDF and RDF Schema, International Semantic Web Conference (ISWC), pp 54–68, 2002.
- [10] RACER: Semantic Middleware for Industrial Projects based on RDF/OWL, <<http://www.cs.concordia.ca/~haarslev/racer/>>.

- [11] A. Gomez-Perez, M. Fernandez-Lopez and O. Corcho, "Ontological Engineering," London:Springer-Verlag, 2004.
- [12] A. Seaborne, Recording Query Results, W3C Discussion document <http://www.w3.org/2003/03/rdfqr-tests/recording-query-results.html>
- [13] RDF semantics W3C Recommendation 10 February 2004 <http://www.w3.org/TR/rdf-mt/>
- [14] S. Staab, M. Erdmann, and A. Maedche. An Extensible Approach for Modeling Ontologies in RDF(S). In First ECDL'2000 SemanticWebWorkshop, Lisbon, Portugal, 2000.
- [15] H. Stuckenschmidt et al. Exploring Large Document Repositories with RDF Technology: The DOPE Project Published by the IEEE Computer Society <http://www.cs.vu.nl/~frankh/postscript/IEEE-IS04.pdf>
- [16] N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and m.c.schraefel, University of Southampton; CS AKTive Space, or How We Learned to Stop Worrying and Love the Semantic Web; Published by the IEEE Computer Science;

Collaborative and Usage-Driven Evolution of Personal Ontologies

Peter Haase¹, Andreas Hotho², Lars Schmidt-Thieme³, and York Sure¹

¹ Institute AIFB, U of Karlsruhe, Germany

{haase, sure}@aifb.uni-karlsruhe.de

² Knowledge Discovery Engineering Group, U of Kassel, Germany

hotho@cs.uni-kassel.de

³ Computer-based New Media Group,

Institute for Computer Science, U of Freiburg, Germany

lst@informatik.uni-freiburg.de

Abstract. Large information repositories as digital libraries, online shops, etc. rely on a taxonomy of the objects under consideration to structure the vast contents and facilitate browsing and searching (e.g., ACM topic classification for computer science literature, Amazon product taxonomy, etc.). As in heterogeneous communities users typically will use different parts of such an ontology with varying intensity, customization and personalization of the ontologies is desirable. Of particular interest for supporting users during the personalization are collaborative filtering systems which can produce personal recommendations by computing the similarity between own preferences and the one of other people. In this paper we adapt a collaborative filtering recommender system to assist users in the management and evolution of their personal ontology by providing detailed suggestions of ontology changes. Such a system has been implemented in the context of Bibster, a peer-to-peer based personal bibliography management tool. Finally, we report on an experiment with the Bibster community that shows the performance improvements over non-personalized recommendations.

1 Introduction

Large information repositories as digital libraries, online shops, etc. rely on a taxonomy of the objects under consideration to structure the vast contents and facilitate browsing and searching (e.g., ACM Topic Hierarchy for computer science literature, Amazon product taxonomy, etc.). As in heterogeneous communities users typically will use different parts of such an ontology with varying intensity, customization and personalization of the ontologies is desirable.

Such personal ontologies reflect the interests of users at certain times. Interests might change as well as the available data, therefore the personalization requires quite naturally support for the evolution of personal ontologies. The sheer size of e.g. the ACM Topic Hierarchy makes it quite difficult for users to easily locate topics which are relevant for them. Often one can benefit from having a community of users which allows for recommending relevant topics according to similar interests. Of particular

interest are therefore collaborative filtering systems which can produce personal recommendations by computing the similarity between own preferences and the one of other people.

We performed our evaluation within the Bibster community. Bibster is a semantics-based Peer-to-Peer application aiming at researchers who want to benefit from sharing bibliographic metadata. It enables the management of bibliographic metadata in a Peer-to-Peer fashion: it allows to import bibliographic metadata, e.g. from BIBTEX files, into a local knowledge repository, to share and search the knowledge in the Peer-to-Peer system, as well as to edit and export the bibliographic metadata.

As our main contribution in this paper we adapt a collaborative filtering recommender system to assist users in the management and evolution of their personal ontology by providing detailed suggestions of ontology changes. The approach is implemented as an extension of the Bibster application and has been thoroughly evaluated with very promising results.

The paper is structured as follows. In the next Section 2 we present related work in the areas of work in recommender systems, work in using taxonomies in recommender systems, and work in learning taxonomies and ontology evolution in general. In Section 3 we describe our underlying ontology model which is based on OWL, the change operations used during the evolution of ontologies, and the ontology rating annotations allowing each user to express more fine-grained the importance of certain ontology parts. The recommender method itself and its functionality is illustrated in Section 4. We will introduce the Bibster applications and its extensions with the recommender functionality in Section 5 followed by evaluation results in Section 6. The evaluation was performed as an experiment within the Bibster community and shows the performance improvements over non-personalized recommendations. Finally, we conclude in Section 7.

2 Related Work

Related work exists in three different aspects: work in recommender systems, especially collaborative filtering in general, work in using taxonomies in recommender systems, and work in learning taxonomies and ontology evolution in general.

Recommender systems have their roots in relevance feedback in information retrieval [15], i.e., adding terms to (query expansion) or re-weighting terms of (term re-weighting) a query to a document repository based on terms in documents in the result set of the original query that have been marked relevant or non-relevant by the user, as well as adaptive hypertext and hypermedia [20], i.e., the automatic adaptation of the link structure of a document repository based on previous link usage by users.

Recommender systems broaden the domain from documents and link structure to arbitrary domains (e.g., movies, products), do not necessarily rely on attributes of the objects under consideration (i.e., terms in the case of documents and called *items* in the context of recommender systems), and typically combine knowledge about different users. They first have been formulated as filtering techniques generally grouped in three different types: (1) *collaborative filtering* is basically a nearest-neighbor model based on user-item correlations; if correlations are computed between users, it is called

user-based, if between items, it is called *item-based*. (2) *content-based* or *feature-based* recommender systems use similarities between rated items of a single user and items in the repository. User- and item-based collaborative filtering and content-based recommender systems have been introduced in [5, 14], [16] and [1], respectively, and are exemplified by the three systems presented there, MovieLens, Ringo, and fab. (3) *Hybrid* recommender systems try to combine both approaches [1, 2]. Although most recommender systems research meanwhile focuses on more complex models treating the task as a learning or classification problem, collaborative filtering models still are under active investigation [8, 3] due to their simplicity and comparable fair quality.

Taxonomies are used in recommender systems to improve recommendation quality for items, e.g., in [13] and [21]. But to our knowledge there is no former approach for the inverse task, to use recommender systems for the personalization of the taxonomy or more generally of an ontology.

Ontology evolution is a central task in ontology management that has been addressed for example in [12] and [17]. In [17] the authors identify a possible six-phase evolution process: (1) change capturing, (2) change representation, (3) semantics of change, (4) change implementation, (5) change propagation, and (6) change validation. Our work addresses the phase of change capturing, more specifically the process of capturing implicit requirements for ontology changes from usage information about the ontology. One approach for *usage-driven change discovery* in ontology management systems has been explored in [19], where the user's behavior during the knowledge providing and searching phase is analyzed. [18] describes a tool for guiding ontology managers through the modification of an ontology based on the analysis of end-users' interactions with ontology-based applications, which are tracked in a usage-log. However, the existing work only addressed the evolution of a single ontology in a centralized scenario. In our work we are extending the idea of applying usage-information to a multi-ontology model by using collaborative filtering to recommend ontology changes based on the usage of the personal ontologies.

3 Ontology Model and Ontology Change Operations

3.1 Ontology Model

As the OWL ontology language has been standardized by the W3C consortium, we will adhere to the underlying OWL ontology model. Because of their computational characteristics, the sublanguages OWL-DL and OWL-Lite are of particular importance. These languages are syntactic variants of the $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$ description logics, respectively [9]. In the following we will therefore use the more compact, traditional $\mathcal{SHOIN}(\mathbf{D})$ description logic syntax, which we review in the following:

We use a datatype theory $\mathbf{D}$, a set of concept names N_C , sets of abstract and concrete individual names N_{I_a} and N_{I_c} , respectively, and sets of abstract and concrete role names N_{R_a} and N_{R_c} , respectively.

The set of $\mathcal{SHOIN}(\mathbf{D})$ *concepts* is defined by the following syntactic rules, where A is an atomic concept, R is an abstract role, S is an abstract simple role (a role not having transitive subroles), $T_{(i)}$ are concrete roles, d is a concrete domain predicate,

a_i and c_i are abstract and concrete individuals, respectively, and n is a non-negative integer:

$$\begin{aligned} C &\rightarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n S \mid \leq n S \mid \{a_1, \dots, a_n\} \mid \\ &\quad \mid \geq n T \mid \leq n T \mid \exists T_1, \dots, T_n.D \mid \forall T_1, \dots, T_n.D \\ D &\rightarrow d \mid \{c_1, \dots, c_n\} \end{aligned}$$

An ontology is a finite set of axioms of the form¹:

- concept inclusion axioms $C \sqsubseteq D$, stating that the concept C is a subconcept of the concept D ,
- transitivity axioms $\text{Trans}(R)$, stating that the abstract role R is transitive,
- role inclusion axioms $R \sqsubseteq S$ ($T \sqsubseteq U$) stating that the abstract role R (or concrete role T) is a subrole of the abstract role S (or concrete role U).
- concept assertions $C(a)$ stating that the abstract individual a is in the extension of the concept C ,
- abstract role assertions $R(a, b)$ and $T(a, c)$ stating that the abstract individuals a , b (or a , c) are in the extension of the role R (T),
- concrete role assertions $T(a, c)$ stating that the abstract individual a and the concrete individual c are in the extension of the concrete role T ,
- individual (in)equalities $a \approx b$, and $a \not\approx b$, respectively, stating that a and b denote the same (different) individuals.

In the following, we denote the set of all possible ontologies with $\mathcal{O}$.

3.2 Ontology Change Operations

Definition 1. An ontology change operation $oco \in \text{OCO}$ is a function $oco : \mathcal{O} \rightarrow \mathcal{O}$. Here OCO denotes the set of all possible ontology change operations.

For the above defined ontology model, we allow the atomic change operations of adding and removing axioms, which we denote with α^+ and α^- , respectively. Complex ontology change operations can be expressed as a sequence of atomic ontology change operations. The semantics of the sequence is the chaining of the corresponding functions: For some atomic change operations $oco_1, \dots, oco_n$ we can define $oco_{\text{complex}} = oco_n \circ \dots \circ oco_1 = oco_n(\dots oco_1)$.

3.3 Ontology Rating Annotations

Our ontology model so far describes the actual state of an ontology for a user. Once we enter the more dynamic scenario of ontology evolution, it makes sense that a user (i) can express more in a more fine-grained way how important a certain symbol (name) or axiom is for him and (ii) can express explicitly negative ratings for symbols (names) or axioms not part of his ontology. In the context of software configuration management, the latter is known as specifying a "taboo".

We model this importance information by a rating annotation.

¹ For the direct model-theoretic semantics of $\text{SHOIN}(\mathbf{D})$ we refer the reader to [10].

Definition 2. Let $N := N_C \cup N_{I_a} \cup N_{I_c} \cup N_{R_a} \cup N_{R_c}$ denote the set of all possible names (symbols) and $\mathcal{A}$ the set of all possible axioms, then an ontology rating annotation is a partial function $r : N \cup \mathcal{A} \rightarrow \mathbb{R}$.

The definition states that we allow ratings on both the axioms of the ontologies as well as the names over which the axioms are defined. High values denote the relative importance of a symbol or axiom, negative values that it is unwanted by the user.

In particular, we define the following two ontology rating annotations:

1. We use an explicit rating, called the membership-rating r^m with taboos, for which (i) all symbols and axioms actually part of the ontology have rating +1, (ii) all symbols and axioms not actually part of the ontology can be explicitly marked taboo by the user and then get a rating -1.
2. We use an implicit, usage-based rating called r^u , which indicates the relevance of the elements based on how it has been used, e.g. counts the percentage of queries issued by the user and instances in his knowledge base that reference a given symbol name.

We will consider rating annotations as an additional ontology component in the following.

3.4 Ontology Alignment

An additional problem that we have to face is: If two ontologies talk about a name s , does this name refer to the same entity? Generally, this will not be the case and we will have to establish mappings between the symbol names of each pair of ontologies. This problem is known as *ontology alignment* or *ontology mapping* in the literature.

As in most applications, individuals eventually may have global IDs – e.g., URIs for web pages, ISBNs for books, etc. – concepts and relations typically have not. But although we think that the ontology alignment task is a crucial requirement for recommending ontology changes, for the sake of simplicity we will not pursue this problem here any further and rather refer the reader to e.g. [11]. In the following we assume that all symbols are global identifiers.

4 Recommending Ontology Changes

A recommender system for ontology changes tries to suggest ontology changes to the user based on some information about him and potential other users. Formally, an *ontology recommender* is a map

$$\varrho : \mathcal{X} \rightarrow \mathcal{P}(\text{OCO}) \quad (1)$$

where $\mathcal{X}$ contains suitable descriptions of the target ontology and user.

For example, let recommendations depend only on the actual state of a user's ontology, i.e., $\mathcal{X} = \mathcal{O}$. where $\mathcal{O}$ denotes the set of possible ontologies. A simple ontology evolution recommender can be built by just evaluating some heuristics on the actual state of the ontology, e.g., if the number of instances of a concept exceeds a given

threshold, it recommends to add subconcepts to this concept. But without any additional information, this is hardly very useful, as we would not be able to give any semantics to these subconcepts: we could recommend to further subdivide the concept, but not how, i.e., neither be able to suggest a suitable label for these subconcepts nor assertions of instances to them. We will call such an approach *content-based* to distinguish it from more complex ones.

Recommendation quality eventually can be improved by taking into account other users' ontologies and thereby establishing some kind of collaborative ontology evolution scenario, where each user keeps his personal ontology but still profits from annotations of other users. The basic idea is as follows: assume that for a target ontology we know similar ontologies called *neighbors* for short, then we would like to spot patterns in similar ontologies that are absent in our target ontology and recommend them to the target ontology. Another wording of the same idea is that we would like to extract ontology change operations that applied to the target ontology increases the similarity with its neighbors.

Let

$$\text{sim} : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R} \quad (2)$$

be such a similarity measure where $\text{sim}(O, P)$ is large for similar ontologies O and P and small for dissimilar ontologies. Typically, these measures are symmetric and maximal for two same arguments. For further properties and examples of similarity functions for ontologies, we refer the reader to [4].

Recall that ontologies in our context may have additional rating annotations that are valuable information to consider in similarity measures suitable for recommendation tasks.

We can choose a simple unnormalized correlation measure (vector similarity) to compute similarities between ontologies of two users based on their ratings of the elements (i.e. symbol names and axioms) in the ontology:

$$\text{sim}_r(O, P) := \frac{\sum_{s \in N \cup A} r_O(s) r_P(s)}{\sqrt{\sum_{s \in N \cup A} r_O(s)^2} \sqrt{\sum_{s \in N \cup A} r_P(s)^2}} \quad (3)$$

Similarities for the two different rating annotations r^m and r^u are computed separately and then linearly combined with equal weights:

$$\text{sim}(O, P) := \frac{1}{2} \text{sim}_{r^m}(O, P) + \frac{1}{2} \text{sim}_{r^u}(O, P) \quad (4)$$

Finally, as in standard user-based collaborative filtering, ratings of all neighbors Ω are aggregated using the similarity-weighted sum of their membership ratings r^m , allowing for a personalized recommender function:

$$r_{\text{personalized}}(O, \Omega, c) := \frac{\sum_{P \in \Omega} \text{sim}(O, P) r_P^m(c)}{\sum_{P \in \Omega} |\text{sim}(O, P)|} \quad (5)$$

The recommendations are obtained directly from the rating: Elements with a positive rating are recommended to be added to the ontology, elements with a negative rating are recommended to be removed. Disregarding the similarity measure between

the users' ontologies, we can build a naive recommender that does not provide personalized recommendations, but instead simply recommends "most popular" operations based on an unweighted average of the membership ratings:

$$r_{baseline}(O, \Omega, c) = \frac{\sum_{P \in \Omega} r_P^m(c)}{|\Omega|} \quad (6)$$

5 Case Study: Bibster

In this section we will first introduce the Bibster system [7] and the role of personalized ontologies in its application scenario. We will then describe how the recommender functionality is applied in the system to support the users in evolving their personalized ontologies.

5.1 Application Scenario: Sharing Bibliographic Metadata with Bibster

Bibster² is an award-winning semantics-based Peer-to-Peer application aiming at researchers who want to benefit from sharing bibliographic metadata. Many researchers in computer science keep lists of bibliographic metadata, preferably in BIBTEX format, that they must laboriously maintain manually. At the same time, many researchers are willing to share these resources, assuming they do not have to invest work in doing so.

Bibster enables the management of bibliographic metadata in a Peer-to-Peer fashion: it allows to import bibliographic metadata, e.g. from BIBTEX files, into a local knowledge repository, to share and search the knowledge in the Peer-to-Peer system, as well as to edit and export the bibliographic metadata.

Two ontologies are used to describe properties of bibliographic entries in Bibster, an application ontology and a domain ontology [6]. Bibster makes a rather strong commitment to the application ontology, but the domain ontology can be easily substituted to allow for the adaption to different domains.

Bibster uses the SWRC³ ontology as application ontology, that describes different generic aspects of bibliographic metadata. The SWRC ontology has been used already in various projects, e.g. also in the semantic portal of the Institute AIFB⁴.

In our scenario we use the ACM Topic Hierarchy⁵ as the domain ontology. This topic hierarchy describes specific categories of literature for the Computer Science domain. It covers large areas of computer science, containing over 1287 topics ordered using taxonomic relations, e.g.:

SubTopic(Artificial_Intelligence, Knowledge_Representation_Formalisms).

The *SubTopic* relation is transitive, i.e. $\text{Trans}(\text{SubTopic})$.

The domain ontology is being used for classification of metadata entries, e.g. *isAbout(someArticle, Artificial_Intelligence)*, therefore enabling advanced query-

² <http://bibster.semanticweb.org/>

³ <http://ontoware.org/projects/swrc/>

⁴ <http://www.aifb.uni-karlsruhe.de/about.html>

⁵ <http://www.acm.org/class/1998/>

ing and browsing. The classification can be done automatically by the application or manually (by drag and drop).

5.2 Extensions for Evolution and Recommendations

In Bibster we initially assumed both ontologies to be globally shared and static. This basically holds for the application ontology, but users want to adapt the domain ontology continuously to their needs. This is largely motivated by the sheer size of the ACM Topic Hierarchy which makes browsing, and therefore also querying and manual classification, difficult for users.

As part of this work we implemented extensions as described in the previous Section 4 to Bibster which support the evolution – i.e. the continuous adaptation – of the domain ontology by the users. A basic assumption here is that all users agree in general on the ACM Topic Hierarchy as domain ontology, but each user is only interested in seeing those parts of it which are relevant for him at a certain point of time.

In the application, we have separated the interaction with the ontology in two modes: a *usage mode* and an *evolution mode*. The usage mode is active for the management of the bibliographic metadata itself, i.e. creating and searching for the bibliographic metadata. This mode only shows the current view on the ontology consisting of the topics that the user has explicitly included in his ontology. The evolution mode allows for the adaptation of the ontology. In this mode also the possible extensions along with the corresponding recommendations are shown.

Ontology Change Operations. To keep things simple and trying to separate effects from eventually different sources as much as possible, we allow as change operations the addition and removal of topics from the personal ontology. More specifically, this addition/removal corresponds to the addition/removal of the individual assertion axiom (e.g. *Topic(Knowledge_Representation_Formalisms)*) and the role assertion axiom that fixes the position in the topic hierarchy (e.g. *SubTopic(Artificial_Intelligence, Knowledge_Representation_Formalisms)*). The addition of topics is restricted to those topics that are predefined in the ACM Topic Hierarchy. Also, the position of the topics is fixed globally by the background ontology.

Ontology Ratings. To elicit as much information as possible from users' work with the application, we gather various ontology rating annotations in the different modes.

Rating	Recommendation		
	Remove	Neutral	Add
Taboo-ed	X topicname	X topicname	+ topicname
Unrated	- topicname	? topicname	+ topicname
Accepted	- topicname	√ topicname	√ topicname

Fig. 1. Visualization of topics in evolution mode

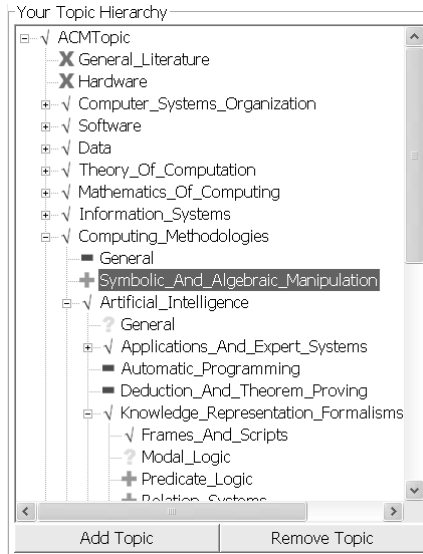


Fig. 2. Screenshot

We obtain the membership-rating r^m in the evolution mode from the explicit user actions (c.f. Figure 2): The user can either add a topic in the taxonomy, which will assigning a rating +1 for the topic, or he can exclude (taboo) the topic from the taxonomy, which will assign -1 for the explicitly taboo-ed topic.

We obtain the usage-based rating r^u in the usage mode by counting the percentage of queries issued by the user and instances in his knowledge base that reference a given topic. (For this, references to all topics are retained, especially also to topics not contained in the ontology of the user.)

The ontology ratings of the individual users are propagated together with peer profile descriptions as advertisements in the Peer-to-Peer network, such that every peers is informed about the usage of the ontology in the network. For the details of this process, we refer the reader to [7].

Recommending Ontology Changes. For the recommendations of topics we rely on the rating function $r_{\text{personalized}}$ presented in the previous section. From the ratings of the topics, we can directly obtain the recommendations: Topics with a positive rating are recommended to be added to the ontology, topics with a negative rating are recommended to be removed. (Please note that adding a topic actually means adding the corresponding axioms, as described above.)

Topics in the topic hierarchy are visualized depending on the current rating r^m of the topic and on the recommendation for the topic using a the coding scheme shown in Figure 1. Figure 2 shows a screenshot of the ontology in the evolution mode.

6 Evaluation

For our evaluation, we wanted to study two questions: (i) do users accept recommendations for ontology changes at all? (ii) is a personalized recommender better suited for the task than a naive, non-personalized recommender?

To answer these questions, we have performed a user experiment in an in-situ setting using the Bibster system, in which we compared the baseline (non-personalized) and the personalized recommender, as defined in the previous section. In the following we will describe the setup of the experiment, evaluation measures, and the results.

6.1 Design of the Experiment

The experiment was performed within three Computer Science departments at different locations. For a pre-arranged period of one hour, 23 users were actively using the system. The recommender strategy (baseline or personalized) was chosen randomly for each user at the first start of the Bibster application. The users were not aware of the existence of the different recommendation strategies.

During the experiment, the users performed the following activities (in no particular order), which are typical for the everyday use of the system:

- *Import data:* The users need to load their personal bibliography as initial dataset. This data should also reflect their research interest. As described before, the classification information of the bibliographic instances is part of the ontology rating and thus used to compute the similarity between the peers.
- *Perform queries:* The users were asked to search for bibliographic entries of their interest by performing queries in the Peer-to-Peer system. These queries may refer to specific topics in the ontology, and are thus again used as ontology ratings.
- *Adapt ontology:* Finally the users were asked to adapt their ontology to their personal needs and interests by adding or removing topics. This process was guided by the recommendations of the respective recommender function. The recommendations were updated (recalculated) after every ontology change operation.

The user actions were logged at every peer for later analysis. The logged information included: The type of the action (e.g. user query, ontology change operations), the provided recommendations, and a timestamp.

6.2 Evaluation Measures

We base our evaluation on the collected usage information in form of events consisting of the actual user action $e \in \text{OCO}$, i.e., the specific ontology change operation performed, and the set $\hat{E} \subseteq \text{OCO}$ of recommendations at that point in time, represented by a set $\mathcal{E} \subseteq \text{OCO} \times \mathcal{P}(\text{OCO})$.

We observe a successful recommendation or a *hit*, when $e \in \hat{E}$. For non-hits, we distinguish two situations: (i) If the actual recommendation was exactly the opposite action, e.g., we recommended to add a topic but the user taboo-ed it, then we call this an *error*. (ii) If there was no recommendation for this action neither for its opposite, we

call this *restraint*. Based on these counts, we can compute the following performance measures.

$$recall(\mathcal{E}) := \frac{|\{(e, \hat{E}) \in \mathcal{E} \mid e \in \hat{E}\}|}{|\mathcal{E}|} \quad (7)$$

$$error(\mathcal{E}) := \frac{|\{(e, \hat{E}) \in \mathcal{E} \mid opp(e) \in \hat{E}\}|}{|\mathcal{E}|} \quad (8)$$

$$restraint(\mathcal{E}) := \frac{|\{(e, \hat{E}) \in \mathcal{E} \mid opp(e) \notin \hat{E} \wedge e \notin \hat{E}\}|}{|\mathcal{E}|} \quad (9)$$

where *opp* denotes the respective opposite operation, e.g., $opp(e^+) := e^-$ and $opp(e^-) := e^+$. Higher recall and lower error and restraint are better.

For a higher level of detail, we do so not only for all user actions, but also for some classes $OCOC \subseteq OCO$ of user actions, such as all *add*- and all *remove/taboo*-operations.

As each of the measures alone can be optimized by a trivial strategy, we also computed the profit of the recommenders w.r.t. the profit matrix in Table 1:

$$profit(\mathcal{E}) := \frac{\sum_{(e, \hat{E}) \in \mathcal{E}} \sum_{\hat{e} \in \hat{E}} profit(e, \hat{e})}{|\mathcal{E}|} = recall(\mathcal{E}) - error(\mathcal{E}) \quad (10)$$

Table 1. Evaluation Profit Matrix

User Action	Recommendation		
	Remove	None	Add
Remove	1	0	-1
None	0	0	0
Add	-1	0	1

An intuitive reading of the profit is: The higher the profit, the better the performance of the recommender. In the best case ($profit = 1$), all user actions were correctly recommended by the system, in the worst case ($profit = -1$), all user actions were opposite of the recommendation.

6.3 Evaluation Results

For the 23 participating users in the experiment, the baseline recommender was active for 10 users, the personalized recommender was active for the other 13 users. The participants performed a total of 669 user actions (452 add topic and 217 remove topic), 335 of these action were performed by users with the baseline strategy, 334 by users with the personalized recommender. Table 2 shows the number of add-topic-actions for the most popular topics. Figure 3 shows the cumulative results of the performance measures defined above for the baseline and the personalized recommender. The diagrams show the results for *Add* and *Remove* operations separately, as well as combined for all change operations.

As we can see in Figure 3 (upper right), overall the personalized recommender correctly recommended more than 55% of the user actions, while the baseline achieved less

Table 2. Most Popular Topics

ACM Topic	# Add Actions
Information_Systems	23
Computing_Methodologies	15
Data	14
Computing_Methodologies/Artificial_Intelligence	12
Information_Systems/Database_Management	12
Software	11
Mathematics.Of_Computing	10
Computer_Systems_Organization	10
Computer_Systems_Organization/Computer_Communication_Networks	10
Computing_Methodologies/Artificial_Intelligence/ Knowledge_Representation_Formalisms_And_Methods	10

than 30%. The error rate of the baseline algorithm is considerably higher: We observed an $error = 17\%$ and 9% for the baseline and the personalized approach, respectively. Further we observed a very large amount of restraint operations with $restraint = 67\%$ for users with the baseline strategy. Probably this is the result of a large number of recommendations irrelevant to the user given by the system with the baseline strategy. In such a case the user would not like to follow the system and constructs the ontology

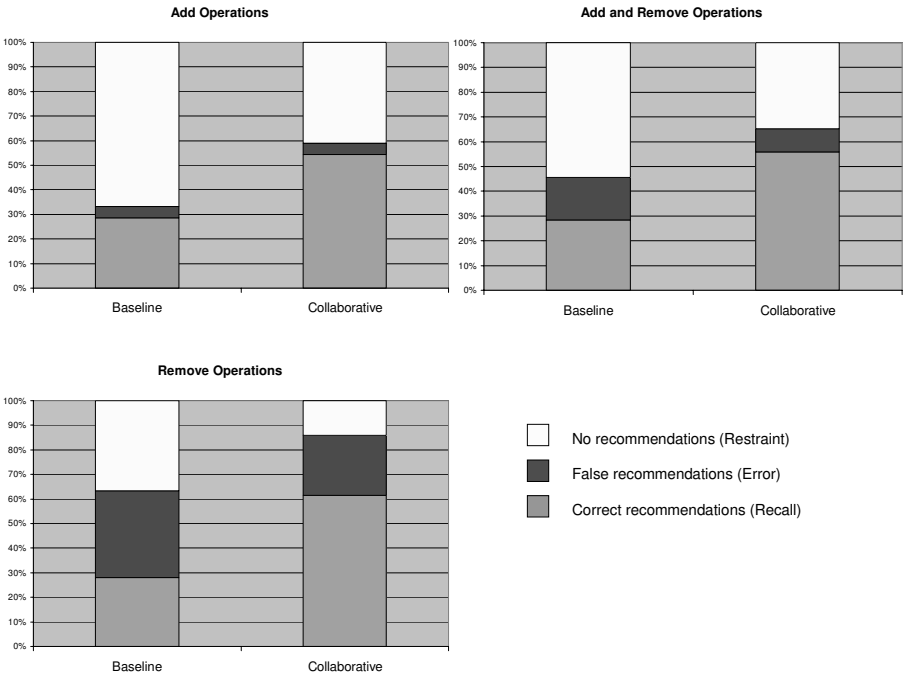


Fig. 3. Performance measures of the recommender

mainly by themselves. Only from time to time he takes some of the recommendations into account.

By comparing add and remove operations we observe a higher amount of *error* recommendations for remove operations in comparison to the a really small amount of it for the add recommendations while the correct recommendations are comparable for both operations (cf. Figure 3, left side). We think that this observation is based on the fact that a user is more likely to follow an add operation without a “substantiated” reason or explanation than a remove operation. While adding something to his “collection” and following the idea of having more the remove operation forces the feeling of “loosing” something, so typically users are more reluctant to remove topics.

Calculating the overall profit of the two recommender functions, we obtain $profit(\mathcal{E}) = 0.11$ for the baseline recommender. For the collaborative recommender, we obtain a significantly better value of $profit(\mathcal{E}) = 0.47$. Concluding we can state that the personalized recommender function provides substantially more useful recommendations.

7 Conclusion and Future Work

We have presented an approach to recommend ontology change operations to a personalized ontology based on the usage information of the individual ontologies in a user community. In this approach we have adapted a collaborative filtering algorithm to determine the relevance of ontology change operations based on the similarity of the users’ ontologies.

In our experimental evaluation with the Peer-to-Peer system Bibster we have seen that the users actually accept recommendations of the system for the evolution of their personal ontologies. The results further show the benefit of exploiting the similarity between the users’ ontologies in a personalized recommender compared with a simple, non-personalized baseline recommender.

In our experiment we have made various simplifying assumptions. Their relaxation will open fruitful directions for future work: We assumed a fixed background ontology which limits the space of change operations. Relaxing this assumption will introduce challenges related to aligning heterogeneous ontologies. Further, the recommendation of adding or removing concepts in a given concept hierarchy can only be a first step. Next steps will therefore also include recommendations of richer change operations.

Acknowledgments

Research reported in this paper has been partially financed by the EU in the IST project SEKT (IST-2003-506826) (<http://www.sekt-project.com>). We would like to thank our colleagues for fruitful discussions.

References

1. M. Balabanović and Y. Shoham. Fab - content-based, collaborative recommendation. *CACM*, 40(3):66–72, 1997.
2. R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User Adapted Interaction*, 12/4:331–370, 2002.

3. M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Sys.*, 22(1):143–177, 2004.
4. M. Ehrig, P. Haase, and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM 2004*, DEC 2004.
5. D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 35(12):61–70, 1992.
6. N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proc. 1st Int. Conf. on Formal Ont. in Inf. Sys. (FOIS)*, volume 46 of *Frontiers in AI and App.*, Trento, Italy, 1998. IOS-Press.
7. P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004*, NOV 2004.
8. J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Sys.*, 22(1):5–53, 2004.
9. I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4), 2004.
10. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
11. Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowl. Eng. Rev.*, 18(1):1–31, 2003.
12. M. Klein and N. Noy. A component-based framework for ontology evolution. In *Proc. of the WS on Ont. and Distr. Sys., IJCAI '03*, Acapulco, Mexico, Aug.9, 2003.
13. S. Middleton, N. Shadbolt, and D. D. Roure. Ontological user profiling in recommender systems. *ACM Trans. on Inf. Systems*, 22:54–88, 2004.
14. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proc. of the Conf. on Comp. Sup. Coop. Work (CSCW'94)*, pages 175–186, Chapel Hill NC, 1994. Addison-Wesley.
15. G. Salton. Relevance feedback and the optimization of retrieval effectiveness. In G. Salton, editor, *The SMART system — experiments in automatic document processing*, pages 324–336. Prentice-Hall Inc., Englewood Cliffs, NJ, 1971.
16. U. Shardanand and P. Maes. Social information filtering: algorithms for automating “word of mouth”. In *Proc. of the SIGCHI conf. on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.
17. L. Stojanovic, A. Mädche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *European Conf. Knowledge Eng. and Management (EKAW 2002)*, pages 285–300. Springer-Verlag, 2002.
18. N. Stojanovic, J. Hartmann, and J. Gonzalez. Ontomanager - a system for usage-based ontology management. In *In Proc. of FGML Workshop. SIG of German Information Society (FGML - Fachgruppe Maschinelles Lernen GI e.V.)*, 2003.
19. N. Stojanovic and L. Stojanovic. Usage-oriented evolution of ontology-based knowledge management systems. In *Int. Conf. on Ontologies, Databases and Applications of Semantics, (ODBASE 2002)*, Irvine, CA, LNCS, pages 230–242, 2002.
20. P. D. Stotts and R. Furuta. Dynamic adaptation of hypertext structure. In *Hypertext'91 Proc., San Antonio, TX, USA*, pages 219–231. ACM, 1991.
21. C. Ziegler, L. Schmidt-Thieme, and G. Lausen. Exploiting semantic product descriptions for recommender systems. In *Proc. 2nd ACM SIGIR Sem. Web and IR WS (SWIR '04)*, July 25-29, 2004, Sheff., UK, 2004.

Towards Semantically-Interlinked Online Communities

John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker

Digital Enterprise Research Institute (DERI), Galway, Ireland

`firstname.lastname@deri.org`

Abstract. Online community sites have replaced the traditional means of keeping a community informed via libraries and publishing. At present, online communities are islands that are not interlinked. We describe different types of online communities and tools that are currently used to build and support such communities. Ontologies and Semantic Web technologies offer an upgrade path to providing more complex services. Fusing information and inferring links between the various applications and types of information provides relevant insights that make the available information on the Internet more valuable. We present the SIOC ontology which combines terms from vocabularies that already exist with new terms needed to describe the relationships between concepts in the realm of online community sites.

1 Introduction

At the moment, most online communities are islands that are not linked. Sites are hosted on stand-alone systems that cannot be interconnected due to application and interface differences. Parallel discussions on interrelated topics may exist on a number of sites, but their users are unaware of that. There is a huge amount of related information that could be harnessed across such online communities, from similar member profile details to common-topic discussion forums.

The goal of SIOC¹ (Semantically-Interlinked Online Communities) is to interconnect these online communities. Community sites can include many discussion primitives, such as bulletin boards, weblogs and mailing lists, which we have grouped under the concept of forum.

SIOC will facilitate the location of related and relevant information; by searching on one forum, the ontology and interface will allow users to find information on forums from other sites that use a SIOC-based system architecture. Other uses include cross-site querying, topic-related searches, and the importing of SIOC data into other systems, for example, using an email program to browse data imported from a SIOC-enabled site. Therefore, SIOC tries to overcome the serious limitations of current sites in making information accessible to their users in an efficient manner [6].

¹ <http://rdfs.org/sioc/>

A part of the task of linking on-line communities is to suggest additional information related to any given forum and forum entry. One approach would be to perform a search on, for example, post title, author, date, keywords or the full post text in community sites. Existing Internet search engines locate the information by performing a keyword search on a full-text index of Internet resources. Some search engines try to improve the quality of search results by analysing the link structure of web resources. But even with these improvements, search engines lack an understanding of the information being searched for and return a high number of irrelevant results. In this paper, we try to solve this problem by narrowing the scope of a search to a set of interlinked community sites and by describing the information in a machine-readable form using the SIOC ontology.

In a typical usage scenario, a user is searching for information on, for example, installing broadband on a Linux-based PC in their house in Galway. There is a post A discussing local ISPs on site 1, a bulletin board dedicated to Galway, that references (on the HTML level) both a Usenet post B comparing broadband modems and a mailing list post C detailing how to install broadband on Linux. Previously the user would have had to traverse three sites to find the relevant information. However, by making use of the SIOC ontology and remote RDF querying, a search for broadband on the Galway bulletin board will also yield the relevant text from the interlinked Usenet and mailing list posts B and C.

There are some challenges for SIOC. The grand challenge is adoption by community sites, i.e. how can the users be enticed to make use of the SIOC ontology. By using concepts that can be easily understood by site administrators, and by providing properties that are automatically created by an end-user, the SIOC ontology can be adopted in a useful way. A second challenge is how best to use SIOC with existing ontologies. This can be partially solved by mappings and interfaces to commonly-used ontologies such as Dublin Core², FOAF³ and RSS 1.0⁴. Another challenge is how SIOC will scale. If there are more sites to query, then there are more potential relevant results, but also longer response times and higher loads on the participating community sites. We will keep the scaling challenge in mind when creating a future architecture for an interconnected system of community sites.

The main contributions of this paper are the development of the SIOC ontology and mappings to other RDF vocabularies, and a prototype to produce SIOC metadata from a community weblog. These contributions will be detailed as follows. In section 2, we describe the SIOC ontology for linking information both within and between community sites using RDF data, and demonstrate how to map to other existing vocabularies (e.g., FOAF, RSS) and formats (email, XHTML, etc.). In section 3, we will discuss the exchange of SIOC instances by exporting and importing to web-based and legacy discussion systems as well as

² <http://purl.org/dc/elements/1.1/>

³ <http://xmlns.com/foaf/0.1/>

⁴ <http://purl.org/rss/1.0/>

RDF stores. Section 4 will describe some usages of the created instances, and related work will be discussed in section 5. Section 6 concludes the paper.

2 Ontology

In this section we present the SIOC ontology. The ontology consists of two major parts: first, it contains classes and properties that describe discussion forums and posts in online community sites. The ontology is available online⁵. Second, it includes mappings that relate SIOC to existing vocabularies such as FOAF and RSS.

We have identified the main concepts in online communities as Site, Forum, Post, Event, Group and User. These are shown in Figure 1. While similar parent concepts are found in other ontologies, it is the relationships, sub-classes and properties of these concepts in the arena of online discussion methods that make SIOC unique and provide use cases that were not previously possible.

2.1 Main Classes

We list the major classes that are used in the SIOC ontology, and describe their usage in more detail.

Site. is the location of an online community or set of communities, with users in groups creating posts on a set of forums. While an individual forum or group of forums are usually hosted on a centralised site, in the future the concept of a “site” may be extended (for example, a topic thread could be formed by posts in a distributed forum on a peer-to-peer environment).

Forum. can be thought of as a channel or discussion area on which posts are made. A forum can be linked to the site that hosts it. Forums will usually discuss a certain topic or set of related topics, or they may contain discussions entirely devoted to a certain community group or organisation. A forum will have a moderator who can veto or edit posts before or after they appear in the forum. Forums may have a set of subscribed users who are notified when new posts are made. The hierarchy of forums can be defined in terms of parents and children, allowing the creation of structures conforming to topic categories as defined by the site administrator. Examples of forums include mailing lists, online bulletin boards, Usenet newsgroups and weblogs.

Post. is an article or message posted by a user to a forum. A series of posts may be threaded if they share a common subject and are connected by parent and child relationships. Posts will have content and may also have attached files, which can be edited or deleted by the moderator of the forum that contains the post.

Event. is a virtual or real-world event with a single or multiple participants. Examples include meet-ups associated with a particular user or set of users,

⁵ <http://rdfs.org/sioc/ns#>

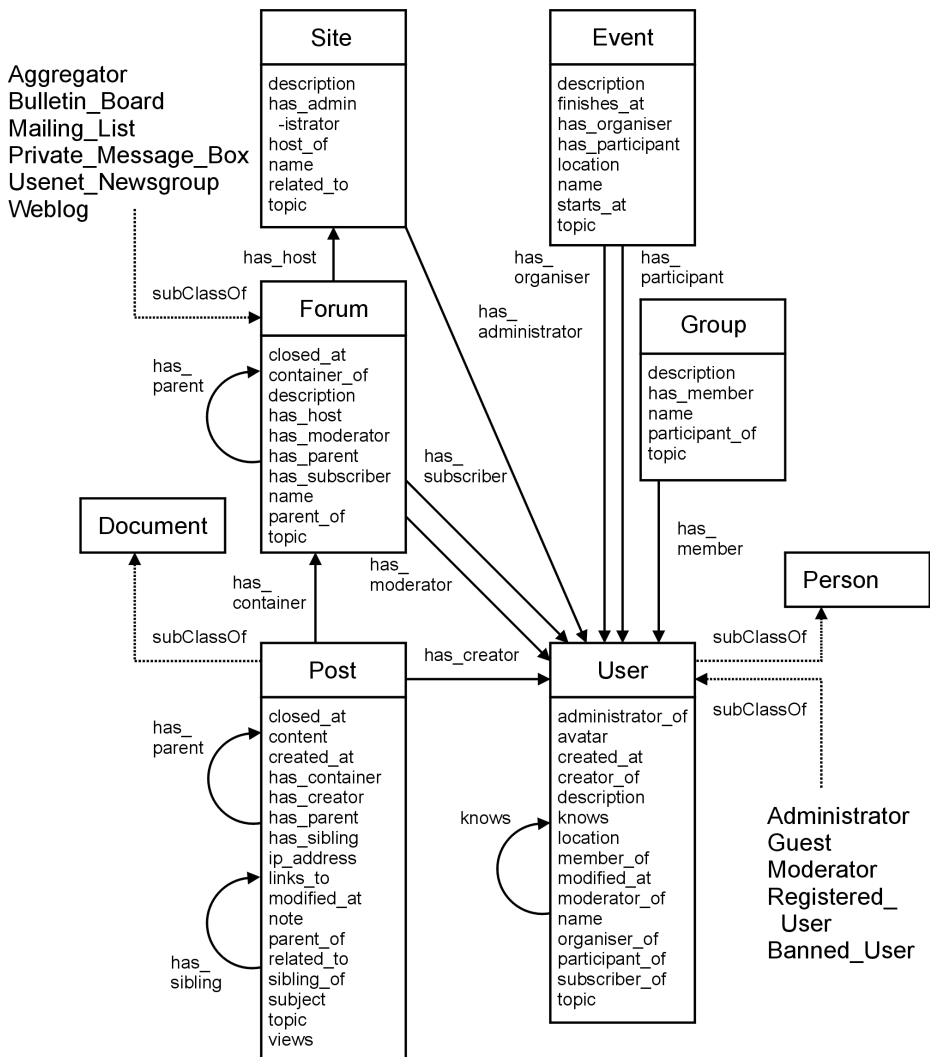


Fig. 1. Overview of classes and properties used in SIOC

a meeting for subscribers of a certain community forum, or private task reminders to a single user.

Group. is a set of members or users of a community site who have a common role, purpose or interest. While a group of users may be a single community that is linked to a certain forum, they may also be a set of users who perform a certain role, for example, moderators or administrators.

User. is a person who is a member of an online community. They are connected to posts that they create or edit, to forums that they are subscribed to or moderate, to sites that they administer, to other users that they know,

and to events that they organise or participate in. Users can be grouped for purposes of allowing access to certain forums or enhanced community site features (weblogs, webmail, etc.).

2.2 Important Properties

In the next paragraphs, we describe properties of SIOC concepts that are important for extracting meaning from and for interlinking online community sites.

topic. A topic definition applies to most of the concepts defined above, and topic metadata can be a useful way to match documents and people to each other.

While it may be more difficult to require a user to assign a topic to a post at creation time, it is more likely that a forum will have an associated topic or set of topics that can be propagated to the posts it contains. Similarly, users or groups can define topics of interest when their profiles are created or modified.

In order to enable the location of related information between the community sites, a common categorisation system has to be used. On large scale, general interest community sites, topics may be quite broad and a general categorisation such as the DMOZ⁶ category hierarchy may be used.

On specialised sites, which may have a very specific category hierarchy, generic categorisation systems are not suitable because they are too broad and may not have the necessary level of detail. For these sites, we propose to define a category hierarchy in the SKOS framework [7] and to create mappings between these concepts and a common category system. In future work, SKOS may be used to describe all category schemes and mappings between them, but the lack of generic taxonomies expressed in SKOS (since it is in an early adoption phase) makes its current use difficult.

A proper use of topics can lead to many interesting scenarios in community sites. For example, a user has defined certain topics of interest on registering an account, after which forums matching those topics are suggested to the user.

views. The views property represents the number of times a particular post or user profile has been viewed. This is an example of where content is automatically created by an end-user, and can increase the content's importance in terms of searching. For example, a user creates a query across a set of SIOC-enabled sites, and is returned a list of subjects and extracts from certain posts, sorted by the popularity of the post, as indicated by the views property.

has_sibling. A recent development in online discussion methods is an article or post that appears in multiple blogs, or has been copied from one forum to another relevant forum. In SIOC, we can treat these copies of posts as siblings of each other if we think of the posts as non-identical twins that share

⁶ <http://dmoz.org/>

most characteristics but differ in some manner. We can avoid duplication of common data in the creation of siblings by linking to the new sibling, the instance of which only contains the changed properties (in the example, `has_container` and `topic` would change). A sibling might also be a version of a post in another language.

closed. The `closed` property applies to posts in a threaded topic, but can also be used for forums. It specifies the date and time that the post or forum was closed. A `closed` property for posts is useful for two reasons. Firstly, it is used to specify that a particular post can have no more children. Secondly, it gives us details of when the closure occurred, and can therefore be used to determine how relevant in time a discussion or set of discussions may be.

has_creator. The `has_creator` property links a post to the user profile of its author. Thus, we can follow the link from the post to the creator and locate the other posts by the same person. The community can be seen as a network of posts with users linked to each post, and there is also a network of other posts created by a given user stemming from there. We can use the information in community sites to locate more contributions by the given author.

knows. The `knows` property is a basic property to show the structure of social networks inside community sites. Who knows whom is the basic property used for describing social network sites and provides information about the links between community users. One of the options to locate relevant information on a given topic is to search for information, not in the full scope of the knowledge base, but in a subset of posts accessed by a person or friends of that person. There are three possible types of `knows` links: linking to a user inside the same community, to a user of other SIOC-enabled communities, or to other resources outside SIOC.

2.3 Mappings

One of the main functions of SIOC is to provide a means for exchanging community instance data. Since there are already a considerable number of classes and properties defined in RDF on the Web, we provide mappings in RDFS and OWL to allow the import and export of SIOC instance data in different vocabularies. Therefore, we can leverage the instance data that is already available.

We provide different kinds of mappings in RDFS for import and export using `rdfs:subClassOf` and `rdfs:subPropertyOf`, and also mappings in OWL using `owl:equivalentProperty` and `owl:equivalentClass` together with other OWL constructs. The mappings to various other RDF vocabularies are online⁷. In Table 1, we show how classes in FOAF, RSS, and various email vocabularies correspond to SIOC classes. Mappings of properties are described in a similar manner.

Carrying out the mappings requires a reasoning engine. Because of the various open issues with regard to OWL reasoning, we split our mappings into two parts. One part defines mappings in RDFS, which is somewhat limited in expressiveness

⁷ <http://rdfs.org/sioc/mappings>

Table 1. Selected SIOC mappings

SIOC	FOAF	RSS	Email	Atom
Site	–	–	–	–
Forum	–	channel	–	feed
Post	Document	item	body	entry
User	Person	–	(from, to, cc)	(author, name)

but there exist scalable reasoning engines that allow for reasoning of class and property hierarchies and classification. A second part is encoded in OWL and describes more complex mapping constructs. At the current stage, we assume that the mappings are carried out on community sites that export or import data, but in theory the mappings can be completely decoupled.

Since mappings in SIOC are not only restricted to ontologies, we provide means to extract information from simple data structures. For example, we might want to map from XML documents into the SIOC ontology using XSL stylesheets⁸. For that purpose, we provide an XSL stylesheet to extract data from XHTML documents to create a SIOC Document instance. In the generic stylesheet, titles, images, and hyperlinks are extracted from Web pages, somewhat similar to how GRDDL⁹ is used to extract information from XHTML documents.

Similarly, an XSL stylesheet can be used that maps from the XML-based RSS formats (0.9x and 2.0) to RSS 1.0, and from there we have RDF mappings to SIOC. Also, we have created a stylesheet that maps Atom¹⁰ to SIOC, and this is used for importing Atom files into SIOC. A mapping from SIOC to Atom for data export requires a combination of queries against RDF data with an Atom template where the appropriate values can be filled in.

3 Exchanging Instances

The core use of SIOC will be in the exchange of instance data between sites. In the following, we elaborate on how the exchange, both importing and exporting data, can be carried out. We show how wrappers can help to achieve export functionality, either based on exporting documents containing the information or by rewriting queries. Another solution for incorporating the “document-based” wrapping into a more sophisticated query infrastructure is to mirror the exported and converted RDF documents in an RDF data store and thus allow for performing queries. We present a third solution, possibly for newly-developed applications, which uses a native RDF repository to store and retrieve statements, making import and export straightforward.

⁸ <http://www.w3.org/TR/xslt>
⁹ <http://www.w3.org/2004/01/rdxh/spec>
¹⁰ <http://www.atomenabled.org/>

3.1 Wrappers to Existing Tools

Wrappers will allow us to export instances of community site concepts such as forums or posts in RDF format. They can also allow us to import SIOC instances to other non-SIOC systems. While there are many possible kinds of community sites for which wrappers could be developed, we will limit discussion to some of them, divided into two categories - legacy systems that do not use HTTP as a transport protocol, and web-based systems that can be accessed via HTTP.

Legacy Systems. A large number of systems preceding the current Web are still deployed and widely used on the Internet. Email is used for exchanging messages and files in an asynchronous way, Internet Relay Chat (IRC) is widely used for synchronous communication, and Usenet is still used to exchange messages. Therefore, to really capture a large amount of data currently exchanged in online communities on the Internet, these legacy systems and protocols need to be considered for SIOC.

In contrast to web-based systems, where we just need to translate the data, we need to employ protocol wrappers for legacy protocols to HTTP. For example, for email we need to translate the data representation format from RFC822¹¹ to SIOC, and provide a wrapper to the access protocol for email stores (usually POP3¹² or IMAP4¹³). Wrappers can be either quite simple (just a dump of the entire data set) or have some “intelligence” that allows for rewriting queries posed over HTTP into the original data format and access protocol. If we also provide importing facilities, for example into a mailing list, then we are building a gateway between a SIOC site and the mailing list.

The email export wrapper accepts a conjunctive query over HTTP GET and returns the results in SIOC. Parameters such as which posts to retrieve, the time duration for results to be returned, etc. are encoded into the query. Certain predicates can be used to restrict the set of posts to retrieve (such as `modified_at > 2004-02-10`). In a next step, the query is parsed and translated into IMAP4 to send to the original data source. The original data source then returns the results in RFC822 format, which is then translated back into RDF and returned to the original caller via HTTP. We have implemented the wrapper and the mapping using the Java programming language.

For imports, the email wrapper can receive `sioc:Posts` via HTTP PUT. Parameters needed for executing the mail sending process are also submitted via a conjunctive query to have the same interface for both GET and PUT. The posts are then translated into the RFC822 format that is suitable for sending via SMTP. The wrapper can then return a status code indicating that the addition of data was completed correctly. The import part of the wrapper still has to be implemented.

¹¹ <http://www.ietf.org/rfc/rfc822.txt>

¹² <http://www.ietf.org/rfc/rfc1939.txt>

¹³ <http://www.ietf.org/rfc/rfc1730.txt>

Interfacing with IRC requires a different approach than wrapping email since the “data representation language” in IRC channel is just free-form text. In IRC, so-called “bots” are responsible for the exchange of data. A very simple bot just logs all utterings in an IRC channel and stores them persistently. More complex bots can understand a defined syntax and perform actions based on the commands issued. Also, some bots understand either a simple query syntax or conjunctive queries that are posed inside the IRC channel. One bot we are providing is logging the channel and recording URIs similar to the chump bot¹⁴. The content that is accumulated is made available in RDF via query over HTTP.

In addition to data that can be auto-generated from the existing sources, a wrapper has to provide additional information which has to be manually added, such as descriptions about mailing lists in sioc:Forum or general information in sioc:Site.

Web-Based Systems. Providing mappings from web-based systems is somewhat easier than mapping from legacy systems since protocol translation is not needed here.

We will discuss three kind of community sites using web-based systems - bulletin boards, weblogs and social networking sites. All these systems are based on content management systems with different complexity levels. Therefore exporting and importing information from and to such systems can be accomplished by adding wrapper interfaces to the existing content management systems.

For bulletin boards, some export functionality is already available (e.g. FOAF from vB¹⁵ and phpBB¹⁶, RSS from phpBB¹⁷). Most bulletin board systems use a LAMP (Linux, Apache, MySQL, PHP/Perl) architecture, and a wrapper to export data from these systems will use existing Perl and PHP libraries such as XML.FOAF, Magpie RSS, etc. However, most existing wrappers don't export their data in SIOC, and only provide a document-based export functionality rather than a query interface.

Weblogs usually are small scale systems consisting of one or more contributors and a community of readers. Most weblog engines already have RSS export functionality and there are some experimental implementations of export of other metadata, such as the Wordpress FOAF plugin¹⁸. Since the majority of these engines are open source software, it is straightforward to modify existing export functions to generate SIOC metadata. Import interfaces can be created in a similar way, allowing weblogs to import SIOC data. One of the use cases for SIOC import is replicating post entries among weblogs and community sites.

Social networking sites are based around the concept of persons and the relations between them. At the same time, many social networking sites are imple-

¹⁴ <http://usefulinc.com/chump/>

¹⁵ <http://www.vbulletin.org/forum/showthread.php?t=66434>

¹⁶ <http://www.phpbb.com/phpBB/viewtopic.php?p=1088960>

¹⁷ <http://www.phpbb.com/phpBB/viewtopic.php?t=144548>

¹⁸ <http://www.wasab.dk/morten/blog/archives/2004/07/05/wordpress-plugin-foaf-output>

menting other functionality, such as bulletin boards or forums. There are existing implementations of FOAF metadata exports of user profiles on ecademy.com and Tribe.net. Similarly for bulletin boards, wrappers to export SIOC metadata on posts and forums can be created using existing Perl and PHP libraries. However, many social networking sites are members-only and are not viewable to the outside world, which raises a question of privacy and trust regarding the information exported from these sites. The issue of privacy can be partially addressed by exchanging sensitive information only among a closed network of trusted community sites.

The main challenge for using SIOC with web-based systems are not in the technical implementation of SIOC wrappers, but rather in the wide adoption of the SIOC ontology to gain incentives for people to provide data and tools for SIOC.

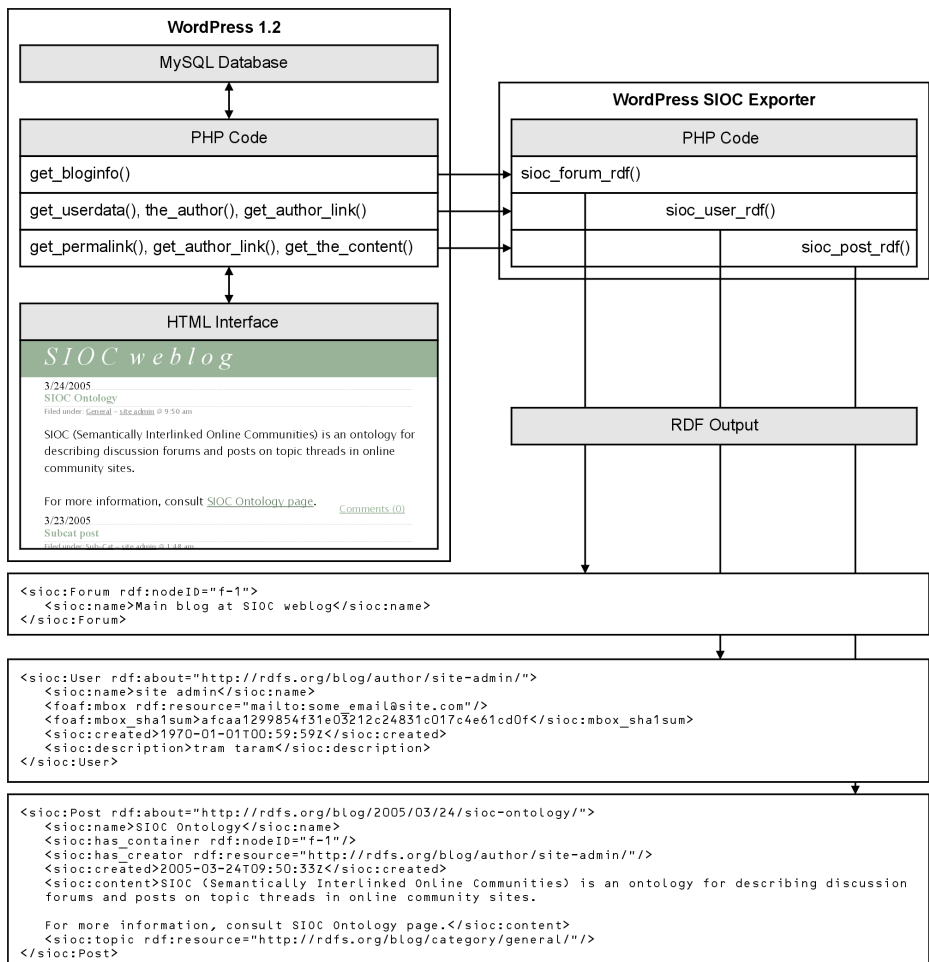


Fig. 2. SIOC metadata export from WordPress

By making SIOC data available through exports, we are encouraging the adoption of SIOC concepts. To this end, we have created a SIOC metadata export facility¹⁹ for the WordPress weblog engine. This makes use of existing WordPress PHP functions to access the information about posts, users and forums (weblog channels) from the underlying relational database. SIOC metadata in RDF is generated for each concept instance. The export process is illustrated by example in Figure 2. Other export facilities are being written for the bulletin board systems phpBB and vBulletin, and the content management system Drupal.

3.2 Mirror Data in RDF Store

Most of the web-based wrappers just provide simple document-based export facilities. Replacing the simple wrappers with full-featured wrappers that are capable of query rewriting takes time. Since our goal is to make SIOC data available for query and to entice people to use SIOC now, we need a method to allow querying of the information that sites publish in flat files.

A solution to provide query facilities for sites that have only simple data export facilities is to replicate the information in a data store that can process queries. Queries are then answered from the replica. The replica is updated either by a scutter - an RDF crawler that traverses `rdfs:seeAlso` links - that periodically crawls the data, or by the original site that pushes updates and changes automatically into the mirror store once the data changes. If the data is exported in a format other than SIOC, then the system also needs to include a component that carries out the mappings from the vocabulary that is used to export data into SIOC.

Replicating the contents of the entire site from the relational database to an RDF store may work initially and create an easy upgrade path. However, in the longer term, storing and integrating data in a native RDF repository is the desirable solution.

3.3 Native RDF Store

The previous two subsections discussed tasks that concerned querying existing sites and their content. We will now describe how newly architected sites can make use of a native RDF repository to store their data.

Exporting data is quite simple because RDF does not restrict you in the way data can be expressed. On the flip side, the flexibility of RDF creates a problem when importing data into systems with a fixed schema. Issues arise here, for example, when an application is importing data using a given schema, and certain mandatory data is missing.

Since community sites provide access to complex structures of information with different types, it is natural to store that information in RDF directly. Repositories such as Jena2 [10], Sesame [3], Redland [1], or YARS [5] can be used to store and retrieve the data. With an RDF store as the data repository,

¹⁹ <http://rdfs.org/sioc/wordpress/>

importing and exporting information is straightforward, and also data integration tasks can be facilitated. An API similar to the RDF NetAPI [9] can be used as well. The route we chose for SIOC is to use a restful interface that uses HTTP methods such as PUT and DELETE for adding and removing data.

We can use an RDF repository as the data store and build the application functionality on top of the repository in a way that is flexible in regards to the schema. The user interface should also function when pieces of data are missing, since we cannot control which data (added or removed from the underlying RDF store) is agnostic to any schema definition.

4 Using SIOC Data

Given the ontology, the mappings, and the wrappers, we are now able to pose queries and add data to individual SIOC sites.

4.1 Browsing

Once we have made the data available using a common query infrastructure, we can use various user interfaces to navigate SIOC data. The simplest solution is to use a mapping from SIOC to a data format where client programs already exist. For example, SIOC data can be mapped to email and then read in any email program. Also, a mapping from SIOC to RSS allows us to navigate a subset of SIOC information inside a regular RSS news reader. Since SIOC has a richer data model than RSS, some information will be lost during the conversion.

Another approach is to use existing RDF browsers such as BrownSauce²⁰ to view arbitrary RDF data. Leveraging the full potential of SIOC requires the provision of custom programs and user interfaces specially tailored towards SIOC.

However, since most programs are already providing browsing facilities for their underlying data structures, implementing import facilities for those programs allows the seamless integration of data without the need for new user interfaces.

4.2 Query

Representing data in SIOC enables users to pose structural queries against the collected data rather than just having keyword search. An implication of structural queries is that you get precise answers as a result, and not just pieces of documents that match the keyword.

Until now, we have only considered querying one community site in isolation. However, since sites are linked together, we might want to perform queries across similar community sites that all share some connections.

²⁰ <http://brownsauce.sourceforge.net/>

One central problem in P2P networks is how to route queries [8]. We plan to exploit the link structure that connects forums or sites to route queries. The forum and site linkage inside SIOC makes it easier to do routing than in general-purpose peer-to-peer networks, since we have some (human-created) links that can be exploited. We expect a scale-free behaviour of these links once SIOC is widely used in practice.

By building the infrastructure for distributing queries into the different site management software or wrappers, we can perform queries without any central components. As a result, querying inside an intranet will be simple and already integrated into the tools used to manage the different community sites inside an organisation, such as mailing lists or forums.

4.3 Locating Related Information

Querying the community sites for information on demand is not the only model of end-user interaction. Another way to enhance the end-user experience is to prepare the data in advance, at creation time of a post.

Once a new post is created in a community site and the SIOC information is available, this site then queries the network of community sites to find related posts. A query is performed based on the post metadata, such as other posts by this person or other posts in the set of the post's topics.

After the information about related resources is received, the community site stores this information using a `related_to` property. Information about the resources the article links to is also extracted from the post body and stored in a `links_to` property. These properties can then be reused by other users of SIOC data and by SIOC and RDF browsers to browse forum entries and navigate through the web of interlinked posts, independent of the underlying site structure that the forums and posts are hosted on.

The results of this information retrieval model are the enhanced functionality added to community sites, and better scalability since the information is prepared in advance.

5 Related Work

Harvest is an early system [2] that can be used to gather information from diverse repositories to build, search, and replicate indexes, and to cache objects as they are retrieved across the Internet. Harvest uses the Summary Object Interchange Format (SOIF) to exchange metadata about resources. In contrast, SIOC uses RDF as the exchange format and allows for mappings between different vocabularies, which is not envisioned in SOIF. The various Harvest subsystems are arranged in a hierarchical fashion, similar to the Domain Name System. We do not have any specified way of accessing resources in SIOC, but intend to apply database techniques for query processing and integration.

Various approaches for data integration on the Web, such as data representation languages, structural information retrieval, and query processing, are

surveyed in [4]. The survey also describes the warehousing approach to data integration that aggregates all information at one central site. However, advanced database techniques have failed so far to surface on the Web. SIOC is a first step in providing a common vocabulary for data representation across online communities. In further work, we plan to apply usable techniques from the database community to web data integration problems.

At the moment, RDF Site Summary (RSS 1.0) is widely used in weblog systems and news sites. RSS 1.0 defines a lightweight vocabulary for syndicating news items, but is used for all sorts of data exchange. Although RSS works well in practice, there are several issues: firstly, only the last “n” news items are typically exported in RSS. There is no standardised way of accessing older posts. Secondly, there is an issue with regard to updates. Different vocabularies have different update semantics: where RSS usually provides a stream of news items that should be accumulated over time, changes in FOAF files mean that the previous version should be replaced by the current. Because vocabularies can be mixed in the same file, determining what update semantics to apply for a certain file is difficult. Thirdly, although there exists a large number of extensions, none of the advanced functionality of RSS is widely deployed, since tools lack support for creating and using the extensions. RSS is widely adopted in certain areas, such as weblogs, but is not used in a wider context such as bulletin boards, mailing lists, Usenet, wikis, etc.

Also, TrackBack²¹ is a system implemented by many blogging tools that allows a weblog article to be linked to the followup articles. This is achieved by sending a summary and metadata of the new article to the weblog containing the original article, and adding this information to the original article. Linking together cross-site conversations is a step in the direction of semantically-interlinked online communities; however there are limitations to TrackBack. Firstly, it is being used in a very limited number of weblog entries and in most implementations the author has to manually enter the TrackBack address. Secondly, it only connects two individual instances of posts, not reflecting the links to the community and, in the case of archived post entries, the readers may even be unaware of the existence of this new link. Thirdly, TrackBack does not have a machine readable representation that would allow one to export its link semantics in RDF, to aggregate the resulting information and reuse it to identify related post entries.

6 Conclusion

We have presented the SIOC ontology and various mappings to and from other vocabularies that are already deployed on the Web. We have described how instance data in SIOC can be exchanged among online community sites. Our initial SIOC ontology can also be used to enable more complex use cases, for

²¹ <http://www.movabletype.org/docs/mttrackback.html>

example cross-site structural queries, and integration based on the warehousing approach.

To tackle the challenge of adoption, we have provided an upgrade path that allows a gradual migration from existing systems to semantically-enabled sites. For combination with other ontologies, we have presented mappings to and from SIOC that allow the export and import of SIOC data using existing systems and tools. We have developed a prototype SIOC exporter for a weblog engine, and several more are in development. In the future, we intend to exploit the characteristics of intra- and inter-site links to guide query routing in a P2P-like environment.

References

1. D. Beckett. The Design and Implementation of the Redland RDF Application Framework. *Computer Networks*, 39(5):577–588, 2002.
2. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1–2):119–125, 1995.
3. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International Semantic Web Conference*, pages 54–68, 2002.
4. D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.
5. A. Harth and S. Decker. Yet Another RDF Store: Complete Index Structures for Storing Semantic Web Data With Contexts. *DERI Technical Report*, 2004.
6. R. Lara, S.-K. Han, H. Lausen, M. Stollberg, Y. Ding, and D. Fensel. An Evaluation of Semantic Web Portals. In *IADIS Applied Computing International Conference 2004, Lisbon, Portugal, March 23–26, 2004*.
7. A. J. Miles, N. Rogers, and D. Beckett. SKOS Core RDF Vocabulary. 2004. <http://www.w3.org/2004/02/skos/core/>.
8. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *WWW*, pages 604–615, 2002.
9. A. Seaborne. An RDF NetAPI. In *International Semantic Web Conference*, pages 399–403, 2002.
10. K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003*, pages 131–150, 2003.

The Personal Publication Reader: Illustrating Web Data Extraction, Personalization and Reasoning for the Semantic Web

Robert Baumgartner¹, Nicola Henze², and Marcus Herzog¹

¹ DBAI, Institute of Information Systems,
Vienna University of Technology,
Favoritenstrasse 9-11, 1040 Vienna, Austria
{baumgart, herzog}@dbai.tuwien.ac.at

² ISI - Semantic Web Group, University of Hannover,
Appelstr. 4, D-30167 Hannover, Germany
henze@kbs.uni-hannover.de

Abstract. This paper shows how Semantic Web technologies enable the design and implementation of advanced, personalized information systems. We demonstrate by means of an example application how personalized content syndication can be realized in the Semantic Web. Our approach consists of two main parts: The web data extraction part, providing the information system with real-time, dynamic data, and the personalization part, which deduces - with the aid of ontological domain knowledge - personalized views on the data. The prototype of the system has been realized using the *Personal Reader Framework* for designing, implementing, and maintaining Web content Readers¹.

Keywords: semantic web, personalization, reasoning on the semantic web, web data extraction.

1 Motivation

The realization of the Semantic Web idea to be “an extension of the current web in which information is given a well-defined meaning, better enabling computers and people to work in cooperation” [5] has in only a few years pushed researchers and computer specialists to explore machine-readable semantics, appropriate markup and description languages, and sharable knowledge representation techniques. While these before mentioned techniques exist (at the writing time of this paper) as W3C recommendations, is the design of the so-called upper layers of the Semantic Web tower[4], e.g. the rule and reasoning layer, or the layers of proof and trust, still to explore.

¹ This research has been partially supported by REWERSE - Reasoning on the Web (rewerse.net), Network of Excellence, 6th European Framework Program.

In this paper, we investigate how advanced information systems for the Semantic Web can be realized. We claim that a huge class of Semantic Web-enabled information systems should be able to extract relevant information from the web, and to process and combine pieces of distributed information in such a way that the content selection and presentation fits to the current and individual needs of the user. From this viewpoint, such systems need to focus especially on the *information extraction process*, and the *personalized content syndication process*. The actual authoring process of information, and the information management processes, are important aspects, too, if we consider portal-like applications. However, there is a sustainable need of systems which can detect and process already existing Web information. To demonstrate our ideas for personalized content syndication, we consider the following scenario:

Peter is working as a researcher at a university. He publishes his research findings in journals and conferences, and also puts his publication online onto his institute's homepage. Peter is also enrolled in a research project. From time to time, he is requested to notify the project coordination office about his new publications.

The project coordination office maintains a member page where information about the members, their involvement in the project, research experience, research publications, etc. is maintained.

When we analyze the scenario, we see that

1. data about the publications is duplicated - it is stored at the university where Peter is working, but also on the Web pages of the project,
2. information about the project (people, research goals, achievements, etc.) is available online, but not related to the publications (unless somebody relates this information by hand).

The questions at hand from the scenario are:

- Can we organize this process in a way that Peter needs to publish his publications only once, e.g. at his institute's Web page? Thus that we avoid duplication of information, together with all negative side-effects like maintenance and update problems?
- Can we make use of the available contextual information on the project?
- Can we extract (relevant) information from Web pages?
- Can we combine the data in an intelligent way in order to provide a user a personally optimized access to the information? From the scenario, we may conclude that information about the role of researchers in the project like "Bob is participating mainly in working group *X*, and working group *X* is about topics *Y* and *Z*. strongly cooperating with working groups *Y* and *Z*" might be available. If we succeed in making this information available to machines to reason about, we can derive new information like: "This research paper of Bob is related to working group *X*, other papers of working group *X* on similar research questions are *A*, *B*, and *C*, etc."

This paper answers the above stated questions and demonstrates their realization within the Personal Reader framework[8, 9]. We have implemented a Personal Reader instance, the so-called *Personal Publication Reader (PPR)* which makes use of web data extraction techniques, reasoning about ontological knowledge and metadata description of informations, and provides a personal semantic view on publication data. The Personal Publication Reader has been designed and developed in the context of the Network of Excellence “REWERSE - Reasoning on the Web” and syndicates and personalizes information about the project structure, people and objectives of the REWERSE project, etc., and information about research papers in the context of the project.

The paper is organized as follows: In Section 2 we briefly outline our idea of establishing personalization services for the Semantic Web, and describe the architecture of the Personal Reader framework. The following Section 3 discusses approaches for Web Data extraction, and introduces the Lixto Suite. Section 4 then describes the realization of the Personal Publication Reader (PPR) in detail: We describe what kind of data is available via the Web (Section 4.1), and how we extract (Section 4.2), and transform it (Section 4.3) for the PPR. The domain ontology of the PPR, describing the REWERSE project, its members and research objectives, is topic of Section 4.4. Section 4.5 shows how various personalization rules derive new facts as well as personalized views on the data on top of extracted data, ontological knowledge, and user profile information. Concluding remarks and an outlook on ongoing and future work end this paper.

2 Personal Web Content Readers

Flexible information systems which need to be capable of adjusting to different application domains require a different architecture: not a monolithic approach, but several, independent components, each one serving a specific purpose. The recent Web service-technology focuses on such-like requirements: A Web service encapsulates a specific functionality, and communicates with other services or software components via interface components (e.g. [20, 15]).

We consider each (personalized) information provision task as the result of a particular service (which itself might be composed of several services, too). The aim of this approach is to construct a Plug & Play - like environment, in which the user can select and combine the kinds of information delivery services he or she prefers. With the Personal Reader Framework, we have developed an environment for designing, implementing and maintaining personal Web content Readers [8, 9]. These personal Web content Readers allow a user to browse information (the *Reader* part), and to access personal recommendations and contextual information on the currently regarded Web resource (the *Personal* part). The next section outlines briefly the architecture of the Personal Reader framework.

2.1 The Personal Reader Framework: Designing and Maintaining Personal Web Content Readers

The architecture of the Personal Reader framework is a rigorous approach for applying Semantic Web technologies. A modular framework of Web services – for constructing *the user interface*, for *mediating* between user requests and currently available personalization services, for *user modeling*, and for offering *personalization functionality* – forms the basis of each Personal Reader Instance (see Figure 1).

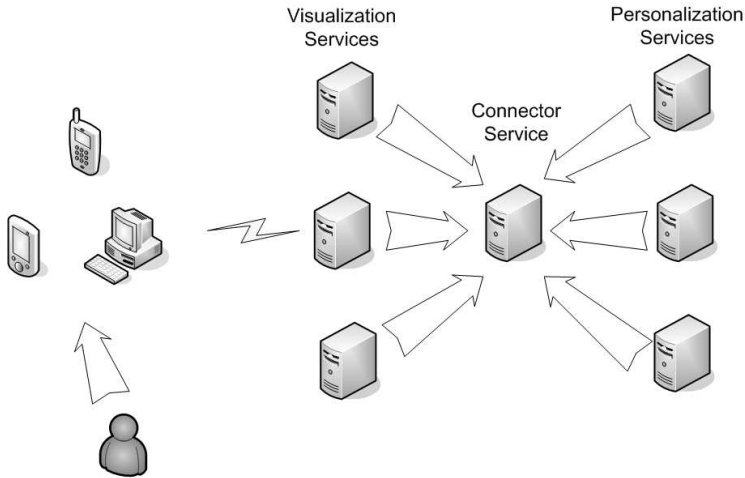


Fig. 1. Architecture of the Personal Reader framework, showing the different components of the Personal Reader: visualization, personalization, and the Personal Reader backbone (consisting of the connector service which organizes the communication and matching between the various visualization and personalization services)

The aim of the Personal Reader framework is to realize Web content Readers which give the user the possibility to select services, which provide different or extended functionality, e.g. different visualization or personalization services, and combine them into a *personal* Web content Reader instance. The framework features a distributed open architecture designed to be easily extensible. It utilizes standards such as XML[21], RDF[17], etc., and technologies like Java Server Pages (JSP)[11] and XML-based-RPC[22]. The communications between all components / services is syntactically based on RDF descriptions. This provides the required flexibility for combining various personalization and visualization services in one application, and thus supports the realization of our Plug & Play idea for personalization functionality on the Semantic Web.

2.2 Related Work on Personalized Information Systems

To the best of our knowledge, we are not aware of personalized information systems on the Semantic Web which realize the *personalization-as-service* idea in a

similar way. Personalized information systems require a sophisticated model of the actual application domain, thus, traditionally, these systems do not provide (and do not aim for) extensible architectures and systems. However, in [10], we have conducted a study on the re-usability aspects of personalization functionality, with special focus on the area of adaptive hypermedia systems. This study led to the conclusion that in fact even highly system-dependent personalization functionality like those from adaptive hypermedia research, can be encapsulated and prepared for re-use, an important precondition for the successful realization of personalization services is given.

3 Web Data Extraction and Integration

3.1 Objectives and Approaches

The *unstructured Web* of today contains millions of documents which are not query-able as a database and heavily mix layout and structure. Moreover, they are not annotated at all. There is a huge gap between Web information and the qualified, structured data as usually required in corporate information systems or as envisioned by the Semantic Web. However, until the vision of a Semantic Web is realized, and also, towards a faster achievement of this goal, it is absolutely necessary to (semi-)automatically extract relevant data from HTML document and automatically translate this data into a structured format, e.g., XML. Once transformed, data can be used by applications, stored into databases or populate ontologies.

A program that automatically extracts data and transforms it into another format or markups the content with semantic information is usually referred to as *wrapper*. Wrappers bridge the gap between unstructured information on the Web and structured databases. A number of classification taxonomies for wrapper development languages and environments have been introduced in various survey papers [6, 12, 13]. In general, it is distinguished between high-level programming languages, machine learning approaches and supervised approaches. Due to the lack of space we refer to the mentioned survey papers for an overview of available methods and tools.

3.2 Lixto Visual Wrapper

Lixto Visual Wrapper [2] is a methodology and tool for visual and interactive wrapper generation developed at the University of Technology in Vienna together with the Lixto Software GmbH. It allows wrapper designers to create so-called “XML companions” to HTML pages in a supervised way. As internal language, Lixto relies on *Elog*. Elog is a datalog-like language especially designed for wrapper generation. The Elog language operates on Web objects, that are HTML elements, lists of HTML elements, and strings. Elog rules can be specified fully visually without knowledge of the Elog language. Web objects can be identified based on internal, contextual, and range conditions and are extracted as so-called “pattern instances”.

In [7], the expressive power of a kernel fragment of *Elog* has been studied, and it has been shown that this fragment captures monadic second order logic, hence is very expressive while at the same time easy to use due to visual specification.

Besides expressiveness of a wrapping language, robustness is one of the most important criteria. Information on frequently changing Web pages needs to be correctly discovered, even if e.g. a banner is introduced. Visual Wrapper offers robust mechanisms of data extraction based on the two paradigms of tree and string extraction. Moreover, it is possible to navigate to further documents during the wrapping process. Validation alerts can be imposed that give warnings in case user-defined criteria are no longer satisfied on a page.

The usage of *Elog* is completely invisible to the average wrapper designer and all operations are carried out by visual means. This is comprised of two steps: First, the identification phase, where relevant fragments of Web pages are extracted (see Figure 2). Such extraction rules are semi-automatically and visually specified by a wrapper designer in an iterative approach. This step is succeeded by the structuring phase, where the extracted data is mapped to some destination format, e.g. enriching it with XML tags to subsequently populate an ontology with instance data.

3.3 Lixto Transformation Server

Heterogeneous environments such as integration and mediation systems require a conceptual information flow model. The usual setting for the creation of services based on Web wrappers is that information is obtained from multiple wrapped sources and has to be integrated; often source sites have to be monitored for changes, and changed information has to be automatically extracted and processed. Thus, push-based information systems architectures in which wrappers are connected to pipelines of post-processors and integration engines which process streams of data are a natural scenario, which is supported by the Lixto Transformation Server [3]. The overall task of information processing is composed into stages that can be used as building blocks for assembling an information processing pipeline. The stages are to

- acquire the required content from the source locations; this component resembles the Lixto Visual Wrapper plus Deep Web Navigation and Form iteration;
- integrate and transform content from a number of input channels and tasks such as finding differences,
- interact with external processes, and
- format and deliver results in various formats and channels and connectivity to other systems.

The actual data flow within the Transformation Server is realized by handing over XML documents. Each stage within the Transformation Server accepts XML documents (except for the wrapper component, which accepts HTML), performs its specific task (most components support visual generation of mappings), and produces an XML document as result. This result is put to the

successor components. Boundary components have the ability to activate themselves according to a user-specified strategy and trigger the information processing on behalf of the user. From an architectural point of view, the Lixto Transformation Server may be conceived as a container-like environment of visually configured information agents. The pipe flow can model very complex unidirectional information flows (see Figure 3). Information services may be controlled and customized from outside of the server environment by various types of communication media such as Web services. The Transformation Server includes a user management that allows application designers to subscribe and parameterize components of other application designers.

4 The Personal Publication Reader

To realize the Personal Publication Reader (PPR) within the Personal Reader framework (see Section 2), we extract the publication information from the various Web sites of the partners in the REWERSE project: All Web pages containing information about publications of the REWERSE network (see Section 4.1) are periodically crawled and new information is automatically detected, extracted and indexed in the repository of semantic descriptions of the REWERSE network (see Sections 4.2, 3.3, 4.3). Information on the project REWERSE, on people involved in the project, their research interests, and on the project organization, is modeled in an ontology for REWERSE (see Section 4.4). Extracted information and ontological knowledge are used to derive a syndicated view on each publication: who has authored it, which research groups are related to this kind of research, which publications are published by the research group, which publications are on the similar research, etc. Information about the current user of the system (such as specific interests of the user, or his membership to the project) is used to individualize the view on the data (see Section 4.5).

4.1 Publication Data on the Web

In this scenario we are in particular interested to give a personalized view on publications of the members of the REWERSE network of excellence. Therefore, the ontology of the Personal Publication Reader has to be populated with instance data from publication sources. In most of the cases, the organizations offer access to their publications through a Web interface. However, each Web presentation is totally different, some use e.g. automatic conversions of *bibtex* or other files, some are manually maintained, some are based on databases. Such a presentation is well suited for human consumption, but hardly usable for automatic processing. Nevertheless, the Web is the most valuable information resource in this scenario. In order to access and understand these heterogeneous information sources one has to apply web extraction techniques as described in Section 3.

In Table 1 selected REWERSE members are given and their publication format is described. The table explains how the publications are structured, and

Table 1. Publication Web pages of selected REWERSE members

<i>Participant</i>	<i>Structure and Presentation</i>
Munich	http://www.pms.informatik.uni-muenchen.de/publikationen all publications on a single page sorted by years (latest on top), auto-generated format, usage of HTML elements inside publications, even for individual authors, links and bibtex available
Hannover	http://www.kbs.uni-hannover.de/Stamm/Publikationen.html all publications on a single page sorted by years (newest on top, publications numbered by years), publications consistent (some formatted differently), data very complete, usage of HTML elements inside publications, links available
Heraklion	http://www.ics.forth.gr/publications.jsp publications on multiple pages structured by years; additional structuring with next links, sites and publications consistent, data very complete, usage of HTML elements inside publications, links and abstracts available
Linköpping	http://www.ida.liu.se/ext/dpr/access2/ publications on multiple pages structured by years, sites and publications not consistent, usage of HTML elements inside publications, links on selected authors, publications numbered

how the format of a single publication looks like. Moreover, it describes whether at least some parts of a single publication are rendered via HTML elements (such as italics for the title). For most member sites it holds that even if HTML elements are used usually authors are merely separated by commas.

Furthermore, the table indicates whether additional information to author, title, and year are available and how complete the information is (if e.g. year or conference is missing). The least common denotator for all member pages are the availability of author names, title name and publication year, in some cases additionally abstracts and links are available.

4.2 Gathering Web Data for the Personal Publication Reader

In the following, we describe a step-by-step construction of this example from the viewpoint of an application designer who creates this application.

A human being tends to assign semantic meaning to parts of a Web page; a designer does not think of *table row* as of a set with text values, but rather as a *publication entry*. Therefore, the basic building block of a wrapper program is a so-called *pattern*, a container for pieces of information with the same meaning. Patterns are structured in a hierarchical fashion. In the lower half of the Visual Wrapper's UI (see Figure 2) an active example Web page is displayed for marking example instances: For each type of Web page, an own wrapper has to be created; in the following the wrapper creation for the publications of Munich is illustrated.

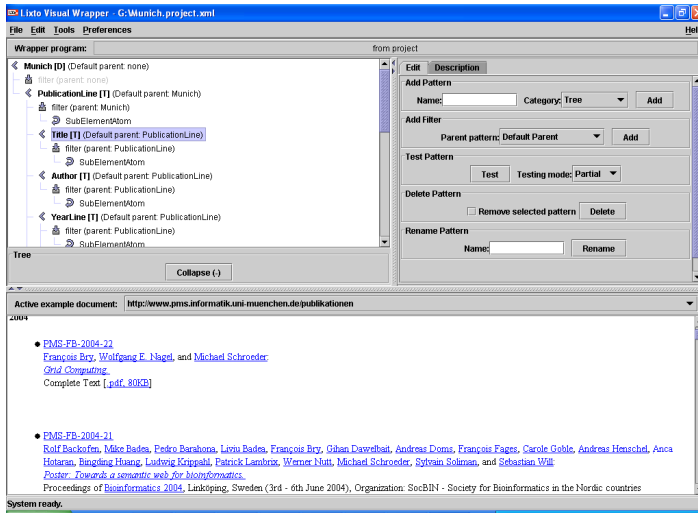


Fig. 2. Lixto Visual Wrapper: Wrapping Publication Pages

In this case, the designer identifies one of the list items (each resembling a publication) as a pattern *PublicationLine*. Once a pattern is created, the designer continues with visually defining a filter, a crucial part of the pattern which defines how to extract relevant information from its parent pattern instances. Internally, filters are represented in Elog, but the language is entirely hidden from the wrapper designer.

Defining a filter expects the designer to select an example publication with two mouse clicks on the example Web page. A filter definition continues with optional fine-tuning of properties for the generated generalization of the chosen example. It is possible to visually debug the wrapper program, i.e., to test filters. Typically, operators test filters after adding new components. Based on results, the designer decides whether to extend (i.e., add a filter) or shrink (i.e., add condition to an existing filter) the set of matched instances.

In this example, the system displays the complete list of matched publications for the so-far created filter by highlighting parts of the Web page. In cases where the system generalization does not detect all instances correctly, additional conditions can be imposed.

Next a child pattern *Title* of the just defined pattern is created and then a filter with the condition that the extracted element is in italics. The pattern *Author* on the Munich page can be easily characterized, too, by the fact that a special hyperlink is present and that the author names precede the title.

On other pages such as e.g. Linköpping the extraction of authors is more advanced. Some authors are inside hyperlinks, others merely separated by commas. Moreover, on other sources authors are sometimes incorrectly splitted, names abbreviated and different separators used. Therefore, we developed an author concept based on all detected variations.

On the Munich page the year can be extracted from several places (see Figure 2). One possibility is from the internal number. The first line of the list item is extracted, and in a subsequent step the four digit number is taken out. On some other sources the year has to be extracted from the headline, and in a subsequent step mapped to each entry.

In a similar fashion the remaining patterns are defined and the wrapper is stored. The XML Companion of the publication Web page that can be regularly generated by applying the wrapper is comprised of entries like the one given below:

```
<Publication>
  <Title>Visual Exploration and Retrieval of XML Document
    Collections with the Generic System X2</Title>
  <Author>Holger Meuss</Author>
    [...more authors...]
  <Year>2004</Year>
  <Link>http://www.pms.informatik.uni-muenchen.de/
    publikationen/PMS-FB/PMS-FB-2004-12.pdf</Link>
</Publication>
```

As next step the XML data of the various sources has to be combined, cleaned, syndicated into the ontology, and regularly scheduled. These operations are carried out by configuring a visual information flow in the *Lixto Transformation Server* as described in Section 4.3.

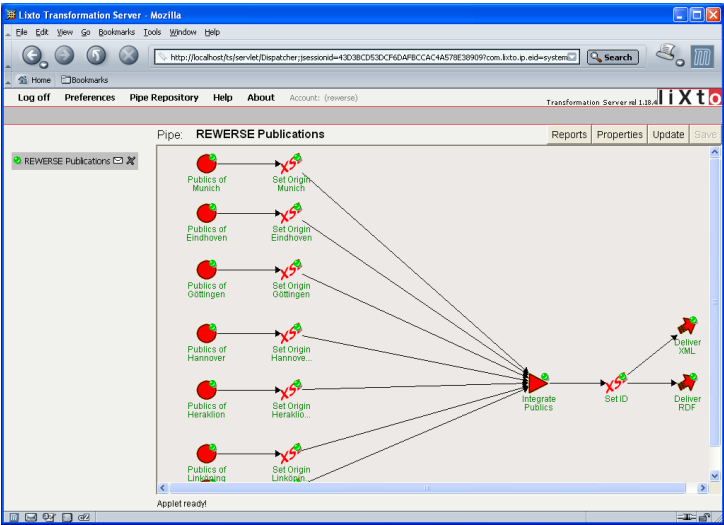


Fig. 3. Lixto Transformation Server: REVERSE Publication Data Flow

4.3 Visual Data Aggregation for the Personal Publication Reader

In the Personal Publication Reader scenario, the application designer visually composes the information flow from Web sources using the Lixto Transformation Server to an RDF presentation that is handed over to the Personal Publication Reader once a week.

First, the application designer creates Source components that contain Lixto wrappers. In the source components (that are reflected as disks in Figure 3) a schedule is defined how often which Web source is queried and Deep Web navigation sequences containing logins and forms can be stored. Next, the wrapper designer can combine the XML documents by adding integration components.

In the “XSL” components publication data is harmonized to fit into a common structure, an attribute “origin” is added containing the institution’s name, and author names are harmonized by being mapped to a list of names known by the system. The triangle in Figure 3 represents a data integration unit; here data from the various institutions is put together and duplicate entries are removed. IDs are assigned to each publication in the subsequent step. Finally, the XML data structure is mapped to a defined RDF structure (this happens in the lower arc symbol in Figure 3) and passed on to the Personal Publication Reader as described below. A second deliverer component delivers the XML publication data additionally. One sample RDF output entry is depicted below:

```
<rdf:Description
  rdf:about="http://www.pms.informatik.uni-muenchen.de/
  publikationen/PMS-FB/PMS-FB-2004-12.pdf">
<dc:publisher>University of Munich</dc:publisher>
<dc:title>Visual Exploration and Retrieval of XML Document
  Collections with the Generic System X2</dc:title>
<dc:creator>
  <rdf:Seq>
    <rdf:li rdf:resource="#Holger Meuss"/>
    <rdf:li rdf:resource="#Klaus U. Schulz"/>
    <rdf:li rdf:resource="#Felix Weigel"/>
    <rdf:li rdf:resource="#Simone Leonardi"/>
    <rdf:li rdf:resource="#Francois Bry"/>
  </rdf:Seq>
</dc:creator>
<dc:date>2004</dc:date>
<dc:identifier>http://www.pms.informatik.uni-muenchen.de/
  publikationen/PMS-FB/PMS-FB-2004-12.pdf</dc:identifier>
</rdf:Description>
```

This application can be easily enhanced by connecting further Web sources. For instance, abstracts from www.researchindex.com can be queried for each publication lacking this information and joined to each entry, too. Moreover, using text categorization tools one can rate and classify the contents of the abstracts. Another possibility is to extract organization and people data from the institution’s Web pages to inform the ontology to which class in the taxonomy an author belongs (such as full professor).

4.4 Modeling Domain Knowledge: The REWERSE Ontology

In addition to the extracted information on research papers that we obtain as described in the previous section, we collect the data about the members of the research project from the member's corner of the REWERSE project. We have constructed an ontology for describing researchers and their involvement in scientific projects like REWERSE. This "REWERSE-Ontology" has been built using the Protégé tool [16]. It extends the Semantic Web Research Community Ontology (SWRC) [19]. An excerpt of the REWERSE-Ontology, written in OWL[14]:

```
<owl:ObjectProperty rdf:ID="hasStaffMember">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasMember"/>
  </rdfs:subPropertyOf>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="employedAt"/>
  </owl:inverseOf>
  <rdfs:label xml:lang="de">Angestellte</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#University"/>
        <owl:Class rdf:about="#Institute"/>
        <owl:Class rdf:about="#Project"/>
        <owl:Class rdf:about="#Department"/>
        <owl:Class rdf:about="#Company"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#Person"/>
  <rdfs:label xml:lang="en">Staffmember</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#employedAt">
  <rdfs:label xml:lang="en">employed at</rdfs:label>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Project"/>
        <owl:Class rdf:about="#Institute"/>
        <owl:Class rdf:about="#University"/>
        <owl:Class rdf:about="#Department"/>
        <owl:Class rdf:about="#Company"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:subPropertyOf rdf:resource="#involvedIn"/>
  <rdfs:label xml:lang="de">angestellt bei</rdfs:label>
  <rdfs:domain rdf:resource="#Person"/>
  <owl:inverseOf rdf:resource="#hasStaffMember"/>
</owl:ObjectProperty>
```


To match the domain knowledge in the REVERSE Researcher Ontology to the extracted publication data, we have a resource identification problem. The author names may vary - for example, F. Bry, François Bry, Prof. F. Bry, etc. . A “helper” ontology, describing the full name of each author, and a variety of commonly used designators of his or her name, is currently used to solve this matching task.

4.5 Content Syndication and Personalized Views

As we have described in the previous sections, we have extracted relevant data from various, non-uniform Web sites, and created an extension of the SWRC ontology to model the needs of scientific projects such as REVERSE. We will now see how personalization rules reason about this collected data in order to syndicated and personalize the view on the data. A discussion on personalization reasoning for the Semantic Web can be found in [1]. As an example, the following rule (using the TRIPLE[18] syntax) determines all authors of a publication:

```
FORALL A, P all_authors(A, P) <-
  EXISTS X, R (
    P['http://.../reverse#':author -> X]@'http:...#':publications
    AND X[R -> 'http://www.../author':A]@'http:...#':publications).
```

Further rules combine information on these authors from the researcher ontology with the author information. E.g. the following rule determines the employer of a project member, which might be a company, or a university, or, more generally, some instance of a subclass of an organization:

```
FORALL A,I works_at(A, I) <-
  EXISTS A_id,X (name(A_id,A)
    AND ont:A_id[ont:involvedIn -> ont:I]@'http:...#':researcher
    AND ont:X[rdfs:subClassOf ->
      ont:Organization]@rdfschema('http:...#':researcher)
    AND ont:I[rdf:type -> ont:X]@'http:...#':researcher).
```

For a user with specific interests, for example “interest in personalized information systems”, information on respective research groups in the project, on persons working in this field, on their publications, etc., is syndicated. As an example, the following rule derives all persons working in specific working groups in the project. Personalization is realized by matching the results of this rule with the individual request, e.g. `ont:WG[ont:name -> 'WG A3 - Personalized Information Systems']`.

```
FORALL WG,M working_group_members(WG,M) <-
  ont:WG[rdf:type -> ont:WorkingGroup]@'http:...#':researcher
  AND ont:WG[ont:hasMember-> ont:M]@'http://...#':researcher.
```

For the PPR, we instantiated a personalization service in the Personal Reader framework which holds the above mentioned rules, and further personalization rules of the PPR. An appropriate visualization service for creating the user interface has been implemented. The screenshot in Figure 4 depicts the output of the visualization service of the PPR.

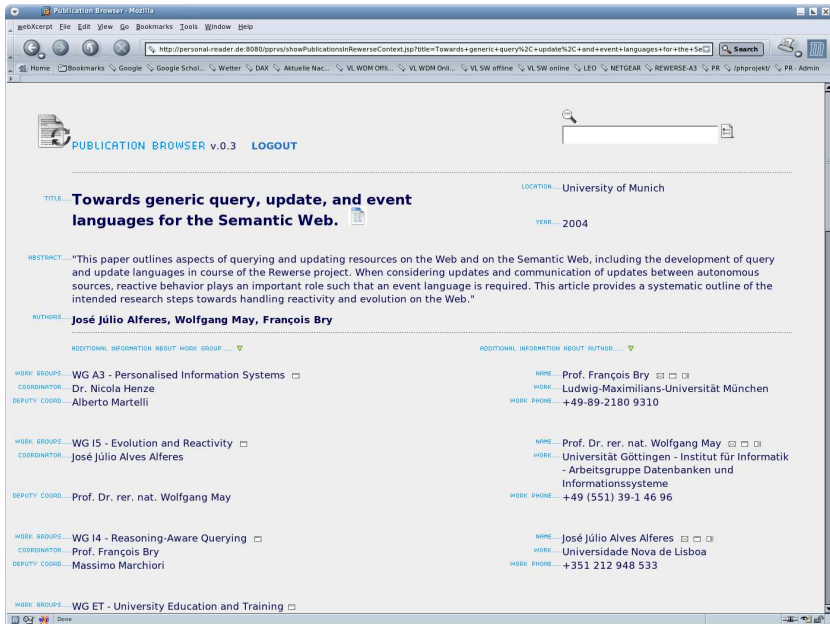


Fig. 4. Screenshot of the Personal Publication Reader, showing the syndicated view on publications in REVERSE, the context in the project in which this research has been done, together with the appropriate links, and additional information about the authors of the publication like homepage, phone number, etc. The Personal Publication Reader is available via the URL www.personal-reader.de

5 Conclusion

This paper describes an approach for realizing advanced personalized information systems in the Semantic Web. We discuss our approach by means of an example application, a Personal Publication Reader, which provides a personalized, syndicated view on distributed, non-uniform web data. The information provision part for the Personal Publication Reader is solved by using the Lixto approach. *Lixto* is an easily accessible technology based on a solid theoretical framework [2, 3, 7] and a visual approach that allows application designers to define continuously running information agents fetching data from the Web. Many functions that will be tangible only in the future “Semantic Web” can be crucially supported by the usage *Lixto*. Content syndication and personalization is achieved by reasoning about ontological knowledge and extracted Web data. The Personal Publication Reader is realized using the Personal Reader Framework for designing, implementing, and maintaining personalized Web Content Readers. Until now, we have realized such Readers for e-Learning and for publication browsing, ongoing work focuses on implementing additional personalization services, and on improving the service orchestration functionality in our frame-

work. In future work we will continue our approach of realizing *Personalization Services for the Semantic Web*.

References

1. G. Antoniou, M. Baldoni, C. Baroglio, R. Baumgartner, F. Bry, T. Eiter, N. Henze, M. Herzog, W. May, V. Patti, S. Schaffert, R. Schindlauer, and H. Tompits. Reasoning methods for personalization on the semantic web. *Annals of Mathematics, Computing & Teleinformatics*, 2(1):1–24, 2004.
2. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Proc. of VLDB*, 2001.
3. R. Baumgartner, M. Herzog, and G. Gottlob. Visual programming of web data aggregation applications. In *Proc. of IIWeb-03*, 2003.
4. T. Berners-Lee. The semantic web - mit/lcs seminar, 2002. <http://www.w3c.org/2002/Talks/09-lcs-sweb-tbl/>.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
6. S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications Vol.17/2*, 2004.
7. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web Information Extraction. In *Proc. of PODS*, 2002.
8. N. Henze and M. Herrlich. The Personal Reader: A Framework for Enabling Personalization Services on the Semantic Web. In *Proceedings of the Twelfth GI-Workshop on Adaptation and User Modeling in Interactive Systems (ABIS 04)*, Berlin, Germany, 2004.
9. N. Henze and M. Kriesell. Personalization functionality for the semantic web: Architectural outline and first sample implementation. In *Proceedings of the 1st International Workshop on Engineering the Adaptive Web (EAW 2004)*, co-located with *AH 2004*, Eindhoven, The Netherlands, 2004.
10. N. Henze and W. Nejdl. A logical characterization of adaptive educational hypermedia. *New Review of Hypermedia*, 10(1), 2004.
11. SUN - java Server Pages, 2004. <http://java.sun.com/products/jsp/>.
12. S. Kuhllins and R. Tredwell. Toolkits for generating wrappers. In *Net.ObjectDays*, 2002.
13. A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. In *Sigmod Record 31/2*, 2002.
14. OWL, Web Ontology Language, W3C Recommendation, Feb. 2004. <http://www.w3.org/TR/owl-ref/>.
15. OWL-S: Web Ontology Language for Services, W3C Submission, Nov. 2004. <http://www.org/Submission/2004/07/>.
16. Protege Ontology Editor and Knowledge Acquisition System, 2004. <http://protege.stanford.edu/>.
17. RDF Vocabulary Description Language 1.0: RDF S, 2004. <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>.
18. M. Sintek and S. Decker. TRIPLE - an RDF Query, Inference, and Transformation Language. In I. Horrocks and J. Hendler, editors, *International Semantic Web Conference (ISWC)*, pages 364–378, Sardinia, Italy, 2002. LNCS 2342.

19. SWRC - Semantic Web Research Community Ontology, 2001. <http://ontobroker.semanticweb.org/ontos/swrc.html>.
20. WSDL: Web Services Description Language, version 2.0, Aug. 2004. <http://www.w3.org/TR/2004/WD-wsdl20-20040803/>.
21. XML: extensible Markup Language, 2003. <http://www.w3.org/XML/>.
22. XML-based RPC: Remote procedure calls based on xml, 2004. <http://java.sun.com/xml/jaxrpc/index.jsp>.

Generating Tailored Textual Summaries from Ontologies

Kalina Bontcheva*

Department of Computer Science,
University of Sheffield, Regent Court,
211 Portobello Street, Sheffield, UK
`kalina@dc.shef.ac.uk`

Abstract. This paper presents the ONTOSUM system which uses Natural Language Generation (NLG) techniques to produce textual summaries from Semantic Web ontologies. The main contribution of this work is in showing how existing NLG tools can be adapted to Semantic Web ontologies, in a way which minimises the customisation effort while offering more diverse output than template-based ontology verbalisers. A novel dimension of this work is the focus on tailoring the summary formatting and length according to a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping for summary generation from different ontologies.

1 Introduction

The Semantic Web aims to add a machine tractable, re-purposeable¹ layer to compliment the existing web of natural language hypertext. In order to realise this vision, the creation of semantic annotation, the linking of web pages to ontologies, and the creation, evolution and interrelation of ontologies must become automatic or semi-automatic processes.

Natural Language Generation² (NLG) takes structured data in a knowledge base as input and produces natural language text, tailored to the presentational context and the target reader [8]. NLG techniques use and build models of the context and the user and use them to select appropriate presentation strategies. For example, deliver short summaries to the user's WAP phone or a longer multimodal text if the user is using their desktop.

In the context of Semantic Web or knowledge management, NLG can be applied to provide automated documentation of ontologies and knowledge bases.

* This work is partially supported by the EU-funded SEKT (<http://sekt.semanticweb.org>) and KnowledgeWeb (<http://knowledgeweb.semanticweb.org>) projects.

¹ Re-purposeable in this case meaning useful in a number of different applications, i.e. application-independent.

² For an in-depth introduction to NLG see [8].

Unlike human-written texts, an automatic approach will constantly keep the text up-to-date which is vitally important in the Semantic Web context where knowledge is dynamic and is updated frequently. The NLG approach also allows generation in multiple languages without the need for human or automatic translation (see [1]). This is an important problem firstly because textual documentation is more readable than the corresponding formal notations and thus helps users who are not knowledge engineers to understand and use ontologies. Secondly, a number of applications have now started using ontologies to encode and reason with internally, but this formal knowledge needs to be also expressed in natural language in order to produce reports, letters, etc. In other words, NLG can be used to present structured information in a user-friendly way.

There are several advantages to using NLG rather than using fixed templates where the query results are filled in:

- NLG can use different sentence structures depending on the number of query results, e.g., conjunction vs itemised list.
- depending on the user's profile of their interests, NLG can include different types of information – affiliations, email addresses, publication lists, indications on collaborations (derived from project information).
- given this variety of what information from the ontology can be included and how it can be presented, depending on its type and amount, writing templates will be unfeasible because there will be too many combinations to be covered.

This variation comes from the fact that it is expected that each user of the system will have a profile comprising of user supplied (or system derived) personal information (name, contact details, experience, projects worked on), plus information derived semi-automatically from the user's interaction with other applications. Therefore, there will be a need to tailor the generated presentations according to user's profile.

NLG systems that are specifically targeted towards Semantic Web ontologies have started to emerge only recently. For example, there are some general purpose ontology verbalisers for RDF and DAML+OIL [12] and OWL [11]. They are based on templates and follow closely the ontology constructs, e.g., "*This is a description of John Smith identified by [http://...His given name is John...](\"http://...\")*" [11]. The advantages of Wilcock's approach [12, 11] is that it is fully automatic and does not require a lexicon. A more recent system which generates reports from RDF and DAML ontologies is MIAKT [3]. In contrast to Wilcock's approach, MIAKT [3] requires some manual input (lexicons and domain schemas), but on the other hand it generates more fluent reports, oriented towards end-users, not ontology builders. It also uses reasoning and the property hierarchy to avoid repetitions, enable more generic text schemas, and perform aggregation.

At the other end of the spectrum are sophisticated NLG systems such as TAILOR [7], which offer tailored output based on user/patient models. Systems like Wilcock's [11] and MIAKT [3] tend to adopt simpler approaches, exploring generalities in the domain ontology, because their goal is to lower the effort for

customising the system to new domains. Sophisticated systems, while offering more flexibility and expressiveness, are difficult to adapt by non-NLG experts. For example, experience in MIAKT showed that knowledge management and Semantic Web ontologies tend to evolve over time, so it is essential to have an easy-to-maintain NLG approach.

This work extends the MIAKT approach towards making it less domain dependent and easier to configure by non-NLG experts. A novel dimension is the focus on tailoring the summary formatting and length according to a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping for summary generation from different ontologies.

The paper is structured as follows. Section 2 introduces the ONTOSUM system and its architecture. Next Section 3 focuses on portability and customisation, including an extended example, using an existing Semantic Web ontology. The formatting and length tailoring algorithms are discussed in Section 4. The paper concludes with a discussion of future work.

2 System Architecture

Since ONTOSUM is designed to be part of interactive applications, it needs to *(i)* respond to user requests in *real-time*, i.e., avoid generation algorithms with associated high computational cost; and *(ii)* be *robust*, i.e., always produce a response. Consequently the system uses some efficient and well-established applied NLG techniques such as text schemas and a phrasal lexicon (see [8]).

The ONTOSUM system is implemented as a set of components in the GATE infrastructure [2], which provides an easy-to-use graphical development environment for NLP systems. In particular, we make use of its ontology support, which provides language-independent access to ontologies. The advantage of this approach is that our generator can handle RDF, DAML+OIL, and OWL, without any modifications, as it uses the format-independent GATE API, rather than format-specific ones.

Similar to other applied NLG systems (see [8]), ONTOSUM is implemented as pipeline system, i.e., the generation modules are executed sequentially.

Summary generation starts off by being given a set of statements (i.e., triples), in the form of RDF/OWL. Since there is some repetition, these triples are first pre-processed to remove already said facts. In addition to triples that have the same property and arguments, the system also removes triples involving inverse properties with the same arguments, as those of an already verbalised one. The information about inverse properties is provided by the ontology (if supported by the representation formalism).

Next is the summary structuring module, which orders the input statements in a coherent summary. This is done using discourse patterns, which are applied recursively and capitalise on the property hierarchy (see Section 3.2). This module also performs *semantic aggregation*, i.e., it joins together statements with the same property name and domain, so they are expressed within one sentence (see Section 3.3).

Finally, the generator transforms statements from the ontology into conceptual graphs [10] which are then verbalised by the HYLITE+ surface realiser [3]. The output is a textual summary (see Figure 3).

A similar approach was first implemented in a domain- and ontology-specific way in the MIAKT system [3]. In ONTOSUM we extended it towards portability and personalisation, i.e., lowering the cost of porting the generator from one ontology to another and generating summaries of a given length and format, dependent on the user target device. These issues are discussed in detail next.

3 Portability and User-Friendliness

Natural Language Generation (NLG) systems consist of two types of components: domain-dependent and domain-independent ones. Typically the text structuring component is domain-dependent, because every domain or application tends to have different conventions for what constitutes a coherent text. Another example domain-dependent module is the lexicon which maps concepts to their lexical items and grammatical information. Therefore, when an NLG system is adapted to a new domain or application, these components need to be modified.

In contrast, the surface realisation module, i.e., the module that generates the sentences, given their formal syntactic structure, is typically domain-independent and does not need to be adapted.

Therefore, this section will focus on the lexicalisation and summary structuring modules in ONTOSUM, while the HYLITE+ surface realiser will not be covered, as it is domain-independent (see [3]).

3.1 Lexicalisations of Concepts and Properties

The lexicalisations of concepts and properties in the ontology can be specified by the ontology engineer, be taken to be the same as concept names themselves, or added manually as part of the customisation process. For instance, the AKT ontology³ provides `label` statements for some of its concepts and instances, which are found and imported in the lexicon automatically:

```
<daml:DatatypeProperty rdf:ID="has-email-address">
  <rdfs:label>has email address</rdfs:label>
  <rdfs:range rdf:resource="&xsd:string"/>
  <rdfs:isDefinedBy rdf:resource="&base;"/>
</daml:DatatypeProperty>
```

The generator is parameterised at run time by specifying which properties are to be used for building the lexicon, e.g., `label`, `name` (in the SWRC ontology⁴) (see the `propertyNames` parameter in Figure 2).

³ <http://www.aktors.org/ontology/>

⁴ <http://ontoware.org/projects/swrc/>

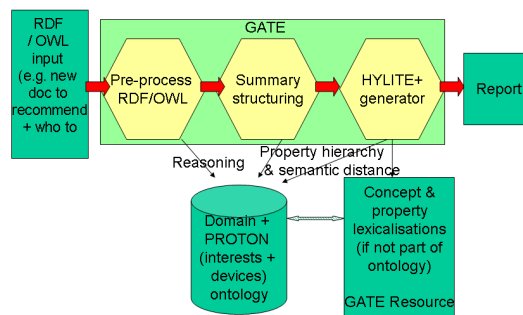


Fig. 1. ONTOSUM's Architecture

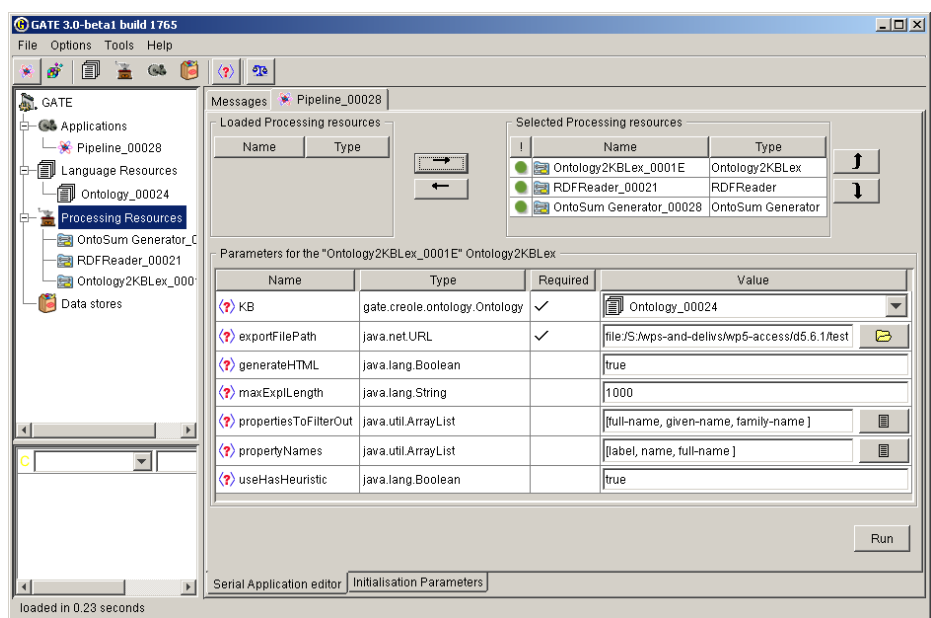


Fig. 2. Example ONTOSUM Configuration Parameters

The lexicon is thus generated automatically from the ontology. By default concepts are assumed to be lexicalised as nouns and properties as verbs. This is a rather strong simplification, but given that it is true in many cases, it does save the user the effort of having to specify these manually for the entire ontology. Instead, the user only needs to verify that the automatically assigned part of speech is correct and only change the exceptions. The lexical entries are in the format <Concept-Name, Lexicalisation, GrammaticalFeatures>. The grammatical features are a list of attribute-value pairs, e.g., **pos** – part-of-speech (noun, verb, adjective, etc.), **num** – number (singular or plural), **massnoun** – value is **true** if this is a mass noun, i.e., uncountable nouns like water. Some sample entries are shown below:

```
lex_entry_eng('System-Administrator',
              'system administrator', [fs(pos,noun),fs(num,sing)]).
lex_entry_eng('Generalised-Means-Of-Transport',
              'Generalised-Means-Of-Transport', [fs(pos,noun),fs(num,sing)]).
```

In this example, the lexicalisation of the first entry was taken from its `label` property, but the second entry did not have such a property, so the concept name was assigned as a lexicalisation. The user can then edit the name as they edit the part of speech and other grammatical information.

3.2 Summary Structuring

Discourse/text schemas, as introduced by [5], are script-like structures which represent discourse patterns. They can be applied recursively to generate coherent multisentential text satisfying a given, high-level communicative goal.⁵ Each schema consists of rhetorical predicates (e.g. *comparison*, *constituency*) which encode communicative goals and structural relations in the text. Rhetorical predicates are also associated with a semantic function which selects appropriate statements from the ontology. In this way, by selecting and instantiating schemas, a text structuring component can produce coherent texts which satisfy given communicative goals.

In more concrete terms, when given a set of statements about a given concept/instance, discourse schemas are used to impose an order on them, such that the resulting summary is coherent. For the purposes of our system, a coherent summary is a summary where similar statements are grouped together.

The top-level schema for describing instances from the ontology is:

```
Describe-Instance ->
  Describe-Attributes,
  Describe-Part-Wholes,
  Describe-Active-Actions,
  Describe-Passive-Actions
```

where `Describe-Attributes`, etc. are recursive calls to other schemas. For example, the `Describe-Attributes` schema collects recursively all properties that are sub-properties of the `attribute-property` and involve the given instance:

```
Describe-Attributes ->
  [attribute(Instance, Attribute)],
  Describe-Attributes *
```

The schemas are independent of the concrete domain and rely only on a core set of 4 basic properties – `active-action`, `passive-action`, `attribute`, and `part-whole`. When a new ontology is connected to ONTOSUM, properties can be defined as a sub-property of one of these 4 generic ones and then ONTOSUM will be able to verbalise them without any modifications to the discourse

⁵ For instance, (`definition AIRCRAFT CARRIER`) – generate text that defines aircraft as a type of carrier – (example D, [5-p.44]).

schemas. However, if more specialised treatment of some properties is required, it is possible to enhance the schema library with new patterns, that apply only to a specific property.

Since most ontologies do not have these 4 properties in their property hierarchy (e.g., AKT ontology⁶, SWRC ontology⁷), we implemented a heuristic for recognising attributive properties as a way of lowering the adaptation effort. If this heuristic is enabled, the generator considers all properties with names starting with **has** as attribute properties. All other properties need to be classified manually according to one of these 4 basic types or a lexicalisation and a new discourse schema need to be provided.

Once the information from the ontology is structured using the schemas, aggregation is performed to join similar RDF triples. This process joins adjacent triples that have the same first argument and have the same property name or if they are sub-properties of **attribute** or **part-whole** properties. For example, in the summary in Figure 3 we have 4 triples with the same first argument (the researcher) and all properties are attribute properties. Therefore, they are joined together as one proposition.

Without this aggregation step, there will be four separate sentences, resulting in a less coherent text:

Kalina Bontcheva has a Dr appellation. Kalina Bontcheva has email K.Bontcheva@dcs.shef.ac.uk. Kalina Bontcheva has web page <http://www.dcs.shef.ac.uk/kalina/>. Kalina Bontcheva has telephone number +4401142221930.

3.3 Extended Example

This section provides a step-by-step example first of how to customise ONTOSUM for an ontology, and next – of the generation process itself. The two processes can be repeated iteratively, i.e., the user can carry out some customisation, run ONTOSUM, analyse the problems, then go back to editing the lexicon or the ontology, etc.

The first stage in connecting an ontology to ONTOSUM is to execute the **Ontology2KBLex** component (see Figure 2) which generates the domain lexicon, as discussed in Section 3.1. For example, this would create the following entry, derived from the **full-name** property, which is one of the two property names (see Figure 2) used in the automatic creation of the lexical entries:

```
lex_entry_eng('K.Bontcheva.dcs.shef.ac.uk', 'Kalina Bontcheva',
              [fs(pos,noun)]).
```

Once the lexicon has been completed, the next (optional) step is to introduce in the ontology property hierarchy the four linguistically-motivated properties discussed in Section 3.2. If the **has** heuristic has been enabled in

⁶ <http://www.aktors.org/ontology/>

⁷ <http://ontoware.org/projects/swrc/>

Ontology2KBLex, then some of the properties are classified automatically as attributive on the basis of their names.

In addition, if the `propertiesToFilterOut` parameter has been set, an ONTOSUM configuration file with this information is created automatically. This parameter enables the user to specify properties which should be filtered out from the textual summaries (see below for more detail).

At this stage, ONTOSUM is ready to be run on a given RDF/OWL description of an instance to produce its natural language summary. It is the responsibility of the application which calls ONTOSUM to choose which instance is to be described, i.e., to provide the RDF/OWL input. In the simplest case, this could be the user browsing the ontology, clicking on an instance, and asking for its textual summary.

For the sake of simplicity, throughout this example we will assume that there are no length restrictions for the summary and it will be generated as plain text. These issues are addressed in Section 4 next.

In this example we will use the AKT ontology and the RDF description of the author, part of which appears below:

```
<rdf:Description rdf:about="http://...#K.Bontcheva.dcs.shef.ac.uk">
  <ns0:family-name>Bontcheva</ns0:family-name>
  <ns0:full-name>Kalina Bontcheva</ns0:full-name>
  <ns0:given-name>Kalina</ns0:given-name>
  <ns0:has-appellation>Dr</ns0:has-appellation>
  <ns0:has-email-address>K.Bontcheva@dcs.shef.ac.uk</ns0:has-...>
  ...
  <rdf:type rdf:resource="http://...#Researcher-In-Academia"/>
</rdf:Description>
```

As shown in Figure 1, the first phase is *pre-processing*. Let us assume that there were no previous explanations, so no properties of this instance need to be removed to avoid repetition. During pre-processing ONTOSUM also removes properties given as values of the `propertiesToFilterOut` parameter of the Ontology2KBLex module. The function of this parameter is enable the user to exclude some information from the summary. For example, properties encoding the provenance of this instance or providing lexical information (e.g., full-name) may be excluded in this way.

In our example (see Figure 2), `full-name`, `family-name`, and `given-name` are specified as properties to be filtered out. Consequently, at the end of the pre-processing phase the input is transformed into:

```
<rdf:Description rdf:about="http://...#K.Bontcheva.dcs.shef.ac.uk">
  <ns0:has-appellation>Dr</ns0:has-appellation>
  <ns0:has-email-address>K.Bontcheva@dcs.shef.ac.uk</ns0:has-...>
  ...
  <rdf:type rdf:resource="http://...#Researcher-In-Academia"/>
</rdf:Description>
```

The next phase is *summary structuring*. Here we will consider two alternatives: one where the **has** heuristic has been enabled and one where it was disabled.

When the heuristic is enabled, the system would know which properties of the given instance are attribute properties, because their names start with **has**. Therefore the user would not need to specify their lexicalisations and the existing ONTOSUM schemas can be applied to order the triples. As we took a simple example containing only attribute properties, their order will not change, i.e., will remain as it was in the original input. However, as they are the same type of property, they will be aggregated into one semantic relation (ATTR) with several values – one for each property value. If there were other property types, then more semantic relations will be created and ordered according to the discourse schemas described in Section 3.2.

```
ATTR(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
  [ Appellation: Dr,
    string: K.Bontcheva@dcs.shef.ac.uk,
    string: +4401142221930,
    string: http://www.dcs.shef.ac.uk/~kalina/
  ]
)
```

The information that **Dr** is a value of type **Appellation** and email, telephone, and URL are strings comes from the range restrictions in the property definitions in the ontology, e.g., see the **has-email-address** definition in Section 3.1. Also, the lexical entry for **K.Bontcheva.dcs.shef.ac.uk** is used instead of the unique identifier from the ontology.

Given this input, the HYLITE+ generator will verbalise it as:

```
Kalina Bontcheva has a Dr appellation, K.Bontcheva@dcs.shef.,
http://www.dcs.shef.ac.uk/ kalina/, and +4401142221930.
```

The problem with this summary comes from the fact that the ontology engineer decided to encode some of the information about researchers using classes (e.g., **Appellation**), while the rest is encoded as datatype properties with range **string**. Since this is not an ontology class, the generator only provides its value (e.g., **+4401142221930**), but it lacks the information that this is a telephone number, as this is only encoded implicitly in the property name – **has-telephone-number**.

One solution is to modify the ontology by introducing the required classes (**Email**, **PhoneNumber**, etc.) and changing the property ranges from **string** to these new classes. This would have the benefit of making explicit the semantics of these properties and their values. However, it may not always be desirable to modify the ontology.

The second solution is to provide the generator with manually written mapping rules which map the ranges of given properties to their lexical classes, e.g., **lex-mapping(has-email, string, email)**. Then, using these mappings, ONTOSUM will produce instead:

```

ATTR(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
[   Appellation: Dr,
    email: K.Bontcheva@dcs.shef.ac.uk,
    telephone number: +4401142221930,
    web page: http://www.dcs.shef.ac.uk/~kalina/
])

```

The disadvantage of the second approach is that it requires the user to create manually these mappings for each problematic datatype property. However, the number of such properties is often quite small and, in our experience, it is feasible to do that in cases when the ontology itself cannot be modified.

The third approach is to not define these properties as **attribute** properties, i.e., to disable the **has** heuristic. In that case, ONTOSUM would use instead the lexicalisations of the properties themselves, derived automatically from their definitions (see Section 3.1). The disadvantage of this approach is that the structuring module will not be able to aggregate the four statements, as they will involve four different properties:

```

has-appellation(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                Appellation: Dr)
has-email-address(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                  string: K.Bontcheva@dcs.shef.ac.uk)
has-telephone-number(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                     string: +4401142221930)
has-web-address(Researcher-In-Academia: K.Bontcheva.dcs.shef.ac.uk,
                 string: http://www.dcs.shef.ac.uk/~kalina/)

```

Consequently, they will be verbalised as four separate sentences:

Kalina Bontcheva has a Dr appellation. Kalina Bontcheva has email K.Bontcheva@dcs.shef.ac.uk. Kalina Bontcheva has web page <http://www.dcs.shef.ac.uk/~kalina/>. Kalina Bontcheva has telephone number +4401142221930.

In this case the information that K.Bontcheva@dcs.shef.ac.uk is an email address comes from the lexicalisation of the property **has-email-address** (see Section 3.1). The same is true for the other datatype properties.

However, while the problem with the implicit semantics of the datatype properties has been solved, the resulting summary is no longer so concise. One solution, to be implemented in future work, would be to implement a syntactic aggregation component which merges two sentences when they have the same subject and verb.

4 Summary Tailoring

The types of tailoring/personalisation considered here are based on information from the user's device profile. Most specifically, we looked into generating summaries within a given length restriction (e.g., 160 characters for mobile phones)

and different formats – HTML for browsers and plain texts for emails and mobile phones.

4.1 Choosing Formatting

Hypertext usability studies [6] have shown that formatting is very important since it improves readability. Bullet lists and font size in particular facilitate skimming by making important information more prominent. Therefore our work focused on generating lists, while font size was made customisable by the user by using the browser’s chosen size. The use of HTML lists in our system is determined as a first step of the text generation process, on the basis of the fully fledged text plan.

The semantic aggregation stage joins all propositions which share the same focused entity and relation, so the resulting more complex propositions can have three or more entities that need to be enumerated in the same sentence. For example, the following complex proposition appears when generating summaries of researchers’ contact details and activities (shown below as a conceptual graph):

```
[RESEARCHER :fs(focus, true)] <- (HAS) <- [EMAIL: xxx]
      - (HAS) <- [WEB-PAGE: yyy]
      - (HAS) <- [TELEPHONE: zzz].
```

The formatter module in HYLITE+ examines each proposition in the text plan to determine if the focused entity participates in the same relation with three or more different concepts. If no formatting is required, then a conjunction will be generated (see example in the previous section).

In the case of HTML summary, such propositions are annotated for bullet list formatting if the repeating relations are ISA, PART_OF, or HAS). No HTML markup is generated at this stage. Instead, the formatting choice is stored as metadata on the proposition:

```
[RESEARCHER :fs(focus, true)] <- (HAS) <- [EMAIL: xxx]
      - (HAS) <- [WEB-PAGE: yyy]
      - (HAS) <- [TELEPHONE: zzz]. : fs(format, ul)
```

This information is used later by the grammar to generate the HTML tags at the same time as the text itself (see Figure 3). This separation allows the formatter (or a later module) to change this choice if it is not appropriate, e.g., to avoid overusing lists.

4.2 Controlling Summary Length

In general, there are two ways in which size constraints can be taken into account during summary generation: (i) approximate the length during content planning; or (ii) given a text plan, decide how to modify or verbalise it to the given size limit via revision. As shown by [9], approximating the text length during content planning is possible but suffers from two problems. The first one is that the result is not exact, so when formatting is added later the text might not fit into the

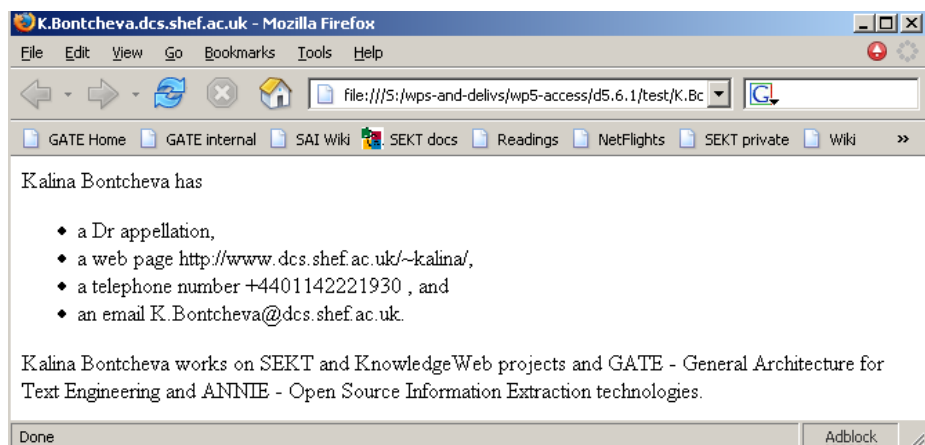


Fig. 3. The HTML summary generated from the RDF input from Section 3.3 with no length restrictions

page any more. The second, more significant problem with this method is that it is hard to maintain and update by non-experts.

Another alternative is to implement a text revision mechanism to analyse the content and structure of the generated summary. However, despite the gains in fluency and coherence, revision-based approaches tend to suffer from computational problems due to the large number of alternatives that need to be explored.

The requirements towards our system are computational efficiency and easy modification by NLG experts. In addition, while the size limit is important, it is not critical if it were exceeded slightly. Therefore, our system always puts in the text structure all statements from the RDF/OWL input. Then given such a text plan, the surface realiser generates the sentences one by one and when a new sentence is added to the summary, summary length is incremented accordingly. As soon as the value of this variable plus the length of the next sentence exceed the limit, the surface realiser is not called further and the last sentence is not included.

The desired summary length is supplied as an input parameter to ONTO-SUM. For instance, when the length restriction is set to 160 characters (and plain text), the generated summary is:

Kalina Bontcheva has a Dr appellation, a web page <http://www.dcs.shef.ac.uk/~kalina/>, a phone number +4401142221930, and an email address K.Bontcheva@dc.shef.ac.uk.

In contrast, when there are no length restrictions and the target formatting is HTML, a longer summary is generated (see Figure 3).

5 Using Ontology Mapping to Run ONTOSUM on Different Ontologies

In the previous sections we discussed how ONTOSUM is adapted to a new ontology. However, frequently there is more than one ontology describing the same or similar domains [4]. For example, both the AKT and SWRC ontologies, discussed above, have concepts describing researchers, their publications, contact details, etc. Therefore, having customised ONTOSUM to the AKT ontology, instead of adapting it to SWRC from scratch, one could use *ontology mapping* rules [4] to “translate” the SWRC instance descriptions into AKT ones and then run ONTOSUM without modifications.

In order to experiment with this approach, we designed manually a set of mapping rules for concepts and properties in the two ontologies. Some concept mappings are: `swrc:AssistantProfessor` is mapped to `akt:Lecturer-In-Academia`, `swrc:AssociateProfessor` – to `akt:Senior-Lecturer-In-Academia`, etc.

Respectively, some property mappings are: `swrc:name` – `akt:full-name`, `swrc:phone` – `akt:has-telephone-number`, `swrc:fax` – `akt:has-fax-number`, `swrc:homepage` – `akt:has-web-address`. Some SWRC properties, e.g., `photo` do not have a corresponding property in the AKT ontology. Therefore, no mapping was provided for them and, consequently, they are not included in the generated summaries.

Once defined, the mapping rules are applied to transform automatically instance descriptions from the SWRC to the AKT ontology, prior to sending them to ONTOSUM for generation. For example, the SWRC instance describing York Sure⁸ looks as follows:

```
<rdf:Description rdf:about="http://www.aifb.uni-
    karlsruhe.de/Personen/viewPersonOWL#instance?id_db=20">
  <rdf:type>
    <owl:Class rdf:about="&swrc;AssistantProfessor"/>
  </rdf:type>
  <swrc:name rdf:datatype="&xsd:string">York Sure</swrc:name>
  <swrc:phone rdf:datatype="&xsd:string"> +49 (0) 721 608 6592
  </swrc:phone>
  <swrc:fax rdf:datatype="&xsd:string"> +49 (0) 721 608 6580
  </swrc:fax>
  <swrc:homepage rdf:datatype="&xsd:string">
    http://www.aifb.uni-karlsruhe.de/WBS/ysu
  </swrc:homepage>
</rdf:Description>
```

After applying the mapping rules and removing properties for which no mappings exist, the system obtains:

```
<rdf:Description rdf:about="http://www.aifb.uni-
```

⁸ The author is grateful to York Sure for supplying the SWRC instance data.

```

        karlsruhe.de/Personen/viewPersonOWL#instance?id_db=20">
<rdf:type>
  <owl:Class rdf:about="http...#Lecturer-In-Academia"/>
</rdf:type>
<akt:full-name rdf:datatype="xsd:string">York Sure</akt:full-name>
<akt:has-telephone-number rdf:datatype="..."> +49 (0) 721 608 6592
</akt:has-telephone-number>
<akt:has-fax-number rdf:datatype="..."> +49 (0) 721 608 6580
</akt:has-fax-number>
<akt:has-web-address rdf:datatype="xsd:string">
  http://www.aifb.uni-karlsruhe.de/WBS/ysu
</akt:has-web-address>
</rdf:Description>

```

When this input is passed to ONTOSUM, it generates the following textual summary, without requiring any customisation:

York Sure has a telephone number +49 (0) 721 608 6592, a fax number +49 (0) 721 608 6580 , and a web page <http://www.aifb.uni-karlsruhe.de/WBS/ysu>.

The advantages of using ontology mapping to enable ONTOSUM to run on different ontologies are: (i) no ONTOSUM customisation is required by the user; (ii) ontology mapping can be performed by ontology engineers and there are even some tools that automate parts of this process [4].

A future extension of this approach would be to allow for more sophisticated mapping or even *ontology merging*, in order to enable ONTOSUM to verbalise also properties and concepts which do not exist in the original ontology. In this case, some limited customisation will be required, mainly concerned with providing new lexical information.

6 Conclusion

This paper presented the ONTOSUM system which uses Natural Language Generation (NLG) techniques to produce textual summaries from Semantic Web ontologies. The main contribution of this work is in showing how existing NLG tools can be adapted to take Semantic Web ontologies as their input, in a way which minimises the customisation effort while being more flexible than template-based ontology verbalisers (e.g., [11]).

A major factor in the quality of the generated summaries is the content of the ontology itself. For instance, the use of string datatype properties with implicit semantics (e.g., **has-web-address**) leads to the generation of summaries with missing semantic information. Three approaches to overcome this problem were presented here and users can choose the one that suits their application best.

We also showed how the generated summaries can be tailored for formatting and length restrictions from a device profile (e.g., mobile phone, Web browser). Another innovative idea is the use of ontology mapping to enable ONTOSUM to generate text from different ontologies, without customisation effort.

Future work will focus on the creation of a user-friendly tool for specifying new summary structuring schemas, because at present this is done directly in the generator's internal structures, which are hard to understand for non-specialists. Another strand of this work will aim at further investigation of the use of ontology mapping and merging in NLG systems.

Another major area for future work is system evaluation. NLG systems are normally evaluated with respect to their usefulness for a particular (set of) task(s), which is established by measuring user performance on these tasks, i.e., *extrinsic* evaluation. This is often also referred to as *black-box* evaluation, because it does not focus on any specific module, but evaluates the system's performance as a whole. Therefore we plan to carry out empirical studies with end-users as part of the SEKT digital library case study. The goal is to carry out a qualitative evaluation of the textual summaries when they appear within a complete semantically-enabled knowledge management system.

References

1. G. Aguado, A. Bañón, John A. Bateman, S. Bernardos, M. Fernández, A. Gómez-Pérez, E. Nieto, A. Olalla, R. Plaza, and A. Sánchez. ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation. In *Workshop on Applications of Ontologies and Problem Solving Methods, ECAI'98*, 1998.
2. K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349–373, 2004.
3. K. Bontcheva and Y. Wilks. Automatic Report Generation from Ontologies: the MIAKT approach. In *Nineth International Conference on Applications of Natural Language to Information Systems (NLDB'2004)*, 2004.
4. J. de Bruijn, F. Martin-Recuerda, D. Manov, and M. Ehrig. State-of-the-art survey on Ontology Merging and Aligning v1. Technical report, SEKT project deliverable D4.2.1, 2004. <http://sw.deri.org/jos/sekt-d4.2.1-mediation-survey-final.pdf>.
5. Kathleen R McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, 1985.
6. Jakob Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, 2000.
7. Cécile L. Paris. Tailoring object descriptions to the user's level of expertise. *Computational Linguistics*, 14 (3):64–78, September 1988. Special Issue on User Modelling.
8. E. Reiter and R. Dale. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, 2000.
9. Ehud Reiter. Pipelines and size constraints. *Computational Linguistics*, 26:251–259, 2000.
10. J.F. Sowa, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, California, San Mateo, CA, 1991.
11. G. Wilcock. Talking OWLs: Towards an Ontology Verbalizer. In *Human Language Technology for the Semantic Web and Web Services, ISWC'03*, pages 109–112, Sanibel Island, Florida, 2003.
12. G. Wilcock and K. Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL. In *Knowledge and Reasoning in Practical Dialogue Systems, IJCAI-2003*, pages 58–63, Acapulco, 2003.

AquaLog: An Ontology-Portable Question Answering System for the Semantic Web

Vanessa Lopez, Michele Pasin, and Enrico Motta

Knowledge Media Institute, The Open University,
Walton Hall, Milton Keynes,
MK7 6AA, United Kingdom
{v.lopez, m.pasin, e.motta}@open.ac.uk

Abstract. As semantic markup becomes ubiquitous, it will become important to be able to ask queries and obtain answers, using natural language (NL) expressions, rather than the keyword-based retrieval mechanisms used by the current search engines. AquaLog is a portable question-answering system which takes queries expressed in natural language and an ontology as input and returns answers drawn from the available semantic markup. We say that AquaLog is portable, because the configuration time required to customize the system for a particular ontology is negligible. AquaLog combines several powerful techniques in a novel way to make sense of NL queries and to map them to semantic markup. Moreover it also includes a learning component, which ensures that the performance of the system improves over time, in response to the particular community jargon used by the end users. In this paper we describe the current version of the system, in particular discussing its portability, its reasoning capabilities, and its learning mechanism.

1 Introduction

The semantic web vision [1] is one in which rich, ontology-based semantic markup is widely available, thus opening the way to novel, sophisticated forms of question answering. However, much work on ontology-driven QA tends to focus on the use of ontologies to support query expansion in information retrieval [2], rather than on exploiting the availability of semantic statements to provide precise answers to complex queries. In particular, a knowledge based QA system can help with answering questions requiring situation-specific knowledge, where multiple pieces of information need to be inferred and combined at run time, rather than simply having a pre-written paragraph of text retrieved [3].

AquaLog is a portable question-answering system which takes queries expressed in natural language and an ontology as input and returns answers drawn from the available ontology-compliant semantic markup. We say that AquaLog is portable, because the configuration time required to customize the system for a particular ontology is negligible. AquaLog combines several powerful techniques in a novel way to make sense of NL queries and to map them to semantic markup. Specifically, it makes use of the GATE NLP platform, string metrics algorithms [4], WordNet

[5, 6], and novel ontology-based *similarity services for relations and classes* to make sense of user queries with respect to the target knowledge base. Also, AquaLog is coupled with a portable and contextualized learning mechanism, which ensures that the performance of the system improves over time, in response to the particular community jargon used by the end users.

AquaLog is implemented in Java as a web application, using a client-server architecture. Moreover, it provides an API, which allows future integration in other platforms and independent use of its components. A key feature of AquaLog is the use of a plug-in mechanism, which allows AquaLog to be configured for different KR languages.

In this paper we describe the current version of the system, in particular discussing its portability, its reasoning capabilities, and its learning mechanism.

The paper is organized as follows: section 2 describes the AquaLog architecture. Section 3 describes the Linguistic Component embedded in AquaLog. Section 4 describes the novel Relation Similarity Service and Learning Mechanism. Section 5 describes a case of integration with Web Services. Section 6 describes the evaluation scenario, followed by discussion and directions for future work. Section 7 describes related work. Finally, section 8 re-iterates the main contributions of this work.

2 The Architecture

At a coarse-grained level of abstraction, the AquaLog architecture can be characterized as a waterfall model, during which a NL query gets translated into a set of intermediate, triple-based representations, query-triples, and then these are translated into ontology-compatible triples, as shown in figure 1. There are two main reasons for adopting a triple-based data model: first of all, it is possible to represent most queries as triples. Secondly, RDF-based knowledge representation (KR) formalisms for the semantic web, such as RDF itself [7] or OWL [8] also subscribe to this binary relational model and express statements as <subject, predicate, object>. Hence, it makes sense for a query system targeted at the semantic web to adopt this data model. However AquaLog triples also have additional features in order to facilitate the reasoning about the answer, such as the voice and tense of the relation and the category. Depending on the category, the triple tells us how to deal with its elements, what inference process is required and what kind of answer can be expected. For instance, different queries may be represented by triples of the same category, since, in natural language, there can be different ways of asking the same question, i.e. “who works in akt¹?” and “Show me all researchers involved in the akt project”. The classification of the triple may be modified during its life cycle in compliance with the target ontology it subscribes to.

In what follows we provide a quick overview of the two main processing modules in AquaLog: the *linguistic component* and the *relation similarity service*. To illustrate

¹ AKT is a EPSRC founded project in which the Open University is one of the partners.
<http://www.aktors.org/akt/>

the system we will consider as test case the semantic web site currently under construction at the knowledge media institute, see <http://plainmoor.open.ac.uk:8080/ksw>, which relies on an ontology which characterizes the key aspects of academic life. Specifically the ontology includes classes and relations to describe projects, technologies, people, news, events, organizations, publications, and research areas. The full specification of the ontology can be found at <http://plainmoor.open.ac.uk:8080/ksw/ontologies.html>. The semantic markup is generated automatically by mining text resources and representing the information held in departmental databases, in terms of the ontology.

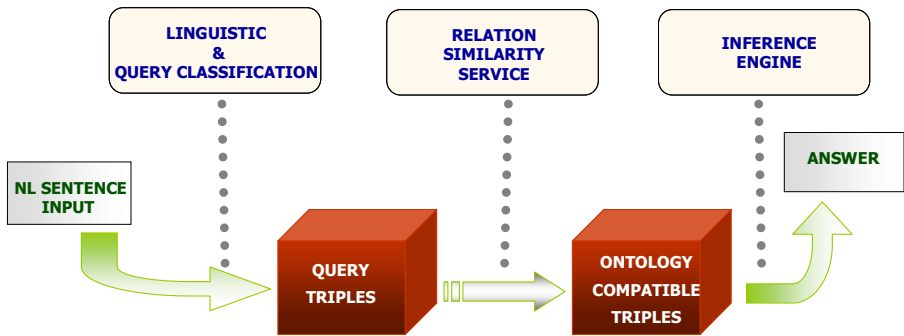


Fig. 1. The AquaLog Data Model

3 Linguistic Component

The Linguistic Component task is to map the NL input query to the Query-Triple. AquaLog uses the GATE [9, 10] infrastructure and resources in order to parse the question as part of the Linguistic Component. Communication between AquaLog and GATE takes place through the standard GATE API.

After the execution of the GATE controller a set of syntactic annotations associated with the input query are returned. These annotations include information about sentences, tokens, nouns and verbs. When developing AquaLog we extended the set of annotations returned by GATE, by identifying terms, relations, question indicators (which/who/when, etc.) and patterns or types of questions. This is achieved through the use of *JAPE grammars*, which allow us to recognize regular expressions using previous annotations in documents. In other words, the *JAPE grammars*' power lie in their ability to regard the data stored in the GATE annotation graphs as simple sequences, which can be matched deterministically by using regular expressions.

Thanks to this architecture it is possible to extend the NL capability of the system in a relatively easy way (NL scalability). Currently, the Linguistic Component, through the *JAPE grammars*, dynamically identifies 23 different linguistic categories or intermediate representations, including: basic queries requiring an affirmation/negation or a description as an answer; or the big set of queries constituted by a wh-question, like “are there any phd students in dotkom?” where the

relation is implicit or unknown or “which is the job title of john?” where not information about the type of the expected answer is provided; etc.

In some cases, e.g. When interpreting the query “list all the projects in KMi about Semantic Web”, the linguistic components cannot resolve the ambiguity associated with the NL query (it cannot identify the constituent to which each modifier has to be attached) and therefore it simply passes the ambiguity on to the Relation Similarity Service (RSS), which can use the ontology or ask the user to solve the ambiguity.

It is important to emphasize that, at this stage the analysis is completely domain independent and is entirely based on the GATE analysis of the English language. The Query-Triple is only a formal, simplified way of representing the NL-query, which we use mainly because at this stage we do not have to worry about getting the representation right in respect to the specific domain knowledge. The role of the intermediate representation is simply to provide an easy way to manipulate input for the RSS. This design choice ensures the easy portability of the system with respect to both ontologies and natural languages.

4 Relation Similarity Service

This is the backbone of the question-answering system. The RSS component is invoked after the NL query has been transformed into a term-relation form and classified into the appropriate category. Essentially the RSS tries to make sense of the input query by looking at the structure of the ontology and the information available on the semantic web, as well as using string similarity matching, generic lexical resources such as WordNet, and a domain-dependent lexicon obtained through the use of a Learning Mechanism, as explained in a later section.

An important aspect of the RSS is that it is interactive. In other words, when the RSS is not sure about how to disambiguate between two or more possible terms or relations in order to interpret a query it will ask the user for disambiguation.

Relations and concepts' names are identified and mapped within the ontology through the RSS and the Class Similarity Service (CSS) respectively. The latter is a sub-module of the RSS, which deals with mapping linguistic terms to classes. Proper names, instead, are mapped into instances by means of the use of string distance metrics algorithms [4]. If this mapping fails a partial solution is implemented for affirmative/negative type of questions, where we make sense of questions in which only one of two instances is recognized. For instance, in the query “is Enrico working in ibm?”, “Enrico” could be mapped into “enrico-motta” in the KB but “ibm” is not found. The answer will output an indirect negative answer, namely the place were Enrico Motta is working.

In any non-trivial natural language system, it is important to deal with the various sources of ambiguity and the possible ways of treating them. Some sentences are syntactically (structurally) ambiguous and although general world knowledge does not resolve this ambiguity, within a specific domain it may happen that only one of the interpretations is possible. The key issue here is to determine some constraints derived from the domain knowledge and to apply them in order to resolve ambiguity

[11]. Whether the ambiguity cannot be resolved by domain knowledge the only reasonable course of action is to get the user to choose between the alternative readings.

Moreover, since every item on the onto-triple is an entry point in the knowledge base or ontology, they are also clickable, giving the user the possibility to get more information about it. The system scans the answers for words denoting instances which are represented in the knowledge base, and then adds hyperlinks to these words/phrases, indicating that the user can click on them. In fact, the RSS is designed to provide justifications for every step of the user interaction. This is crucial to ensure user acceptance of the system.

A typical situation the RSS has to cope with is one in which the structure of the intermediate query does not match the way the information is represented in the ontology.

For instance, the query “who is the secretary in KMi?” is parsed into <person/organization, secretary, kmi>, following purely linguistic criteria. Then, the first step for the RSS is to identify, in the target KB that “kmi” is actually a “research-institute” called “knowledge-media-institute”. Once a successful match is found, the problem becomes to find a relation which links the class research institute (or its superclass organization) to class person (or any of its subclasses, such as academic, student, etc...) or to class organization, by analyzing the taxonomy and relationships in the target KB. However, in this particular case there is a successful matching in the KB for *secretary*, even if secretary is not a relation but a subclass of person. The RSS reasons about the mismatch, re-classifies the intermediate query and generates the correct logical query, in compliance which the ontology, which is organized in terms of <secretary, works-for, kmi>.

Whenever multiple relations are possible candidates for interpreting the query, if the ontology does not provide ways to further discriminate between them, string matching is used to determine the most likely candidate, using the relation name, the learning mechanism, or eventual aliases provided by lexical resources such as WordNet [12]. If no relations are found by using these methods, then the user is asked to choose from the current list of candidates.

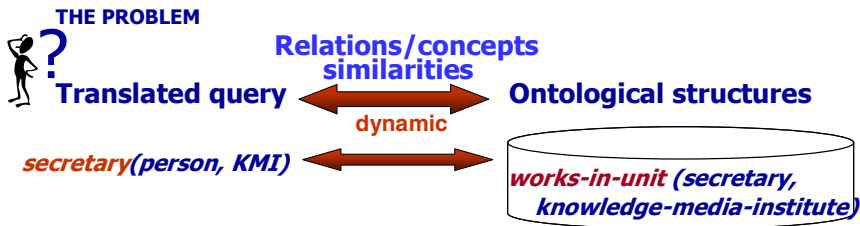


Fig. 2. Scheme for mapping a Query-Triple into an Onto-Triple

Another case is the one in which a query map to a set of triples. In these cases the ambiguity can also be related to the way the triples are linked. The RSS deals with

these cases both by analyzing the structure in the ontology and through the use of heuristics.

For example, let’s consider the query “which news stories have been written by researchers in akt?”. To handle this case the RSS uses a heuristic which suggest the modifier “in akt” to be attached to the closest term that is represented by a class or non-ground term in the ontology, in this case the class “researchers”.

An example of query disambiguation using a combination of linguistic and semantic information from the ontology can be seen in Figure 3. Here a user has asked “Who is the researcher in akt who is interested in the Semantic Web?”. This query is syntactically ambiguous, because the second clause, “who is interested in the Semantic Web”, could syntactically link to either the researcher or “akt”. Because AquaLog knows that “who” can only be a person or an organization, it correctly links it to “researcher”, rather than “akt”. However, there can be other situations where the disambiguation cannot be resolved by using the use of linguistic and/or heuristics and/or the context or semantics in the ontology, as for example in the query “which academic works with peter who has an interest in the semantic web?”. In this case since “academic” and “peter” are respectively a subclass and an instance of “person”, the sentence is truly ambiguous. In fact, it can be understood either as a combination of the resulting lists of the two questions “which academic works with peter” and “which academic has an interest in the semantic web”, or as the relative query “which academic works with peter where the peter we are looking for has an interest in the semantic web”. In such cases, user’s feedback is always required.

Question Answering

Ask a query

who is the researcher in akt who is interested in the Semantic Web

Ask!

Examples

LOGIN

Make Use of Learning Mechanism for relations ☒

Relation Similarity Service

Query Validated ... Category PATTERNS_2

Logical Representation ... Query Term - Relation - Second Term - Third Term

Linguistic Triple:	person organization	- researcher	- akt	-
	person organization	- interested	- semantic web	-
Ontology Triple:	researcher	- has-project-member	- akt	- [WH_GENERICTERM]
	person	- has-research-interest	- semantic-web-area	- [WH_GENERICTERM]

Note: Cannot find a relation to map. The only possible relation is has-research-interest

The answer to the question:

[victoria-uren](#) [farshad-hakimpour](#) [john-domingue](#)

[<< BACK](#)

KNOWLEDGE MEDIA
KMI
INSTITUTE

ADVANCED KNOWLEDGE
TECHNOLOGIES
AKT

DOT.KOM

Powered by
GATE

Fig. 3. Example of context disambiguation by the RSS

4.1 Class Similarity Service

The use of string metrics to map the generic term of the linguistic triple into a term in the ontology may not be enough. Therefore, an additional combination of methods to get synonyms (such as WordNet or our own lexicon) may be used in order to obtain the possible candidates in the ontology. This lexicon can be generated manually or can be built through a learning mechanism (a similar simplified approach to the learning mechanism for relations explained in a later section). The only requirement to execute this learning mechanism for classes is the availability of the ontology mapping for one of the two terms of the triple. In this way, through the ontology relationships that are valid for this term, we can identify a set of possible candidate terms that can complete the triple. User's feedback is required to select whether one of the candidate terms is the one we are looking for, so that the system is able to learn it for future occasions.

4.2 Learning Mechanism

Since the universe of discourse we are working with is determined by and limited to the particular ontology used, there will normally be a number of discrepancies between the natural language questions prompted by the user and the set of terms recognized in the ontology. External resources like WordNet generally help in making sense of unknown terms, giving a set of synonyms and semantically related words which could be detected in the knowledge base. However, in quite a few cases, the RSS fails in the production of a genuine onto-triple because of a user-specific "jargon" found in the linguistic triple. In such a case, it is necessary to learn the new terms employed by the user and disambiguate them in order to produce an adequate mapping of the classes of the ontology. A very common and highly generic example, in our departmental ontology, is the relation *works-for*, to which users normally relate a number of different expressions: *is working*, *works*, *collaborate*, *is involved*. In all these cases the user is asked to disambiguate the relation (choosing from the set of ontology relations consistent with the two question's arguments) and decide if a new mapping should be learned between his/her natural-language-universe and the ontology-language-universe.

4.2.1 Architecture

The learning mechanism in AquaLog consists of two different methods, the *learning* and the *matching* (fig. 4). The latter is called whenever the RSS cannot relate a linguistic triple to the ontology or the knowledge base, while the former is always called after the user manually disambiguates an unrecognized term (and this substitution gives a positive result).

When a new item is learned, it is recorded in a database together with the relation it refers to and a series of constraints *that will determine its reuse within similar contexts*. As it will be explained below, the notion of context is crucial in order to deliver a feasible matching of the recorded words. In the current version the context is defined by the arguments of the question, the name of the ontology and the user information. This set of characteristics constitutes a particular representation of the

context and defines a structured space of hypothesis analogue to that one of a version space² [13].

In future work, this context will be further extended to provide more granularity and semantic expressiveness.

When a question with a similar context is prompted, if the RSS cannot disambiguate the relation-name, the database is scanned for some matching results. Subsequently, these results will be *context-proved* in order to check their consistency with the stored version spaces. By tightening and loosening the constraints of the version space, the learning mechanism is thus able to determine when to propose a substitution and when not to. For example, the user-constraint is a feature that is often bypassed, because we are inside a generic-user session, or because we might want to have all the results of all the users from a single database query.

Before the matching method, we are always in a situation where the onto-triple is incomplete, the relation is unknown or it is a concept. If the new word is found in the database, the context is checked to see if it is consistent with what has been recorded previously. If this gives a positive result we can have a valid onto-triple substitution that triggers the inference engine (this latter basically just scans the knowledge base for results); instead, if the matching fails, a user disambiguation is needed in order to complete the onto-triple. In this case, before letting the inference engine work out the results, the context is drawn from the particular question entered and it is learned together with the relation and the other information in the version space.

Of course, the matching method's movement in the ontology is opposite to the learning method's one. The latter, starting from the arguments, tries to go up until it reaches the highest valid classes possible (GetContext method), while the former takes the two arguments and checks if they are subclasses of what has been stored in the database (CheckContext method). It is also important to notice that the Learning Mechanism does not have a question classification on its own, but it relies on the RSS classification.

4.2.2 Context Definition

As said above, the notion of context is fundamental in order to deliver a feasible substitution service. In fact, two people could use the same jargon but meaning different things.

For example, let's consider the question "Who collaborates with the knowledge media institute?" and assume that the system is not able to solve the linguistic ambiguity of the word "collaborate". The first time, some help from the user is needed, who selects "has-affiliation-to-unit" from a list of possible relations in the ontology. A mapping is therefore created between "collaborate" and "has-affiliation-to-unit", so that the next time the learning mechanism is called it will be able to recognize this specific user jargon.

Let's imagine now a professor, who asks the system the same question "Who collaborates with the knowledge media institute?", but is referring to other research

² A version space is an inductive learning technique proposed by Mitchell in order to represent the consistency of a set of hypothesis with a target concept.

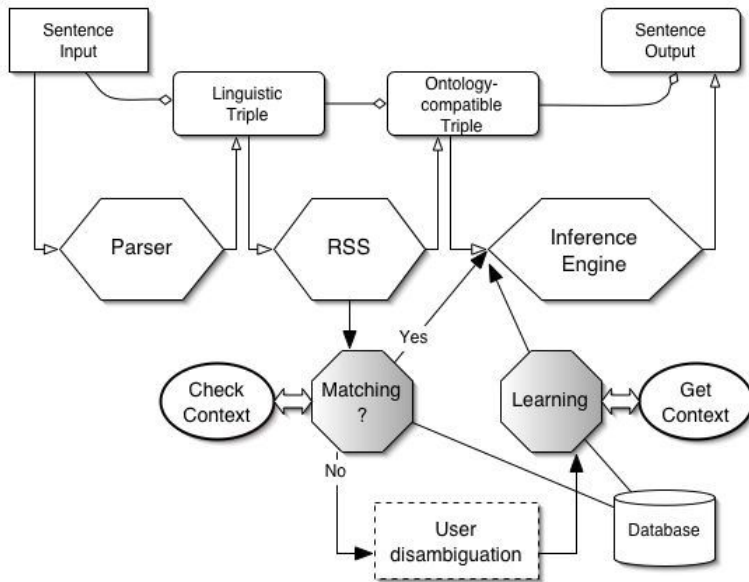


Fig. 4. The learning mechanism architecture

labs or academic units involved with the knowledge media institute. In fact, when asked to choose from the list of possible ontology relations, he/she will possibly enter “works-in-the-same-project”.

The problem, so, is to maintain the two mappings separated while still providing some kind of generalization. This is achieved through the definition of the question's context as determined by its coordinates in the ontology. In fact, since the referring (and *pluggable*) ontology is our universe of discourse, the context must be found within this universe. In particular, since we are dealing with triples, and in the triple what we learn is usually the relation (that is, the middle item), the context is delimited by the two arguments of the triple. In the ontology, these are classes or instances, connected by the relation.

Therefore, in the question “Who collaborates with the knowledge media institute?” the context of the mapping from “ collaborates ” to “ has-affiliation-to-unit ” is given by the two arguments “person” (in the ontology “who” is always translated into “person” or “organization”) and “ knowledge media institute ”. What is stored in the database, for future reuse, is the new word (which is also the key field in order to access the lexicon during the matching method), its mapping in the ontology, the two context-arguments, the name of the ontology and the user details.

4.2.3 Context Generalization

Of course, this kind of recorded context is quite specific and does not let other questions benefit from the same learned mapping. For example, if afterwards we

asked "Who collaborates with the Edinburgh department of informatics?" we would not get an appropriate matching, even if the mapping made sense also in this case.

In order to generalize these results the strategy adopted is to record the most generic classes in the ontology which corresponds to the two triple's arguments, and, at the same time, can handle the same relation. Namely, in our case, we would store the concepts "people" and "organization-unit". This is achieved through a backtracking algorithm in the Learning Mechanism, that takes the relation, identifies its type (the type already corresponds to the highest possible class of one argument, by definition) and goes through all the connected superclasses of the other argument while checking if they can handle that same relation, with the given type. Thus, since only the highest classes of an ontology's branch are kept, all the questions similar to the ones we have seen will fall within the same set, because their arguments are subclasses or instances of the same concepts.

If we go back to the first example presented ("Who collaborates with the knowledge media institute?"), we can see that the difference in meaning between the two interpretations $\langle \text{collaborate} \rangle \rightarrow \langle \text{has-affiliation-to-unit} \rangle$ and $\langle \text{collaborate} \rangle \rightarrow \langle \text{works-in-the-same-project} \rangle$ is preserved, because the two mappings entail two different contexts. Namely, in the first case, the context is given by $\langle \text{people} \rangle$ and $\langle \text{organization-unit} \rangle$, while in the second case the context will be $\langle \text{organization} \rangle$ and $\langle \text{organization-unit} \rangle$. Any other matching could not mistake the two, since what is learned is abstract but still specific enough to rule out the different cases.

4.2.4 User Communities

Another important feature of the learning mechanism is its support for a community of users. As said above, the user details are maintained within the version space and can be considered when interpreting a query. AquaLog allows the user to enter his/her personal information and thus to log in and start a session where all the actions performed on the learned lexicon table are also strictly connected to his/her profile. For example, during a specific user-session it is possible to delete some previous recorded mappings, action that is normally not permitted to the generic user. This latter has in fact the roughest access to the learned material: having no constraints on the user field, the database query will return many more mappings and, quite likely, also meanings that are not desired.

Current work on the learning mechanism is pretty much concentrated on the augmentation of the user-profile's details. In fact, through a specific auxiliary ontology that describes a series of user's profiles, it is possible to infer connections between the type of mapping and the type of user. Namely, it will be possible to correlate a particular jargon to a set of users. Moreover, through an intelligent reasoning service, this correlation will become dynamic, being continually extended or diminished consistently with the relations between user's choices and user's information. For example, if the system detects that a large number of registered users, all characterized by the fact of being PhD students, keep employing the same jargon, it could extend the same mappings to all the other registered PhD students.

5 Integration with Web Services

As we said before, every item in the onto-triple is an entry point to the knowledge base or to the ontology. Therefore, items are clickable and the user can get more information about them. Optionally, AquaLog can be configured to use Semantic Web Services in order to get more information about a particular item (i.e. instance or concept), when required. Here AquaLog uses the same mechanism used by Magpie [14], accessing services published against the same ontology and KB.

6 Evaluation Scenario

AquaLog allows a user who has a question in mind and knows something about the domain to query the semantic markup viewed as a knowledge base. The aim is to provide a system which does not require users to learn specialized vocabularies, or to know the structure of the knowledge base. However, as pointed in [11], although they have to have some idea of the contents of the domain they may have some misconceptions. Therefore some process of familiarization is normally required.

A full evaluation of AquaLog requires both an evaluation of its query answering ability as well an evaluation of the overall user experience. Moreover, because one of our key aims is to make AquaLog an interface for the semantic web, the portability across ontologies will also have to be evaluated formally.

For the first version of AquaLog [15] we performed an initial study, whose aim was to assess to what extent the AquaLog application built using AquaLog with the AKT ontology and the KMi knowledge base satisfied user expectations about the range of questions the system should be able to answer. A second aim of the experiment was also to provide information about the nature of the possible extensions needed to the ontology and the linguistic components – i.e., we not only wanted to assess the current coverage of the system but also get some data about the complexity of the possible changes required to generate the next version of the system.

Thus, we asked 10 members of KMi, none of whom had been involved in the AquaLog project, to generate questions for the system. Because one of the aims of the experiment was to measure the linguistic coverage of the system with respect to user needs, we did not give them much information about the linguistic ability of the system.

We collected in total 76 different questions, 37 of which were handled correctly by AquaLog, i.e., 48.68% of the total. This was a pretty good result, considering that no linguistic restrictions were imposed on the questions.

As pointed in [27] it is very difficult to devise a sublanguage which is sufficiently expressive, yet avoids ambiguity and seems reasonable natural. Furthermore the limitations on linguistic coverage will not be obvious for the user and as a result, independently of whether a particular set of queries is answered or not, the system becomes unusable. Therefore, the conclusion of this previous study was that it was

absolutely crucial to improve the linguistic coverage of the system, which accounted for 69% of the failures.

For the current version of AquaLog, the linguistic coverage (and therefore data model and similarity services) has been extended considerably. At the same time AquaLog can now also deal with the ambiguity problems, derived from the use of more extensive grammars.

However, in this previous study we also identified failures due to a lack of services defined over ontologies (accounted for 20.5% of the errors). For instance, one query asked about “the top researchers”, which requires a mechanism for ranking researchers in the lab - people could be ranked according to citation impact, formal status in the department, etc. In the context of the semantic web, we believe that these failures are less to do with shortcomings of the ontology than with the lack of appropriate services, defined over the ontology.

No work has been done yet in relation to the service failures, which remains a future line of work for future versions of the system.

In order to evaluate the portability of the system we interfaced AquaLog to the Wine Ontology [16], an ontology used to illustrate the specification of the OWL W3C recommendation. The experiment confirmed the thesis that AquaLog is ontology independent, as we did not notice any hitch in the behaviour of this configuration compared to the others built previously. However, this ontology highlighted some AquaLog limitations, which must be addressed in the near future. For instance, a direct question like “which wines are recommended with cakes” will fail because there is not a direct relation between wines and desserts, as there is a mediating concept called “mealcourse”. However, the knowledge is in the ontology, and the question can be addressed if reformulated as “what wines are recommended for dessert courses based on cakes?”.

The wine ontology does not have much information instantiated, and as a result no answer can be found for most of the questions. However, it is a good test case for the Linguistic and Similarity Components responsible for creating the ontology compliance triple (from which an answer can be inferred in a relatively easy way).

7 Related Work

7.1 Close-Domain Natural Language Interfaces

This scenario is of course very similar to asking natural language queries to databases (NLDB), which has long been an area of research in the artificial intelligence and database communities [17, 18, 19, 20, 21], even if as [22, 23] say “in the past decade has somewhat gone out of fashion”. The use of natural language to access relational databases can be traced back from the late sixties and early seventies. In [22] a detailed overview of the state of the art for these systems can be found. The main difference between AquaLog and the latest generation of NLDB systems [24] is that AquaLog uses an intermediate representation throughout the entire process, from the representation of the user’s query (NL front end) to the representation of an ontology compliant triple (through the use of similarity services), from which an answer can be

directly inferred. It takes advantage of the structure of ontologies in a way that makes the entire process highly portable.

PRECISE [25] maps questions to the corresponding SQL query, by identifying classes of questions that are easy to understand in a well defined sense: the paper defines a formal notion of semantically tractable questions. Questions are sets of attribute/value pairs and a relation token corresponds to either an attribute token or a value token. In PRECISE the problem of finding a mapping from the tokenization to the database requires that all tokens must be distinct; questions with unknown words are not semantically tractable and cannot be handled. In contrast with PRECISE, AquaLog employs similarity services to interpret the user query by means of the vocabulary in the ontology. As a consequence, AquaLog is able to reason about the ontology structure in order to make sense of unknown relations or classes which appear not to have any match in the KB or ontology.

7.2 Open-Domain QA Systems

Most current work on question answering is somewhat different in nature from AquaLog as it concerns open-domain systems. However, there are linguistic problems common in most kinds of natural language understanding systems.

Most text based QA applications typically involve two steps [26]: 1. Identifying the semantic type of the entity sought by the question (a date, a person and so on); 2. Determining additional constraints on the answer entity, i.e. identifying key words or syntactic or semantic relations to be used in matching candidate answers. Various systems have, therefore built hierarchies of question types based on the types of answers sought [27, 28, 29, 30].

As pointed by R. Srihari et al. in [28]: (i) IE can provide solid support for QA; (ii) low-level IE like Named Entity (NE) tagging is often a necessary component (an analysis showed that over 80% out of 200 questions asked for an NE as a response); (iii) a robust natural language shallow parser provides a structural basis for handling questions; (iv) high-level domain independent IE, i.e., extraction of multiple relationships between entities, is expected to bring about a breakthrough in QA.

AquaLog also subscribes to point (iii), however the main two differences with open-domain systems are: (1) it is not necessary to build hierarchies or heuristics to recognize name entities, as all the semantic information needed is in the ontology; (2) AquaLog has already implemented mechanisms to extract and exploit the relationships to understand a query. Nevertheless, the goal of the main similarity service in AquaLog, the RSS, is to map the relationships in the linguistic triple into an ontology-compliant-triple. As described in [28] NE is necessary but not complete in answering questions because NE by nature only extracts isolated individual entities from text, therefore methods like “the nearest NE to the queries key words” are used.

Both AquaLog and open-domain systems attempt to find synonyms plus their morphological variants to the terms or key words. Also in both cases, at times, the rules leave ambiguity unresolved and produce non-deterministic output for the focus of the question or asking point (for instance, who can be related to a person or to an organization).

As in open-domain systems, AquaLog also automatically classifies the question beforehand. The main difference is that AquaLog classifies the question based on the kind of triple needed, while most of the open-domain QA systems classify questions according to their answer target [30] (person, location, date, ..). The triple contains information not only about the answer expected or focus, which is what we call the generic term of the triple, but also about the relationships between the generic term and the other terms participating in the question (each relationship is represented in a different triple). Different queries may belong to the same triple category. An efficient system should therefore group together equivalent questions types.

The best result of the TREC9 [31] were obtained by the system FALCON described in Harabaigiu et al. [32]. When the question concept indicating the answer type is identified, it is mapped into an answer taxonomy. The top categories are connected to several word classes from WordNet. The example shown in [32] identifies the expected answer type of the question “what do penguins eat?” to be food since it is the most widely used concept in the glosses of the subhierarchy of the noun *synset* {eating, feeding}. Also, FALCON gives a cached answer if the similar question has already been asked before; a similarity measure is calculated to see if the given question is a reformulation of a previous one. A similar approach is adopted by the learning mechanism in AquaLog, where the similarity is given by the context stored in the triple.

7.3 Open-Domain QA Systems Using Triple Representation

The START [33] system goal is also to extract answers from text. AquaLog relational data model (triple-based) is somehow similar to the approach adopted by START, called “object-property-value”. The difference is that instead of properties we are looking for relations between terms, or between a term and its value. Using an example presented in [33]: “What languages are spoken in Guernsey?”, for START the property is “languages” between the Object “Guernsey” and the value “French”; for AquaLog it will be translated into a relation “are spoken” between a term “language” and a location “Guernsey”.

The system described in Litkowski et al. [34], called DIMAP, extracts “semantic relation triples” from a document. The semantic relation triple described consists of a discourse entity, a semantic relation that characterizes the entity’s role in the sentence and a governing word (generally the word in the sentence that the discourse entity stood in relation to). The semantic relation and the governing words were not identified for all discourse entities, but a record for each entity was still added to the database sentence (on average 9.8 triples per sentence). The same analysis is performed to create a set of records for each question (in average 3.3 triples per sentence), in which one of the semantic relation triples contained an unbound variable as a discourse entity, corresponding to the type of question. DIMAP-QA converts the document into triples and AquaLog uses the ontology, which it may be seen as a collection of triples. One of the current AquaLog limitations is that the number of

triples is fixed for each query category, although, the AquaLog triples change during its life cycle. However, the performance is still high as most of questions can be translated into one or two triples.

7.4 Ontologies in Question Answering

We have already mentioned that many systems simply use an ontology as a mechanism to support query expansion in information retrieval. In contrast with these systems AquaLog is interested in providing answers derived from semantic annotations to queries expressed in NL. In the paper by R. Basili [35] the possibility of building an ontology-based question answering system in the context of the semantic web is discussed. Open domain QA systems do not rely on specialized conceptual knowledge as they use a mixture of statistical techniques and shallow linguistic analysis. Ontological QA systems propose to attack the problem by means of an internal unambiguous knowledge representation. Their approach is being investigated in the context of EU project MOSES, with the explicit objective of developing an ontology-based methodology to search, create, maintain and adapt semantically structured Web contents according to the vision of semantic web. The approach and scenario has many similarities with AquaLog. However, AquaLog is implemented on-line and has a wider linguistic coverage. The query classification is guided by the equivalent semantic representations or triples. The mapping process is converting the elements of the triple into entry-points to the ontology and KB.

8 Conclusion

In this paper we have described the AquaLog ontology-driven query answering system in the context of the Semantic Web scenario. AquaLog presents an elegant solution in which different strategies are combined together to make sense of an NL query with respect to the universe of discourse covered by the ontology. Its ontology portability capabilities make AquaLog a suitable NL front-end for the Semantic Web.

Acknowledgements

This work was partially supported by the Advanced Knowledge Technologies (AKT), which is sponsored by the UK Engineering and Physical Sciences Research Council and by the Dot.Kom project under grant IST-2001-34038. The authors would like to thank Yuanguai Lei, Anne de Roeck, Davide Guidi, Dnyanesh Rajpathak, Martin Dzbor, John Domingue, Victoria Uren and Kalina Bontcheva for useful AquaLog related input and those members of the lab who took part in the evaluation.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American*, 284 (5) (2001) 33-43
2. Mc Guinness, D.: Question Answering on the Semantic Web. *IEEE Intelligent Systems*, 19 (1) (2004) 82-85
3. Clark, P., Thompson, J., Porter., B.: A Knowledge-Based Approach to Question-Answering. In the *AAAI Fall Symposium on Question-Answering Systems*, CA: AAAI. (1999) 43-51
4. Cohen, W., W., Ravikumar, P., Fienberg, S., E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In *IIWeb Workshop*, (2003), <http://www-2.cs.cmu.edu/~wcohen/postscript/ijcai-ws-2003.pdf>
5. Pasca, M., Harabagiu, S.: The Informative Role of WordNet in Open-Domain Question Answering. In *2nd Meeting of the North American Chapter of the Association for Computational Linguistics (Naacl)* (2001)
6. JWNL (Java WordNet library) <http://sourceforge.net/projects/jwordnet>
7. RDF: <http://www.w3.org/RDF/>
8. Mc Guinness, D., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation 10 (2004) <http://www.w3.org/TR/owl-features/>
9. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia (2002)
10. Tablan, V., Maynard, D., Bontcheva, K.: *GATE - A Concise User Guide*. University of Sheffield, UK. <http://gate.ac.uk/>
11. Copestake, A., Jones, K., S.: Natural language interfaces to databases. *Knowledge Engineering Review*, 5 (4) (1990) 225-249
12. Fellbaum, C. (Ed.), *WordNet, An Electronic Lexical Database*. Bradford Books, May, (1998)
13. Mitchell, T. M.: *Machine learning*. McGraw-Hill, New York (1997)
14. Dzbor, M., Domingue, J., Motta, E.: Magpie – Towards a Semantic Web Browser. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, *Lecture Notes in Computer Science*, 2870/2003, Springer-Verlag (2003)
15. Lopez, V., Motta, E.: Ontology Driven Question Answering in AquaLog. In *Proceedings of the 9th International Conference on Applications of Natural Language to Information Systems*, Manchester, England (2004)
16. W3C, OWL Web Ontology Language Guide: <http://www.w3.org/TR/2003/CR-owl-guide-0030818/>
17. Burger, J., Cardie, C., Chaudhri, V., et al.: Tasks and Program Structures to Roadmap Research in Question & Answering (Q&A). *NIST Technical Report*, 2001 <http://www.ai.mit.edu/people/jimmylin/%0Apapers/Burger00-Roadmap.pdf>
18. Kaplan, J.: Designing a portable natural language database query system. *ACM Transactions on Database Systems*, 9 (1) (1984) 1-19
19. Androutsopoulos, I., Ritchie, G.D., and Thanisch, P.: MASQUE/SQL - An Efficient and Portable Natural Language Query Interface for Relational Databases. In Chung, P.W. Lovegrove, G. and Ali, M. (Eds.), *Proceedings of the 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Edinburgh, U.K., Gordon and Breach Publishers (1993) 327-330

20. Chu-Carroll, J., Ferrucci, D., Prager, J., Welty, C.: Hybridization in Question Answering Systems. In Maybury, M. (Ed.), *New Directions in Question Answering*, AAAI Press, (2003)
21. Jung, H., Geunbae Lee, G.: Multilingual Question Answering with High Portability on Relational Databases. *IEICE transactions on information and systems*, E86-D (2) (2003) 306-315
22. Androutsopoulos, I., Ritchie, G.D., Thanisch P.: Natural Language Interfaces to Databases - An Introduction. *Natural Language Engineering*, 1 (1) (1995) 29-81
23. Hunter, A.: Natural language database interfaces. *Knowledge Management*, (2000)
24. De Roeck, A., N., Fox, C., J., Lowden, B., G., T., Turner, R., Walls, B.: A Natural Language System Based on Formal Semantics. In *Proceedings of the International Conference on Current Issues in Computational Linguistics*, Pengang, Malaysia, (1991)
25. Popescu, A., M., Etzioni, O., Kautz, H., A.: Towards a theory of natural language interfaces to databases. In *Proceedings of the International Conference on Intelligent User Interfaces*, Miami, FL, USA, Jan. 12-15 (2003) 149-157
26. Hirschman, L., Gaizauskas, R.: Natural Language question answering: the view from here. *Natural Language Engineering, Special Issue on Question Answering*, 7 (4) (2001) 275-300
27. Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Goodrum, R., Girju, R., Rus, V.: LASSO: A Tool for Surfing the Answer Net, in *Proceedings of the Text Retrieval Conference (TREC-8)*, Nov. (1999)
28. Srihari, K., Li, W., Li, X.: Information Extraction Supported Question- Answering, In T. Strzalkowski & S. Harabagiu (Eds.), in *Advances in Open- Domain Question Answering*. Kluwer Academic Publishers (2004)
29. Hovy, E.H., Gerber, L., Hermjakob, U., Junk, M., Lin, C.-Y.: Question Answering in Webclopedia. In *Proceedings of the TREC-9 Conference*. NIST, Gaithersburg, MD (2000)
30. Wu, M., Zheng, X., Duan, M., Liu, T., Strzalkowski, T.: Question Answering by Pattern Matching, Web-Proofing, Semantic Form Proofing. NIST Special Publication: *The Twelfth Text REtrieval Conference (TREC)* (2003) 500-255
31. De Boni, M.: TREC 9 QA track overview.
32. Harabagiu, S., Moldovan, D., Pasca, M., Mihalcea, R., Surdeanu, M., Bunesco, R., Girju, R., Rus, V., Morarescu, P.: Falcon - Boosting Knowledge for Answer Engines. In *Proceedings of the 9th Text Retrieval Conference (Trec-9)*, Gaithersburg, Maryland, Nov. (2000)
33. Katz, B., Felshin, S., Yuret, D., Ibrahim, A., Lin, J., Marton, G., McFarland A. J., Temelkuran, B.: Omnibase: Uniform Access to Heterogeneous Data for Question Answering. In *Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB)* (2002)
34. Litkowski, K. C. Syntactic Clues and Lexical Resources in Question-Answering. In Voorhees, E. M. and Harman, D. K. (Eds) *Information Technology: The Ninth Text REtrieval Conference (TREC-9)*, NIST Special Publication 500-249. Gaithersburg, MD: National Institute of Standards and Technology (2001) 157-66
35. Basili, R., Hansen, D., H., Paggio, P., Pazienza M., T., Zanzotto F., M. Ontological resources and question answering *Workshop on Pragmatics of Question Answering, held jointly with NAACL 2004* Boston, Massachusetts, May (2004)

Lexically Evaluating Ontology Triples Generated Automatically from Texts

Peter Spyns¹ and Marie-Laure Reinberger²

¹ Vrije Universiteit Brussel - STAR Lab,
Pleinlaan 2 Gebouw G-10, B-1050 Brussel - Belgium
tel.: +32-2-629.1237; fax: +32-2-629.3819

`Peter.Spyns@vub.ac.be`

² University of Antwerp - CNTS,
Universiteitsplein 1, B-2610 Wilrijk - Belgium
tel.: +32-3- 820.2766; fax: +32-3-820.2762
`marielaure.reinberger@ua.ac.be`

Abstract. Our purpose is to present a method to lexically evaluate the results of extracting in an unsupervised way material from text corpora to build ontologies. We have worked on a legal corpus (EU VAT directive) consisting of 43K words. The unsupervised text miner has produced a set of triples. These are to be used as preprocessed material for the construction of ontologies from scratch. A quantitative scoring method (coverage, accuracy, recall and precision metrics resulting in a 38.68%, 52.1%, 9.84% and 75.81% scores respectively) has been defined and applied.

1 Introduction and Background

A recent evolution in the areas of artificial intelligence, database semantics and information systems is the advent of the Semantic Web [1]. It evokes "futuristic" visions of intelligent and autonomous software agents including mobile devices, health-care monitoring, ubiquitous and wearable computing. E.g., a heartbeat monitoring device integrated in a person's shirt could trigger, in case of observed rhythm deviations, a web agent that schedules an appointment with his/her doctor via the mobile network.

An essential condition to the actual realisation and unlimited use of these smart devices and programs is the possibility for interoperability, which is currently still lacking to a large extent. Indeed, intelligent agents have to be able to exchange "meaningful" messages¹ while continuing to function autonomously (interoperability with local autonomy as opposed to integration with central control). Exchange of meaningful messages is only possible when the intelligent devices or agents share a common conceptual system representing their "world"², as is the case for human communication. Meaning

¹ We make abstraction here of the feasibility of physically connecting these devices and services or agents to a (global) network.

² See [28] for more details on the semantics of the Semantic Web.

ambiguity should be, by preference, eliminated. Nowadays, a formal representation of such (partial) intensional definition of a conceptualisation of an application domain is called an ontology [10].

The development of ontology-driven applications is currently slowed down due to the knowledge acquisition bottleneck. Indeed, the process of conceptualising an application domain and its formalisation need substantial human resources and efforts. Therefore, techniques applied in computational linguistics and information extraction (in particular machine learning) are used to create or grow ontologies in a period as limited as possible with a quality as high as possible. Sources can be of different kinds including databases and their schemas, semi-structured data (XML, web pages), ontologies³ and texts. Activities in the latter area are grouped under the label of Knowledge Discovery in Text (KDT), while the term "Text Mining" is reserved for the actual process of extracting the information [14].

In addition, there is hardly any method available to thoroughly evaluate the results of (unsupervised) text mining for ontologies. We have looked to the domain of information science to suggest a quantitative method - see [21,25] - that will be refined in this paper. Previously, criteria for ontology evaluation have been put forward by Gruber [9-p.2] and taken over by Ushold and Grüninger [27]: clarity, coherence, extendibility, minimal encoding bias and minimal ontological commitment. Gómez-Pérez [8-p.179] has proposed consistency, completeness and conciseness. Neither set of criteria are well suited to be applied in our case as the triples produced by the unsupervised miner are merely "terminological combinations" (i.e., no explicit meaning for the terms and roles is provided, not to mention any formal definition of the intended semantics). Recent proposals for evaluation methods have been discussed during the ECAI2004 workshop on ontology learning and population [4]. The majority of them proposes to evaluate an ontology mediating improvement measures of an existing application or by a comparison with another ontology acting as a gold standard. Typical of our approach will be that only the corpus (lemmatised but otherwise unmodified) constitutes the reference point, and not an annotated corpus or some other ontology. We aim at defining an evaluation method that is extremely easily applicable by laymen.

We have been mainly inspired by the criteria proposed by Guarino [11-p.7] and the classical information extraction measures [29]. In the current ontology engineering field, it is problematic to objectively evaluate ontologies in an automated way as in the overwhelming majority of cases (suitable) gold standards are lacking [12]. Below (section 3), we give our definition of these criteria that allow computation, which are closer to the traditional information extraction definitions of recall and precision.

The remainder of this paper is organised as follows. The next two sections present the material (section 2) and methods (section 3). The evaluation results are described in section 4 and discussed subsequently (section 5). Related work (section 6) is presented. Indications for future research are given in section 7, and some final remarks (section 8) conclude this paper.

³ This is called ontology aligning and merging

2 Material

2.1 Unsupervised Text Mining

We have opted for extraction techniques based on unsupervised learning methods since these do not require specific external domain knowledge such as thesauri and/or tagged corpora. As a consequence, these techniques are expected to be more easily portable to new domains. In order to extract this information automatically from our corpus, we used the memory-based shallow parser for English, which is being developed at CNTS Antwerp and ILK Tilburg [3]⁴. This shallow parser takes plain text as input, performs tokenisation, part of speech (POS) tagging, phrase boundary detection, and finally finds grammatical relations such as subject-verb and object-verb relations, which are particularly useful for us. The software was developed to be efficient and robust enough to allow shallow parsing of large amounts of text from various domains. We extract from the shallow parser output semantic relations that match predefined syntactic patterns. Additional statistics using normalised frequencies and probabilities of occurrence are calculated to separate noise (i.e. false combinations generated by chance) from genuine results. More details on the linguistic processing can be found in [20, 21, 22].

2.2 Corpus

The VAT corpus (a single long document) consists of 49,5K words. It constitutes the sixth EU directive on VAT (77/388/EEC of 27 January 2001 - English Version) that has to be adopted and transformed into local legislation by every Member State⁵. We applied the memory based shallow parser to this corpus. After some format transformation, the text miner outputs 315 triples subject-verb-object, such as *<person pay tax>*, and 500 triples noun phrase-preposition-noun phrase such as *<accordance with article>* resulting in a total of 815 triples. In addition, the Wall Street Journal corpus (a collection - 1290K words⁶ - of English newspaper articles) serves a "neutral" corpus representing the general language use - see below.

To compute the necessary frequencies and statistics about the corpora, specific Perl scripts have been used. Further manipulation of the numbers is done by means of other small scripts implemented in Tawk v.5 [32] in combination with some standard DOS or Linux commands (mainly "sort").

2.3 DOGMA Ontology Engineering Framework

Before presenting the actual experiments, we shortly discuss the framework for which the results of the experiments are meant to be used, i.e. the *VUB STAR Lab DOGMA* (Developing Ontology-Guided Mediation for Agents) ontology engineering approach⁷.

⁴ See <http://ilk.kub.nl> for a demo version.

⁵ This directive serves as input for the ontology modelling and terminology construction activities in the EU FP5 IST FF Poirot project (IST-2001-38248).

⁶ The Linux `wc -c` command has been used to count the words of the VAT and WSJ corpora.

⁷ see <http://www.starlab.vub.ac.be/research/dogma>

The results of the unsupervised mining phase are represented as *lexons*. These are binary fact types indicating the entities as well as the roles assumed in a semantic relationship [24]. Formally, a lexon is described as $\langle(\gamma, \lambda): term_1 \text{ role } co\text{-role } term_2\rangle$. For the sake of brevity, the context (γ) and language (λ) identifiers will be omitted. Informally we say that a lexon expresses that the $term_1$ (or head term) may plausibly have $term_2$ (or tail term) occur in an associating *role* (with *co - role* as its inverse) with it. The basic insights of DOGMA originate from database theory and model semantics [17]. With some simplifications, one can state that a lexon can be considered as a combination of two RDF-triples.

2.4 Combining all the Above

As the triples resulting from the unsupervised mining consist of three elements⁸ (two terms consisting of one or several words and one role represented by the verb or the preposition⁹) extracted from the VAT corpus, it is possible to investigate to what extent the vocabulary of triples (to be converted afterwards to DOGMA lexons) adequately represents the notions of a particular application domain. Note that this technique in principle could be applied not only to DOGMA lexons but also to RDFS and OWL Lite ontologies.

3 Methods

3.1 Introduction

The starting point in this paper is that triples, representing the basic binary facts expressed in natural language about a domain, can be extracted from the available textual sources using the unsupervised text miner described above. The basic research question is whether or not suitable metrics can be defined to quantitatively evaluate the goodness of fit between the vocabulary of the triples extracted and the intended domain model "embodied" in the textual sources.

We have combined the criteria of Guarino [11] with the more classical information extraction measures [29]. We stress that text mining does not deal with an actual conceptualisation, but rather with its representation or lexicalisation in a text, meaning that we cannot access directly the conceptualisation (meaning level) but have to stay on the linguistic level [26]. However, as many ontology engineers seem to overlook this distinction, the evaluation method proposed here can be applied to existing ontologies as well.

The four measures are:

- *coverage*: are the triples retrieved representing the domain ?
- *accuracy*: are the triples retrieved not too general but reflecting the specialised terms of the domain ?

⁸ In fact, the words composing an element have been lemmatised, i.e. reduced to their base form. E.g., working, works, worked $\rightarrow$ work. In this paper, the terms 'word', 'term', and 'lemma' are used interchangeably.

⁹ Co-roles and context are not provided by the CNTS unsupervised miner.

- *recall*: have all the relevant triples been retrieved
- *precision*: are the triples retrieved relevant for the domain ?

In the following sections, we shall elaborate on a computable definition of these criteria and on the ideas that form the basis of the metrics. The exact formulas will be explained as well. The core of the method relies on decomposing the triples into their constituting words and performing calculations on the individual words.

3.2 Coverage

A simplistic metric to determine the coverage would be to calculate the intersection between the vocabulary of the triples and the entire corpus. As many words do not represent domain concepts (e.g. adverbs, determiners, particles, ..., which are by definition not retained by the unsupervised text miner) the triples generated automatically most probably will not attain a high domain coverage rate. In order to differentiate more important words from less important ones, the frequency of a word can be taken into account. Naively, one would expect that important domain words are mentioned more often than others. Therefore, the words are grouped into frequency classes, i.e. the absolute number of times a word appears in a corpus. E.g., in the VAT corpus, the word 'the' appears 3573 times while it is the only element in the frequency class 3573. Conversely, 'by-product' and 'chargeability' each occur only once, but there are 1521 different words in the frequency class 1. For each frequency class the ratio of the vocabulary intersection and the frequency class is calculated, and subsequently averaged over the number of classes.

$$coverage(triples, text) =$$

$$\frac{\sum_{i=1}^n \frac{\#(words\_triples\_freq\_class_i \cap words\_text\_freq\_class_i)}{\#words\_text\_freq\_class_i}}{n} * 100$$

The coverage of a text by the vocabulary of triples automatically mined will be measured by counting for each frequency class the number of words, constituting the triples, that are identical with words from that frequency class and comparing this number to the overall word count for the same class. The mean value of these proportions constitutes the overall coverage percentage.

3.3 Precision and Recall

It is difficult to compute the precision, i.e. determining if the triples retrieved are correct, whereby correct is to be interpreted as making sense for the application domain. These decisions require the involvement of human evaluators, and/or an established gold standard. An earlier experiment on evaluating the precision of unsupervised text mining for ontologies is reported in [20] using UMLS [13] as gold standard.

In the approach proposed here, we use a metric from quantitative linguistics [6] to automatically build a gold standard. The standard consists of a set of words that characterise an application domain text resulting from a quantitative comparison with another text. Regarding technical texts, one can easily assume that the specialised vocabulary

constitutes the bulk of the characteristic vocabulary, especially if the other corpus with which to compare is the Wall Street Journal (= collection of general newspaper articles), as is the case here.

The following statistical formulas (used to calculate the difference between two proportions) determine which words are typical of one text compared to another:

$$\tilde{f} = \left(\frac{f_{word\_text}}{N} \right) * 100$$

with f being the absolute frequency of a word in a text and N being the total number of words of that text.

$$z = \frac{\tilde{f}_1 - \tilde{f}_2}{\sqrt{\left(\frac{\tilde{f}_1 * (100 - \tilde{f}_1)}{N_1} \right) + \left(\frac{\tilde{f}_2 * (100 - \tilde{f}_2)}{N_2} \right)}}$$

with z expressing a significance value for the deviation between the relative frequencies $\tilde{f}_1$ and $\tilde{f}_2$. Depending on one's preference for the threshold, values of z (expressed in units of σ) below 1,96 ($p < 5\%$) or 2,57 ($p < 1\%$) are statistically not significant.

$$recall(triples, text) =$$

$$\left(\frac{\#(words\_of\_triples\_mined \cap statistically\_relevant\_words)}{\#statistically\_relevant\_words} \right) * 100$$

The ratio of the vocabulary common to the retrieved triples and statistically significant (threshold = 1,96) characteristic words and these characteristic words determines the recall value.

$$precision(triples, text) =$$

$$\left(\frac{\#(words\_of\_triples\_mined \cap statistically\_relevant\_words)}{\#words\_of\_triples\_mined} \right) * 100$$

The ratio of the vocabulary common to the triples mined and statistically significant (threshold = 1,96) characteristic words and the vocabulary of the triples mined determines the precision value.

As is done for the coverage, one could also compute the average over the frequency classes of their recall and precision values.

3.4 Accuracy

The purpose of calculating the accuracy is to refine the coverage measure that is based only on word frequency, by combining it with the precision measure. The source of inspiration is Zipf's law [31]. It states that the product of the frequency and the rank order is approximately constant [29–p.2]. Or said in a simpler way, in each text there is a small set of words that occur very often and a large set of words that rarely occur. Zipf has discovered experimentally that the more frequently a word is used, the less meaning it carries. Hence his observation that the higher frequency classes (i.e. containing the

few words that appear very often) contain mostly "empty" words (also called function or stop words).

A corollary from Zipf's law is that domain or topic specific vocabulary is to be looked for in the middle to lower frequency classes. Consequently, triples mined from a corpus should preferably contain terms from these "relevant" frequency classes. Luhn [15] has defined intuitively a frequency class upper and lower bound between which the most significant words are found in the middle of the area of the frequency classes between these boundaries. He called this the "resolving power of significant words".

The metrics from quantitative corpus linguistics mentioned above are re-used to objectively determine the range of relevant frequency classes. The frequency classes that contain a high number of typical words will be considered as "relevant" frequency classes. Currently, we assume that a frequency class should contain at least 60% of characteristic words in order to be a relevant class. Additionally, one could apply the statistical significance threshold (not done for these experiments). Notions represented by words of the relevant frequency classes should be maximally included in an ontology for that particular application domain.

$$accuracy(triples, text) =$$

$$\frac{\sum_{i=1}^n \frac{\#(words\_triples\_rel\_freq\_class_i \cap words\_text\_rel\_freq\_class_i)}{\#words\_text\_rel\_freq\_class_i}}{n} * 100$$

The accuracy of automatically mined triples to lexically represent the important notions of a text will be measured by averaging the coverage percentage for the relevant frequency classes. A frequency class is considered to be relevant if it contains more than 60% of typical vocabulary.

3.5 Experiments

In the experiment, various scripts have been used to calculate the absolute and relative frequencies as well as the coverage, recall, precision and accuracy measures mentioned above. The input texts have not been filtered or modified except for the lemmatisation.

88,44% of the lemmas (=types) falls into the first 110 FCs, which represents 10,98% of the total word occurrences (=tokens). There are 66 FCs more above 110. The ten highest are 752, 790, 1011, 1110, 1157, 1260, 1378, 2011, 2401 and 3573, all consisting of one word (see 1).

We have also determined a baseline against which the results of our method will be compared. The core of the baseline algorithm is a random number generator (built-in TAWK function [32]) that is used to pick out a word from the lemmatised corpus vocabulary (3210 unique base forms). An equal amount of lemmas is randomly selected as there are lemmas in the triples list.

4 Results

It should be clear from the on-set that high scores will not be attained. Only terms in a verb-object and a subject-object grammatical relation are selected by the shallow

parser, combined by clustering and submitted subsequently to several selection thresholds, which already constitutes a substantial reduction of the number of lemmas that constitute the triples.

4.1 Coverage

A coverage rate of 39.68% is obtained (the naive coverage rate being 8.62%). Figures 1 and 3 show that, especially for the first six frequency classes (FC) (i.e. lemmas appearing once up to six times) the coverage rate is below 10%. In figure 1, for reasons of graphical visibility, a ceiling for the number of lemmas (Y-axis) has been established on 170. FC 1 contains 1521, FC 2 442 and FC 3 223 lemmas respectively. The high dispersion of the coverage for FCs starting from class 40 is to be explained by the low number of lemmas in these classes (rarely higher than 5). From class 82 onwards, a FC consists of a single lemma (FC 93, 108, 120, 128, 131 and 169 being the exceptions containing two and 100 three lemmas).

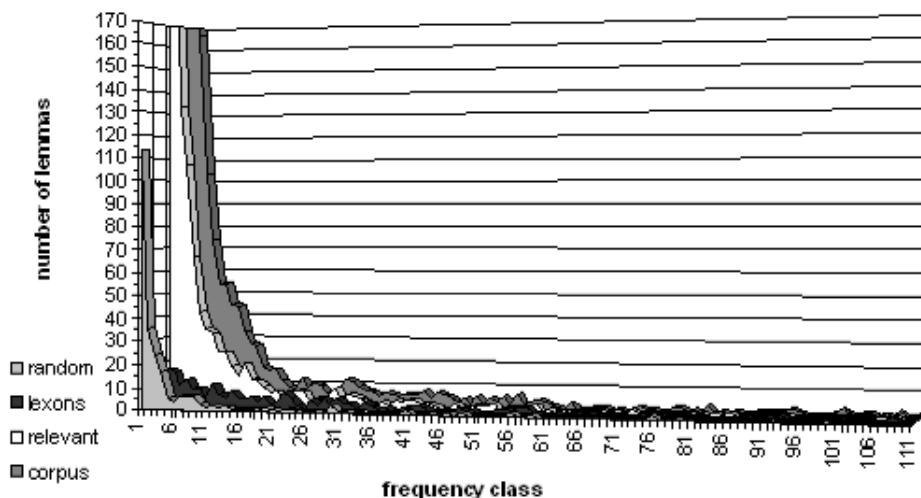


Fig. 1. absolute coverage of frequency classes

The unsupervised miner seemingly misses a lot of low frequency terms that are considered as typical of the VAT corpus. Even a naive random selection mechanism scores "better" for the FC 1 till (and including) 4.

4.2 Recall and Precision

The precision ratio is of 58.78% while the recall is 9.84%. In absolute numbers, it means that 211 lemmas have been selected as representing domain knowledge by both the unsupervised miner and the statistical comparison formula. Figure 2 shows the distribution of the recall ratios per frequency class. The averaged recall is 48,74% and the averaged precision is 58,27%.

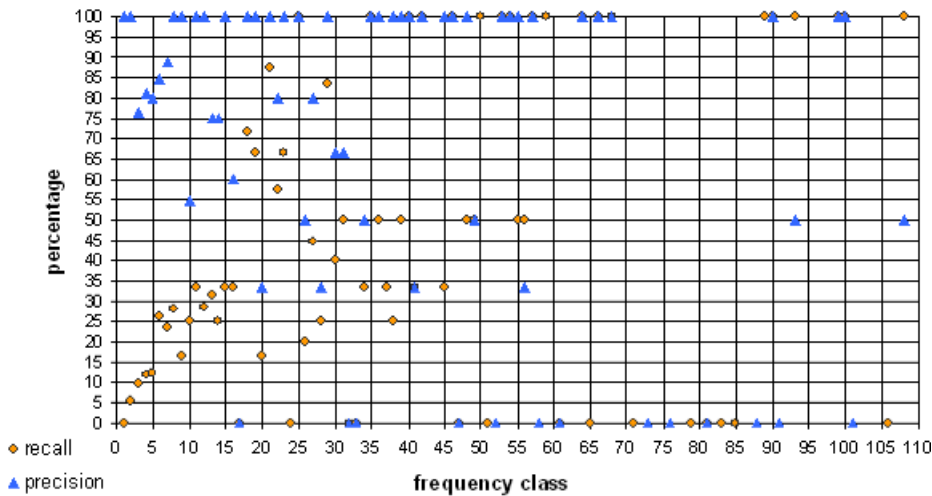


Fig. 2. recall and precision per frequency class

4.3 Accuracy

There are 34 typical frequency classes (i.e. containing at least 60% of words that are judged to be statistically typical). The classes are 1 - 5, 7, 13 -15, 18, 21, 22, 24 - 27, 29, 30, 34, 36 - 38, 45, 48, 51, 55, 57, 64 - 66, 68, 71, 79, 83, 85, 90, 99, 106, 111, 119, 121, 145, 169, 173, 181, 199, 219, 276, 277, 385, 597, 727, 1011, and 1378. It has to be noted that, from class 60 onwards the FCs only contain a single word and it is judged as typical (except for class 72: two words and both typical). The average coverage ratio (= the accuracy) for these 34 typical frequency classes is 52,1%.

5 Discussion

5.1 The Material

The two corpora exhibit the expected behaviour as expressed by Zipf’s law. The corresponding FCs of the two corpora contribute more or less to the same extent to the overall vocabulary. Therefore, it is rather disappointing that the text miner only attains low coverage and recall scores on the one hand, and it is surprising on the other that the lowest FCs are to be considered as relevant - see figure 3.

Table 1 illustrates the effect of applying the accuracy calculation. The first two data rows show Zipf’s law in practice (the top frequency classes contain empty words), while the other two data rows display the ten topmost typical frequency classes. Calculating the accuracy measure apparently does not adequately filter out the empty words or non relevant words.

A closer examination of the entire corpus revealed an important part of non-words (numbers of all kinds, section indications, percentages, typos, ...) in both the VAT (655 items or 20,24% of the lemmas) and WSJ corpora (6236 items or 14,71%). This is

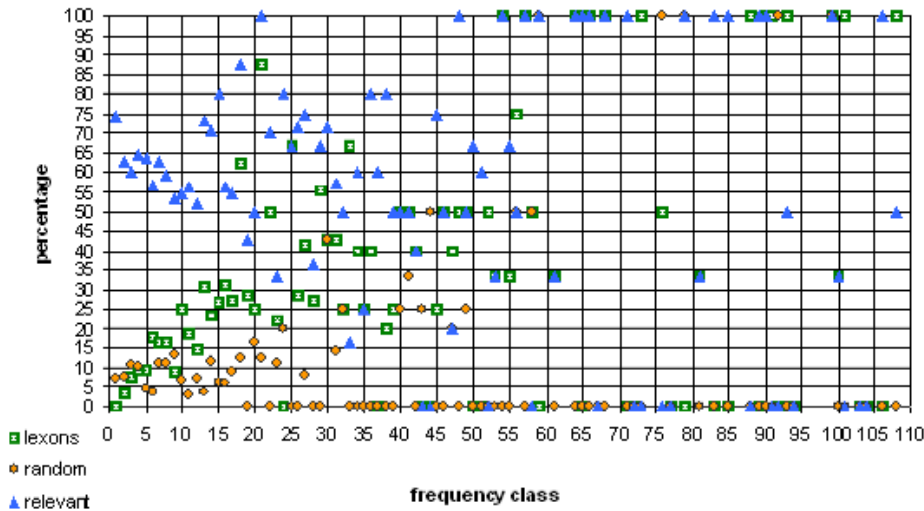


Fig. 3. relative coverage of frequency classes: by mining triples or lexons, by randomly picking words versus selecting statistically typical words

Table 1. Ten topmost frequency classes and their members (before and after accuracy calculation)

"raw" data									
FC	3573	2401	2011	1378	1260	1157	1110	1011	790
word	the	of	,	)	to	in	be	(	and
"accurate" data									
FC	1378	1011	727	597	385	277	276	219	199
word	)	(	or	shall	-	;	:	/	add
									refer

particularly annoying for the VAT corpus as almost all (551) of these non-words are considered to be characteristic (on a total of 1965 characteristic words). As, naturally, the text miner does not retain these non-words, the coverage and recall scores are biased in a negative way. These non-words also bias the accuracy score as they influence the status of a FC (being typical or not). This explains to a large extent why even very low FCs are considered as relevant FCs, which contradicts Zipf's and Luhn's findings. Therefore, we plan to redo the experiment, but with an adequate definition of what a "good" formally word consists of. A professional concordancy program (e.g., Word-Smith) will be used to this aim in a next iteration of the experiments.

Luckily, the precision score is not affected by this problem - see figure 2. A score of a bit less than 60% is not spectacularly good, but neither particularly bad. If we look at it from a positive angle, it means that a knowledge engineer disposes, with a sufficient degree of trust, of two thirds of the important domain words. It would be interesting to investigate which kind of notions the words represent. We believe that these words are representative for the "middle out" ontology engineering approach,

and therefore it is most likely that human domain experts are able to rapidly fill in the more general domain notions that are missing. More research, involving application domains of various nature, is needed to investigate how to mine the very specialised, and therefore, less often used notions. However, it is our intuition that the reduction of the cognitive load for a knowledge engineer (studying some 815 lexons instead of an entire text) is already substantial.

5.2 The Unsupervised Text Miner

The text miner clearly behaves in a non-random way: the distribution of the lemmas randomly picked follows the overall corpus distribution - see figure 1. Because of the high number of non-words in this experiment, it is almost certain that randomly picking words will produce a lot of garbage.

What is evident from this evaluation experiment is that the CNTS unsupervised text miner currently misses too many important notions, but that the results produced are of an acceptable quality. It is unclear to the authors how they would have reached this objective conclusion in a fast way without the support of the evaluation method reported on.

Some mistakes made by the shallow parser have a strong influence on the quality of the semantic extraction process. This happens if words unknown to the parser are improperly tagged, or if syntactic relations are missed or wrongly identified. The structure of the corpus also plays a role in that respect. The VAT corpus contains a lot of enumerations that are difficult to analyse for the parser due to the distance between the main verb and some complements. The fact that the shallow parser has not been trained on legal material plays a role as well. It is the nature of unsupervised mining that no tuning to a specific corpus is done. Therefore, the overall results are worse than with supervised mining. There is a trade-off to be made between resource investment and quality of the results.

Concerning the extraction of triples, the size of the corpus matters a lot as one common technique used to judge the appropriateness of a term relies on its frequency in the corpus. The extraction process of the text miner discards some relevant terms because they appear only once in the VAT corpus. A new experiment (without the non-words) including human validation should determine if the statistical thresholds of the unsupervised miner are to be relaxed.

5.3 The Evaluation Method

It is quite evident that the coverage measure is a too "naive" measure to be useful, except as an intermediary step to calculate the accuracy. Table 1 shows why. Recall and precision are considered traditionally as complementary (and are often combined in the F-measure). Accuracy could be an alternative to recall as it tries to somehow combine Luhn's theoretical findings on the resolving power of significant words with using a gold standard. More practical work should be done in order to validate this assumption.

Note that the evaluation method proposed does not give any indication on the correctness of a triples as a whole. It means that, if the words "fish" and "exception"

are typical of the application domain, the method cannot rate the triple <fish with exception> as invalid. We did not yet examine these aspects.

As already mentioned, the method stays on the word level. It is to be expected that grouping synonyms might improve the score, but it is unclear to what extent. Eventually some way of abstraction (especially for the RDF predicate or lexon role) will have to be done. Also these aspects require further investigations.

The important point of applying these metrics, how imperfect they currently might be, is that the scores can be used to monitor changes (preferably improvements) in the behaviour of the text miner (regression tests). As soon as the scores for a particular (and commonly agreed) textual source have been scientifically validated, the source and the scores together can become an evaluation standard in bench-marking tests involving other text miners, or even to some extent any RDF-based ontology producing tool. A logical next step would be that ontologies, automatically created by a text miner, are documented with performance scores on their textual source material as well as with scores for that particular text miner on the evaluation standard (commonly agreed text and outcomes).

6 Related Work

Previous reports on our work contain additional details on the unsupervised miner [22], its application to a bio-medical corpus [21], and a qualitative evaluation [25]. To the best of our knowledge, so far only one other approach has been presented that addresses the quantitative and automated evaluation of an ontology by referring to its source corpus.

Brewster and colleagues have recently presented a probabilistic measure to evaluate the best fit between a corpus and a set of ontologies as a maximised conditional probability of finding the corpus given an ontology. The specific probabilistic formula to compute the conditional probability of a concept label given a term occurrence takes synonyms into account mediating a form of query expansion [2-p.166]. It seems that some training needs to be done on basis of the annotated corpus, which is something we explicitly want to avoid with our approach. Unfortunately, no concrete results or test case are presented.

In addition, *Velardi* and colleagues have proposed to use the combination of "domain relevance" and "domain consensus" metrics to prune non domain terms from a set of candidate terms [30]. They use a set of texts typical of the domain next to other ones. *Domain relevance* is in fact the proportion of the relative frequency of a term in the domain text compared to the maximum relative frequency of that term over several non domain texts. *Domain consensus* is defined as the entropy of the distribution of a term in all the texts of the corpus. In our approach, we have computed the difference between two proportions, more specifically the z-values of the relative difference between the frequency of a word in a technical text vs. a general text (WSJ), which enables us to filter out words that are only seemingly typical of the technical text. In [18], the authors also present a method to semantically interpret novel complex terms with the help of WordNet and to organise them in a hierarchy. An evaluation of these latter aspects is also provided. Remark that both of

the proposed methods clearly (and correctly) differentiate a term or word from a concept.

Another statistical approach is elaborated by *Gillam and Tariq* [7] as part of a method to extract technical complex terms. They as well try to compare a specific text with a general text and characterise words by their weirdness (z-score for the ratio of the two relative frequencies of a word). More research is to be done to determine the exact difference with our approach.

Finally, although the main focus of the reports does not cover exactly the work presented here, the methods for ontology evaluation presented in [5, 12] can provide complementary information and inspiration. In particular, we could extend the notion of "relevant" as used above to an entire triple and define additional metrics, as has been done by Sabou [23] for extracted significant pairs. In the same vein, one could consider additionally the work of Maedche and Staab [16] who include the Levenshtein edit distance in their approach to measure the similarity between two ontologies. However, it is our conviction that the Levenshtein measure is too crude and naive to be of any use for our purposes.

In short, our approach is a first step to evaluate quantitatively and objectively triples generated by an unsupervised text miner. It does not aim directly at selecting relevant compounds terms and providing their semantic compositional interpretation. Although it would be interesting to see whether for our VAT corpus sensible interpretations could be generated using the structural semantic interconnections algorithm of Velardi and colleagues [18]. Also, their domain relevance measure, when applied to two texts, is equivalent to the corpus linguistic statistical formulas presented in section 3.3. could be an alternative metric to be taken into account for our evaluation purposes.

7 Future Work

There still remain some major points for improvement. An important issue is to extend the evaluation techniques presented above to multiple documents (i.e. by using the inverse document frequency (TF/IDF) metric, chi-square or the domain relevance and consensus metrics instead of simple word frequency for a single document). Although the unsupervised text miner detects compounds, the evaluation component currently takes only simple words into account. A compound detection module should thus be added.

Concerning the text miner itself, alternative statistical measures could be considered or thresholds could be relaxed to capture more low frequencies words. Additional syntactical patterns (e.g., subject - verb - prepositional object) should be retained. Ideally, the choice for a specific pattern should be done automatically in function of the structure and content of the corpus.

A next step would be to compare manually created ontologies with their source texts, which necessitates the integration of semantic distance measures such as the WordNet similarity functions [19] to operate on the meaning level instead of the linguistic level. Brewster et al. add two levels of WordNet hypernyms [2–p.166] for that purpose. That implies that (novel) compound terms should be assigned a semantic interpretation as is done by Navigli and Velardi [18].

8 Conclusion

We have presented the results of a simple quantitative evaluation method for the outcomes of an unsupervised mining algorithm applied to a financial corpus. Coverage, accuracy, recall and precision measures have been defined and calculated accordingly resulting in a 38.68%, 52.1%, 9.84% and 75.81% score respectively. These results (although biased because of the presence of many non-words in the corpus) have permitted us to identify a weak spot in the performance of the text miner, which will be improved in the future. New experiments to further validate the method are scheduled. An outline of a future research agenda has been given.

Acknowledgments. This research has been carried out during the OntoBasis project (IWT GBOU 2001 #10069), sponsored by the IWT Vlaanderen (Institution for the Promotion of Innovation by Science and Technology in Flanders). In addition, some parts have served as a contribution to the joint research activity program [12] of the EU FP6 IST NeO KnowledgeWeb (IST-2004-507482).

References

1. T. Berners-Lee, *Weaving the Web*, Harper, 1999.
2. Christopher Brewster, Harith Alani, Srinandan Dasmahapatra, and Yorick Wilks. Data Driven Ontology Evaluation. In, N. Shadbolt and K. O'Hara (eds.), *Advanced Knowledge Technologies: selected papers 2004*, pp. 164 – 164, 2004 (reprint from LREC2004).
3. Sabine Buchholz, Jorn Veenstra, and Walter Daelemans, Cascaded grammatical relation assignment, in *Proceedings of EMNLP/VLC-99*. PrintPartners Ipskamp, 1999.
4. Paul Buitelaar, Siegfried Handschuh, and Bernardo Magnini (eds.). *Proc. of the ECAI04 Workshop on Ontology Learning and Population*, 2004.
5. Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini (eds.). *Ontology Learning from Text: Methods, Applications and Evaluation*, IOS Press, Amsterdam, 2005 (forthcoming).
6. Josse De Kock. *Elementos para una estilística computacional - tomo I*. Editorial Coloquio, Madrid, 1984.
7. Lee Gillam and Mariam Tariq. Ontology via Terminology? In F. Ibekwe-San Juan and S. LainCruz (eds.), *Proceedings of the Workshop on Terminology, Ontology and Knowledge Representation*, 2004. <http://www.univ-lyon3.fr/partagedessavoirs/termino2004/programb.htm>
8. Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering*. Springer Verlag, 2003.
9. T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6 (2):199–221, 1993.
10. N. Guarino and P. Giaretta, 'Ontologies and knowledge bases: Towards a terminological clarification', in *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, ed., N. Mars, pp. 25 – 32, IOS Press, Amsterdam, 1995.
11. Nicola Guarino. Towards a Formal Evaluation of ontological Quality. *IEEE Intelligent System*, 19 (4):78–80, 2004.
12. Jens Hartmann, Peter Spyns, Diane Maynard, Roberta Cuel, Mari Carmen Suarez de Figueroa and York Sure. Methods for Ontology Evaluation, KnowledgeWeb Deliverable #D1.2.3, 2005.

13. B. Humphreys and D. Lindberg. The unified medical language system project: : a distributed experiment in improving access to biomedical information. In K.C. Lun, (ed.), *Proc. of the 7th World Congress on Medical Informatics (MEDINFO92)*, pp. 1496–1500, 1992.
14. H. Karanikas and B. Theodoulidis, ‘Knowledge discovery in text and text mining software’, Technical report, UMIST - CRIM, Manchester, 2002.
15. H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, **2** (2):159 – 195, 1958.
16. Alexander Maedche and Steffen Staab. Measuring Similarity between Ontologies. In, *Proceedings Of the European Conference on Knowledge Acquisition and Management (EKAW02)*, pp. 251 – 263, LNAI 2473, Springer Verlag, 2002
17. Robert Meersman. Ontologies and databases: More than a fleeting resemblance. In A. d’Atri and M. Missikoff (eds.), *OES/SEO 2001 Rome Workshop*. Luiss Publications, 2001.
18. Roberto Navigli and Paola Velardi. Learning Domain Ontologies from Document Warehouses and Dedicated Web Sites. *Computational Linguistics*, **30** (2):151–179, 2004.
19. T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *The Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.
20. Marie-Laure Reinberger, Peter Spyns, Walter Daelemans, and Robert Meersman. Mining for lexons: Applying unsupervised learning methods to create ontology bases. In Robert Meersman, Zahir Tari, and Douglas Schmidt et al. (eds.), *On the Move to Meaningful Internet Systems 2003: CoopIS, DOA and ODBASE*, LNCS 2888, pp. 803 – 819, 2003. Springer.
21. Marie-Laure Reinberger, Peter Spyns, A. Johannes Pretorius, and Walter Daelemans. Automatic initiation of an ontology. In Robert Meersman, Zahir Tari et al. (eds.), *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA and ODBASE (part I)*, LNCS 3290 , pp. 600 – 617 , 2004. Springer Verlag.
22. Marie-Laure Reinberger and Peter Spyns. Unsupervised Text Mining for the Learning of DOGMA-inspired Ontologies. In P. Buitelaar, Ph. Cimiano, and B. Magnini, (eds.), *Ontology Learning from Text: Methods, Applications and Evaluation*, IOS Press Amsterdam, 2005.
23. Marta Sabou. Extracting Ontologies from Software Documentation: a Semi-automatic Method and its Evaluation. In P. Buitelaar, Ph. Cimiano, and B. Magnini, (eds.), *Ontology Learning from Text: Methods, Applications and Evaluation*, IOS Press Amsterdam, 2005.
24. Peter Spyns, Robert Meersman, and Mustafa Jarrar. Data modelling versus ontology engineering. *SIGMOD Record Special Issue*, **31** (4):12–17, 2002.
25. Peter Spyns, A. Johannes Pretorius and Marie-Laure Reinberger. Evaluating DOGMA-lexons generated automatically from a text corpus. In Cimiano P., Ciravegna F., Motta E. and Uren V. (eds.), *Proceedings of the EKAW2004 Workshop on Human Language Technology and Knowledge Management*, pp. 38 – 44, 2004.
26. Peter Spyns and Jan De Bo. Ontologies: a revamped cross-disciplinary buzzword or a truly promising interdisciplinary research topic? *Linguistica Antverpiensia, new series* (3), 2004 (forthcoming).
27. M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Sharing and Review*, **11** (2), June 1996.
28. M. Ushold, ‘Where are the semantics in the semantic web?’, *AI Magazine*, **24** (3):25 – 36, 2003.
29. C. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.
30. Paola Velardi, Michele Missikoff, and Roberto Basili. Identification of relevant terms to support the construction of Domain Ontologies. In Maybury M., Bernsen N., and Krauwert S. (eds.)*Proc. of the ACL-EACL Workshop on Human Language Technologies*, 2001.
31. George K. Zipf. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, Cambridge MA, 1949.
32. Tawk Compiler v.5. Thompson Automation Software, Jefferson OR, US.

Pedro Ontology Services: A Framework for Rapid Ontology Markup

Kevin Garwood¹, Phillip Lord¹, Helen Parkinson²,
Norman W. Paton¹, and Carole Goble¹

¹ Department of Computer Science,
University of Manchester, Oxford Road,
Manchester M13 9PL, UK

² European Bioinformatics Institute,
Cambridge M13 9PL, UK
`kevin.garwood@cs.man.ac.uk`
`http://pedro.man.ac.uk`

Abstract. Semantic Web technologies offer the possibility of increased accuracy and completeness in search and retrieval operations. In recent years, curators of data resources have begun favouring the use of ontologies over the use of free text entries. Generally this has been done by marking up existing database records with “annotations” that contain ontology term references. Although there are a number of tools available for developing ontologies, there are few generic resources for enabling this annotation process. This paper examines the requirements for such an annotation tool, and describes the design and implementation of the Pedro Ontology Service Framework, which seeks to fulfill these requirements.

1 Introduction

The development of many high-throughput technologies has industrialised the production of laboratory data. This has led to increased opportunities for performing biological *in silico* experiments. While some aspects of the data can be characterised by formal data models, significant amounts of biological information are represented as free text or semi-structured data. Experiments are often annotated with free text that describes important aspects such as the experimental techniques used. This annotation enables biologists both to make sense of main data sources and to conduct useful searches.

Traditionally, many different formats and formalisms have been used for database annotation. While it is not a formalism per se, free text has and continues to be the most common way of annotating databases. Although this has the advantage of expressivity, it responds to only limited forms of computational searching or comparison. The simplest solution to this difficulty is the use of controlled vocabularies. This approach provides limited expressivity unless the vocabularies are made very large, in which case they cease to be controlled.

More recently, there has been great interest in the application of ontological technologies, particularly since the advent of the Gene Ontology [2], which has been widely adopted.

One of the difficulties of applying ontology technology in this way has been the absence of appropriate tools for generating appropriate ontological annotations. Most of the effort made by the Semantic Web community has focused either on providing annotation in a single formalism, or on providing tools for generating ontologies (e.g. [3, 4]). While these are important areas of development, there is a need for tools which support a variety of data sources that in turn support different annotation formalisms. In this paper, we describe these requirements in more detail and the design and implementation of the Pedro tool, which seeks to fulfill them.

2 The Case Studies

The *myGrid* project has developed a service-oriented architecture to enable bioinformaticians to: gather distributed data; use data and analysis tools presented as services; compose and enact workflows; and to manage the generated [9] data. There are now more than a thousand different services available for use, which creates a substantial difficulty in terms of service selection. Therefore, *myGrid* has sought to apply user-oriented semantic service selection to support users in their decisions [7]. The project has required tool support for generating semantic descriptions of the services it provides. An ontology of approximately 500 classes was developed, initially using DAML+OIL and later OWL. It is deployed as an asserted hierarchy represented as RDF(S) [6]. Currently, the *myGrid* ontology and associated descriptions are being managed in a relatively informal manner; there is no formal commitment to the maintenance the ontology as a standard.

The Microarray Gene Expression Data (MGED) Society (<http://www.mged.org>) defines standards for the representation of micro-array information, which describes levels of gene expression within some defined sample. Often complex, these data describe the experimental procedures used to gather the data; information about the source of the sample (e.g. the source species, the anatomical location, cell type etc.) and the experimental context. Consequently, MGED has defined the MGED ontology (MO) [10], which comprises approximately 100 concepts using OWL. Hundreds of thousands of records are expected to be annotated with terms from this ontology. The MO has been crafted using a layered approach: while one layer of core information remains static, other layers of the ontology are allowed to change as knowledge evolves.

The case studies share many common themes that are directly relevant to activities of the Semantic Web. In each case, there is an attempt to model many different kinds of data within the same data set. They try to capture both knowledge about biological systems and knowledge about the context of the experiments. For the microarray case study, the context can include descriptions of laboratory equipment and procedures. In *myGrid*, the context describes aspects

of the *in silico* experiment. Both projects have attempted to foster reuse of autonomous data sets by carefully crafting model relationships that link the sets. Most important, the projects share the common goal of enabling high fidelity retrieval of data.

MGED has the additional aim of supporting data mining activities. It is apparent that the quantity of micro-array data stored will greatly increase, which will require that it be managed at different sites. The efficacy of queries applied across multiple autonomous data sets will critically depend on technologies that allow the structure and content of the experiment records to be clearly defined.

3 User Roles and Ontology Life Cycle

One of the key features of knowledge engineering in bioinformatics is the need for community involvement in the development of schemas and ontologies. Probably the best known and most widely used ontology is the Gene Ontology (GO), a Directed Acyclic Graph (DAG) of terms describing the function, biological role and sub-cellular localisation of gene products. This ontology now has approximately 17,000 terms and several million annotated instances. The key reason for its success has not been the adoption of a particular formalism, but its social engagement with its community of users [2]. In this setting, different users play different roles, such as:

Schema Developer: Responsible for developing a data model to which data must conform. The schema developers may be working in the context of a Standards Committee, such as MGED, which seeks to ensure that the model supports the requirements of a wide user community.

Knowledge Engineer: Responsible for the generation and curation of the ontologies that are used within the schema.

Data Provider: Responsible for the generation of data sets according to the schema and using the ontologies.

Data Consumer: Responsible for making effective and systematic use of the data sets generated.

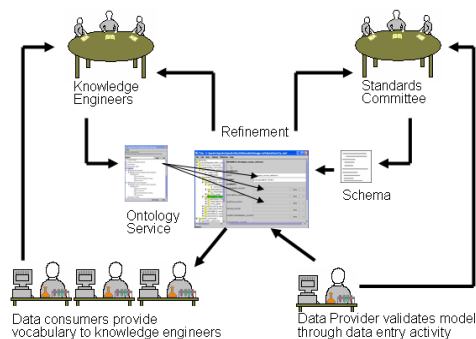


Fig. 1. The knowledge engineering life cycle

Although these roles are different, specific individuals within the community may fulfil more than one. These different user communities are involved in a development life cycle depicted in Figure 1. As well as producing data, the data providers are critical in providing feedback to the standards committee regarding ease of use and coverage of their standard. Similarly, the data consumers are heavily involved in the knowledge engineers' work. Most of them are highly specialised and provide the knowledge required to model their sub-domain.

4 The Requirements

In this section, we define the key requirements for knowledge acquisition within the specific case studies described in the last section.

Rapid Modelling: Bioinformatics is a large and complex domain; modelling even a small part of it has proven to be extremely challenging. Moreover, within this domain, a council of perfection is a council of despair: any domain model will be wrong in the initial stages and will need to be evolved iteratively before it approaches correctness. Consequently, there is a strong requirement for a tool that enables collection of data, and which is resilient to change in the schema by which that data is organised.

Ontological Annotation: Not all of the data we wish to support are represented ontologically. Some kinds of information, such as probabilistic or numeric data, cannot be well represented using ontologies. Other kinds of information are represented using legacy formalisms. It is unlikely these problems will be fully addressed as the field of bioinformatics evolves. Therefore, the requirement is not to generate instance data according to some ontology, but to use terms from standard ontologies to annotate aspects of the data within some schema.

Distributed and Autonomous Ontologies: Although a Standards Committee may control the overall schema, they do not necessarily control the development of the different ontologies in use to populate the fields of the schema. Any tool must be flexible enough to adapt to independent evolution of the ontologies used.

Multiple Formalisms: As well as existing legacy data, multiple different formalisms for ontological data are required. In the simplest case, an ontology may be represented as a controlled vocabulary. A more sophisticated form could be a directed acyclic graph, which is the most common representation within bioinformatics [2]. Other forms include the RDF(S) representation used within projects such as *myGrid* [6] and full property based OWL representations such as those used by the MGED ontology.

Multiple Ontology Views: Biologists tend to have strong aversions to filling in forms¹. This has effected the development of expressive ontologies that

¹ Our experiences suggest the problem is somewhat wider: biologists dislike most methods of knowledge acquisition and that this dislike is shared by people other than biologists!

describe anatomy. While large anatomy ontologies are available to the bioinformatics community, work has recently begun on small controlled vocabularies aimed at reducing the complexity of form filling [1], while maintaining the link back towards the more expressive ontologies. Therefore, it is clear that as well as supporting multiple underlying ontology formalisms, we need to support multiple different views of these ontologies (and sometimes of the same ontology!) to support the needs of the different user bases within the community.

5 The Architecture

This section describes the architecture of the Pedro software, and in particular its ontology framework. The application supports data capture using screens of the form illustrated in Figure 2. The tree view in the left hand panel illustrates the structure of the model, and the data entry form in the right hand panel is being used to capture the properties of a specific *BIOSAMPLE*. The overall architecture of the Pedro software is illustrated in Figure 3.

The principal components of the architecture are described in the following subsections.

5.1 Model Manager

The model manager provides access to the data model that is to be viewed or manipulated. XML Schema is used as the primary mechanism for schema representation, although the adaptor interface could in principle be used to provide

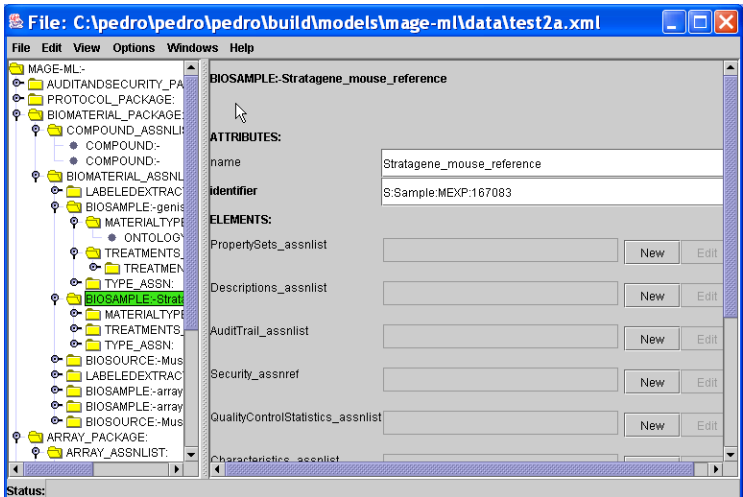


Fig. 2. Browsing and manipulating the MAGE-ML microarray data model using the Pedro Data Capture Tool

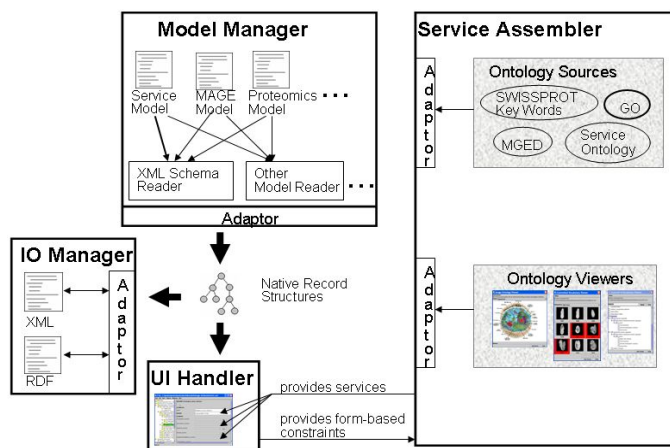


Fig. 3. The Pedro architecture

access to schemata described using different data models. XML is becoming widely used in bioinformatics, and forms the basis of several data standards activities (e.g. [8, 11]).

The Model Manager reads an XML Schema, along with a configuration file that indicates where special behaviours are to be associated with parts of the model (e.g. a configuration file entry could be used to indicate that the values for a particular element are to be obtained from a specific ontology). The hierarchical structure of the XML Schema is reflected in the tree view, which is used both to provide an overview of the structure of the data conforming to the model and to identify where data are to be added or modified.

5.2 Ontology Service

The ontology service provides access to external resources that support the population of XML elements with values drawn from external ontologies. For each kind of external resource, it is necessary: (i) that an adaptor interface has been implemented that provides access to ontologies of the relevant type; and (ii) that a viewer is available that is appropriate for presenting the values from the ontology to annotators. Thus, as illustrated in Figure 4, an element within the model can be associated with both an source location for terms, and an appropriate viewer. Both the *Ontology Source* and the *Ontology Viewer* are defined as Java interfaces, enabling full independent implementation of these components. Through the configuration layer, it is possible to associate one or more ontology services with a given field. This reflects the reality of bioinformatics: that there are often overlapping and non-orthogonal ontologies.

Because of the requirement to support multiple, different formalisms, the Ontology Service interface does not exploit their different levels of expressivity.

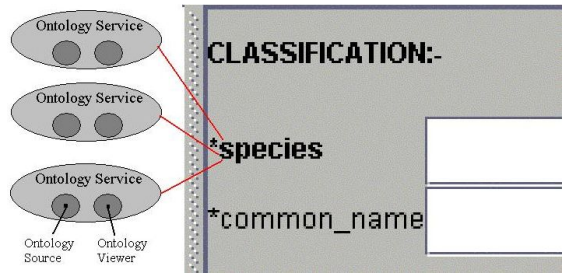


Fig. 4. Associating elements with ontology services

Currently, Pedro provides a number of different default Ontology Sources that can read from local resources. These include:

- A simple text list:** which provides a straightforward mechanism for the deployment of unstructured controlled vocabularies.
- A tab indented list:** which provides a mechanism for representing controlled vocabularies organised as a tree.

Currently, most of our Ontology Sources use local copies of the distributed ontologies because this suits requirements for shrink-wrapped software. However, with the increasing uptake of programmatic interfaces providing access to Ontological terms [1], we expect that this situation will change, with the majority of ontologies being served remotely.

5.3 UI Handler

Following the standard Model-View-Controller design pattern, the *Model Manager* represents the instances defined according to the underlying XML Schema as Java objects, and the *UI Handler* is responsible for rendering this model as a set of Java interface components. While the choice of a model-driven UI fulfills the requirement of a tool that is resilient to change in the underlying models, we are aware that such a generic approach may produce an interface that is less than ideal for specific users or types of data. In this context, the *Ontology Viewer* offers a key abstraction, separating the concern of term selection from that of serving the ontology. This allows different viewers to be selected based on the user community, or on the size or nature of the ontology.

In many cases, the size of the ontology in use is relatively small, while the number of annotated records is much larger. For this reason, providing convenient “in place” access to ontology terms is the most common mechanism for term selection. Figure 5, which uses an example from *myGrid*, shows the selection of terms describing the functionality of a web service. The rapidity of these interface is of critical importance for data producers who generate large numbers of records. In this figure, we also show the use of “anchoring”. The *myGrid* ontology’s some 500 classes are too numerous to place in a context menu when only some of these are appropriate for use within the current form field. It

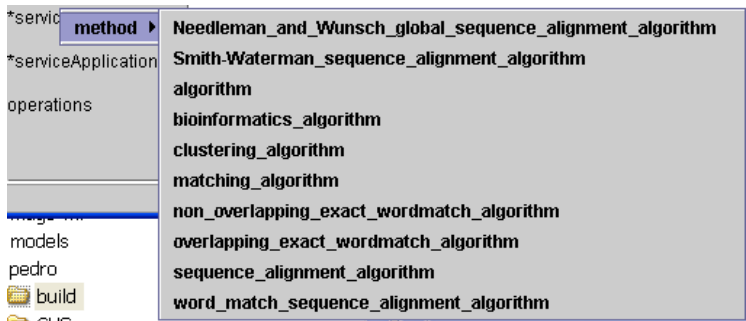


Fig. 5. Invoking ontology services through a right-click menu

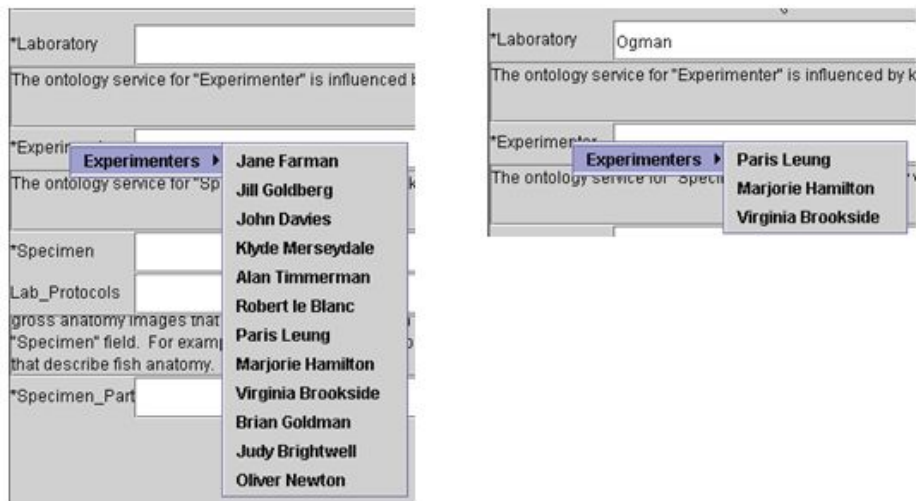


Fig. 6. The use of Ontology Context: This figure shows a right-click menu offering terms from a controlled vocabulary

is possible to configure Pedro to display only this appropriate subset of classes from the context menu.

While anchoring can reduce the number of concepts to a manageable size, it is not always possible to determine the appropriate subset at design time. For this reason, Pedro supports the notion of “Ontology Contexts”; the values of local fields can be used to restrict the concepts. In Figure 6, the presence of a “laboratory” field is used to restrict later fields to only the members of this laboratory. This Ontology Context functionality is a property of the Ontology Service; Pedro’s support for multiple formalisms means that the task of expressing constraints must be devolved to its ontology framework.

When appropriate anchors or contexts cannot be used to restrict the number of concepts on display, Pedro uses a component that displays as a table or a

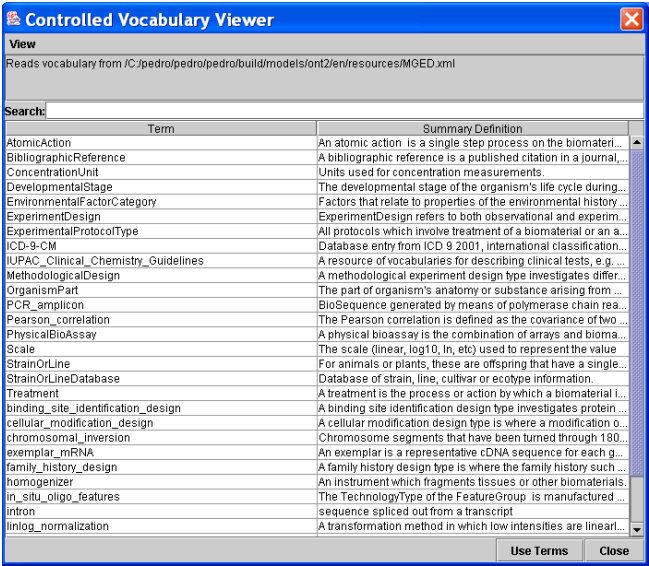


Fig. 7. An ontology viewer that visualises terms in a table with term and definition columns

tree as shown in Figure 7. In our experience, viewing as a table is often the most appropriate representation. While the structure of an ontology is intrinsic to its functionality, data providers are often intimately knowledgeable about the terms available and already know which term they wish to use. The table view provides a simple mechanism for rapidly selecting such terms.

Pedro also provides “type ahead” incremental search functionality for term selection which, again, enables rapid use. However, not all data providers have the level of intimate knowledge required to use such a facility. Moreover, biological knowledge is often hard to express textually. For this reason, Pedro also provides a selection of image-based Ontology Viewers. In Figure 8(a), we show selection of terms based on an image map, different parts of the image corresponding to different sections of the ontology. This is useful for an ontology that describes concepts which have a spatial relationship to each other. In Figure 8(b), a set of thumbnail images is presented to the data provider, each one representing a specific ontology term.

As well as providing convenient access to ontology terms for some users, the support of images has another advantage: implicit internationalisation. This is a significant problem in biology, where data providers are often geographically distributed, and translation of technical terminology used in both concept names and associated documentation is difficult and expensive. It is also apparent that the use of images, to represent ontological terms, could have significant advantages for generating queries over data.

Currently, the ontology views within Pedro are limited to the selection of named concepts, rather than general class expressions. We would like to inves-

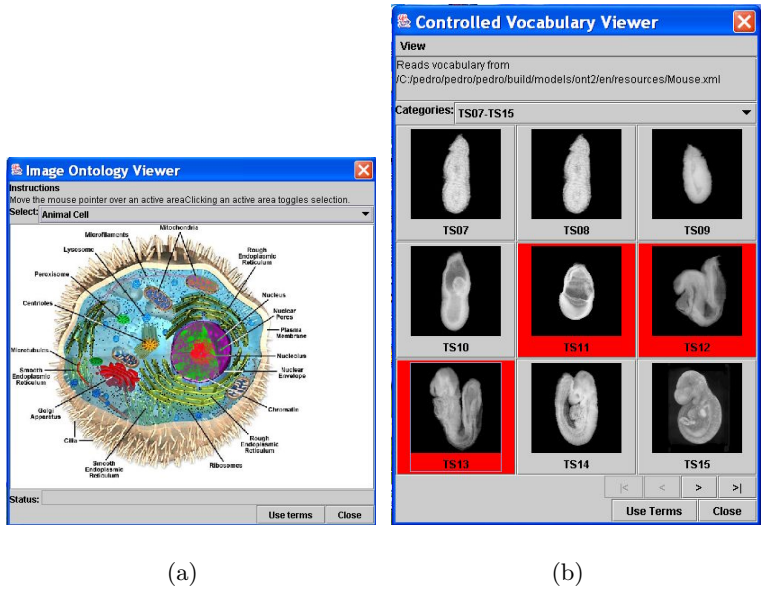


Fig. 8. Selection of terms with images

tigate adding more capabilities to Pedro, as widget sets for the easy generation of such expressions become available [5]. However, currently the use of reasoning technology in the downstream applications of *my*Grid and MGED is limited. Therefore, the use of these expressions would require significant changes to the architecture of these applications.

5.4 IO Manager

The IO Manager supports the reading and writing of part or all of a data set from or to an external resource. Again, this functionality is pluggable; Pedro comes with IO managers for XML file formats and for an internal representation. However, additional IO manager components have been written that, for example: (i) read data from a relational database into specific elements in a model; and (ii) use an XML database as the source or destination of data that is to be updated using Pedro.

6 Meeting the Requirements

Section 4 described various requirements for data annotation tools in bioinformatics. This section revisits the requirements, indicating the approach taken by Pedro to try to satisfy the requirements and the level of support provided.

Rapid Modelling: Although Pedro was designed to support data capture and annotation, and not data modelling, in practice several projects have exploited Pedro as part of an iterative modelling activity. That the Pedro interface is model-driven means that a data capture interface can be created directly from a proposed data model. A schema designer can thus develop a version of a model and validate the model by leading the users through a data capture task using the latest version of the model. This has proved an effective way of detecting both errors of omission and commission. Data models change frequently in bioinformatics, reflecting an evolution both in experimental practice and in understanding. Therefore, it is important that bioinformatics tools can be readily evolved to work with new models. Pedro provides no direct support for the versioning of models or ontologies, but its model-driven architecture means that minimal coding is likely to be required to take account of changes to external models. Where software does need to be changed, these changes are likely to be contained within specific adaptors (e.g. within the IO Manager, where the storage format of an external data resource is modified).

Ontological Annotation: Pedro integrates ontological annotation with other aspects of data capture by associating elements in an XML Schema with ontology sources and viewers. As such, in Pedro, values for elements can be obtained: (i) through direct user entry; (ii) by selecting from a list of *enumeration* values within the Schema; (iii) by reading values from tab-delimited files using an interactively tailorable import facility; (iv) by reading values from a database using an *IO Manager* adaptor; or (v) by obtaining a value from an ontology accessed through an *Ontology Source* adaptor and presented using visual representations accessed using an *Ontology Viewer* adaptor. Therefore, annotation using ontologies is one of several pluggable components of the Pedro architecture, whereby annotation using ontologies is seamlessly integrated with other forms of data capture.

Distributed and Autonomous Ontologies: While a Standards Committee may control an overall schema, the committee is unlikely to exercise centralised control over the development and use of different ontologies. For example, different ontologies may be applied to different families of organism or environmental settings. Pedro supports the use of different ontologies and ontology languages by using pluggable adaptors to access either bundled lists of terms or programmatic interfaces to external reasoning services.

Multiple Formalisms: In a community that decides to develop different ontologies, individual groups may choose to use different languages. Pedro communicates with ontology services via an adaptor interface that reflects what the user is offered. The interface shields the rest of the software from details about how offerings are made (e.g. this may be by looking-up an asserted ontology, or by inferring relationships in a description logic ontology).

Multiple Ontology Views: Considering the diversity in user communities and the variability of size and complexity of ontologies, there is no single best way to present terms to end-users. Therefore, Pedro provides an adaptor

interface for ontology viewers that has been used to support a broad range of visualisations (Section 5.3). Several of these representations have been widely used in practice, although no systematic usability evaluations have yet been conducted.

Pedro's extensible, model-based architecture can provide some measure of support for a wide range of requirements. We see the development of tools that integrate ontologies with other aspects of an application as being important to their efficient and effective deployment in challenging domains such as bioinformatics.

7 Discussion

Bioinformatics is already a significant adopter of ontological and semantic web technologies because they allow data sets to be indexed and retrieved using expressive domain models. However, the community wants to adopt these technologies in an incremental fashion. Scientists may initially want to add ontological annotation to existing database records, rather than recasting all of their data in an ontological formalism.

The Pedro Ontology Service Framework provides a common point of entry, from within the Pedro data capture tool, which enables users to generate potentially rich and contextualised annotations, using multiple distributed and autonomous ontologies, with multiple different and independent formalisms.

The Pedro Ontology Service Framework allows the Pedro data capture tool to uniformly access multiple, distributed, autonomous ontologies, each having its own formalisms. Using the tool's ontology services, end-users can generate contextualised annotations.

These capabilities have ensured that it has already found significant use in independent projects within the bioinformatics domain. We anticipate that its ontology-based annotation capabilities will also be of significant interest to other domains.

Most of the efforts of the Semantic Web community have focused on developing tools that demonstrate the application of a specific formalism. Relatively little effort has been spent on making end-user tools. We suggest that our experiences designing for these use cases will help spur the development of more open, adaptable Semantic Web technologies.

In aiming to support a highly iterative style of knowledge capture and engineering, we have also been surprised by some requirements. In most cases, where Pedro offers the end-users terms from an ontology, schema designers have generally also allowed them to enter free text noun phrases. While this defeats the purpose of using controlled vocabularies, it suggests that designers believe there is a possibility users will not find the terms that they are looking for. Moreover this capability provides significant feedback to the knowledge engineers, who often wish to incorporate these terms into later versions of their ontologies.

Significant future work remains for Pedro to fulfill its potential. Currently its weakest area of development is its treatment of versioning and change management. Different ontology communities use different methods for representing updates. This is a severe problem for those performing rolling updates on a daily basis. This lack of common procedures between different groups reflects the lack of clear best practices within the community at large. If these experiences are reflected in the Semantic Web community, it will present a significant barrier to adoption of these technologies. Currently, the Pedro framework provides a rudimentary abstraction over these different methods, devolving the task of change management to the various Ontology Service providers; but we are actively seeking ways to improve this abstraction.

Availability: the Pedro software is freely available in open source form from <http://pedro.man.ac.uk/>; at the time of writing Pedro has been downloaded around 1000 times, and is being used in a wide range of application communities.

Acknowledgements. this work is supported by the UK e-Science Programme *my*Grid and North-West Regional e-Science Centre grants, and through a BB-SRC grant under the Proteomics and Cell Function Initiative.

References

1. Start Aitken, Richard Baldock, Jonathan Bard, Albert Burger, Duncan Davidson, Terry Hayamizu, Helen Parkinson, Alan Rector, Martin Ringwald, Jeremy Rogers, Cornelius Rosse, and Chris Stoeckert. The SOFG Anatomy Entry List (SAEL): an annotation tool for functional genomics data. *Comparative and Functional Genomics*, 2005. *In Press*.
2. Michael Bada, Robert Stevens, Carole Goble, Yolanda Gil, Michael Ashburner, Judith A. Blake, J. Michael Cherry, Midori Harris, and Suzanna Lewis. A Short Study on the Success of the Gene Ontology. Accepted for publication in the *Journal of Web Semantics*, 2004.
3. S. Bechhofer, R. Möller, and P. Crowther. The dig description logic interface. In *Description Logics*. CEUR Workshop Proceedings, 2003.
4. H. Knublauch, R.W. Fergerson, N.F. Noy, and M.A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In *3rd International Semantic Web Conference*, pages 229–243, 2004.
5. Holger Knublauch, Mark A. Musen, and Alan L. Rector. Editing description logic ontologies with the Protégé owl plugin. In *International Workshop on Description Logics*, Whistler, BC, Canada, 2004.
6. Phillip Lord, Pinar Alper, Chris Wroe, and Carole Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *European Semantic Web Conference*. Accepted for Publication, 2005.
7. Phillip Lord, Sean Bechhofer, Mark D. Wilkinson, Gary Schiltz, Damian Gessler, Duncan Hull, Carole Goble, and Lincoln Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *International Semantic Web Conference*, pages 350–364, 2004.

8. P.T. Spellman et al. Design and implementation of microarray gene expression markup language (mage-ml). *Genome Biology*, 3(9):research0046.1–0046.9, 2002.
9. R.D. Stevens, H.J. Tipney, C.J. Wroe, T.M. Oinn, M. Senger, P.W. Lord, C.A. Goble, A. Brass, and M. Tassabehji. Exploring Williams Beuren Syndrome Using *my*Grid. In *Bioinformatics*, volume 20, pages i303–310, 2004. Intelligent Systems for Molecular Biology (ISMB) 2004.
10. C.J. Stoeckert and H. Parkinson. The mged ontology: a framework for describing functional genomics experiments. *Comp. Funct. Genom.*, 4:127–132, 2003.
11. C.F. Taylor et al. A systematic approach to modeling, capturing and disseminating proteomics experimental data. *Nature Biotech.*, 21(3):247–254, 2003.

Semantic Annotation of Images and Videos for Multimedia Analysis

Stephan Bloehdorn¹, Kosmas Petridis², Carsten Saathoff³,
Nikos Simou⁴, Vassilis Tzouvaras⁴, Yannis Avrithis⁴,
Siegfried Handschuh¹, Yiannis Kompatsiaris²,
Steffen Staab³, and Michael G. Strintzis²

¹ University of Karlsruhe,
Institute AIFB, D-76128 Karlsruhe, Germany

² Informatics and Telematics Institute,
GR-57001 Thessaloniki, Greece

³ University of Koblenz-Landau,
Institute for Computer Science, D-56016 Koblenz, Germany

⁴ National Technical University of Athens,
School of Electrical and Computer Engineering,
GR-15773 Zographou, Athens, Greece

Abstract. Annotations of multimedia documents typically have been pursued in two different directions. Either previous approaches have focused on low level descriptors, such as *dominant color*, or they have focused on the content dimension and corresponding annotations, such as *person* or *vehicle*. In this paper, we present a software environment to bridge between the two directions. *M-OntoMat-Annotizer* allows for linking low level MPEG-7 visual descriptions to conventional Semantic Web ontologies and annotations. We use *M-OntoMat-Annotizer* in order to construct ontologies that include prototypical instances of high-level domain concepts together with a formal specification of corresponding visual descriptors. Thus, we formalize the interrelationship of high- and low-level multimedia concept descriptions allowing for new kinds of multimedia content analysis and reasoning.

1 Introduction

Representation and semantic annotation of multimedia content have been identified as important steps towards more efficient manipulation and retrieval of visual media. Although new multimedia standards, such as MPEG-4 and MPEG-7 [1], provide important functionalities for the manipulation and transmission of objects and associated metadata, the extraction of semantic descriptions and annotation of the content with the corresponding metadata is out of the scope of these standards and is left to the content manager. This motivates heavy research efforts in the direction of automatic annotation of multimedia content.

Here, we recognize a broad chasm between current multimedia analysis methods and tools on the one hand and semantic annotation methods and tools on the other hand. State-of-the-art multimedia analysis systems are severely limiting themselves by

resorting mostly to visual descriptions at a very low level, e.g. the *dominant color* of a picture. This may be observed even though the need for semantic descriptions that help to bridge the so called *semantic gap* has been acknowledged for a long time [2, 3]. At the same time, the semantic annotation community has only recently started to work into the direction of semantic annotation in the multimedia domain and still remains a long way to go. Work in semantic annotation currently addresses mainly textual resources [4] or simple annotation of photographs [5, 6].

Acknowledging both the relevance of low-level visual descriptions as well as a formal, uniform machine-processable representation [7], we here try to bridge the chasm by providing a semantic annotation framework and corresponding tool, *M-OntoMat Annotizer*, for eliciting and representing knowledge both about the *content domain* and the *visual characteristics* of multimedia data itself. Specifically, MPEG-7 compliant low-level multimedia features are associated with semantic concepts thus forming an a-priori knowledge base.

In the framework we propose, this link between the MPEG-7 visual descriptors and domain concepts is made explicit by means of a conceptualization based on a *prototyping* approach. The core idea of our approach lies in a way to associate concepts with instances that are deemed to be prototypical by their annotators with regard to their visual characteristics. To establish this semantic link we have implemented our framework in a user-friendly annotation tool, *M-OntoMat-Annotizer*, extending our previous framework for semantic annotations of text [4]. The tool has been built in order to allow content providers to annotate visual descriptors without prior expertise in semantic web technologies or multimedia analysis.

The existence of such a knowledge base may be exploited in a variety of ways. In particular, we envision its exploitation in two modes:

(1) *Direct exploitation:* In this mode, an application uses the knowledge base directly. For instance, during the semantic annotation process one may gather information like *the blue cotton cloth 4711 in image 12 has a rippled texture described by values 12346546*. Such kind of semantic knowledge may be used later, e.g. for combined retrieval by semantics and similarity in an internet shop. Obviously, such kind of knowledge is expensive to be acquired manually, even when resorting to a user friendly tool. Thus, this kind of knowledge may only be provided for valuable data, such as images or videos of commercial products or of items from museum archives.

(2) *Indirect exploitation:* In this mode, the a-priori knowledge base ‘only’ serves as a data set provided to prepare an automatic multimedia analysis tool. For instance, consider the provider of a sports portal offering powerful access to his database on tennis, soccer, etc. He uses semantic annotation of multimedia images or videos in order to prepare an analysis system. For instance, he uses *M-OntoMat-Annotizer* in order to describe the shape and the texture of tennis balls, rackets, nets, or courts and he feeds these descriptions into an analysis system. The system uses the descriptions in order to learn how to tag and relate segments of images and video keyframes with domain ontology concepts. A customer at the portal may then ask the system what it could derive about the images and the videos, e.g. he could ask for all the scenes in which a ball touches a line in a tennis court.

Our long term objectives are dedicated to the indirect exploitation of semantic multimedia annotation as presented in the second paragraph, which is an ongoing comprehensive and complex endeavor, providing a flexible infrastructure for further multimedia content analysis and reasoning, object recognition, metadata generation, indexing and retrieval. In the context of this paper, we only sketch the main steps of our approach.

During image/video analysis, a set of atom-regions is generated by an initial segmentation of images, video sequences and key frames into areas corresponding to salient semantic objects. These objects are also tracked over time while MPEG-7 visual descriptors are extracted for each region. A distance measure between these descriptors and the ones of the prototype instances included in the domain ontology is estimated using a neural network approach. A genetic algorithm then decides the initial labelling of the atom regions with a set of hypotheses, where each hypothesis represents a class from the domain ontology. Finally, a constraint reasoning engine enables the final merging of the regions, while at the same time reducing the number of hypotheses. This approach is generic and applicable to any domain as long as new domain ontologies are designed and made available.

The remainder of the paper is organized as follows: after briefly studying related work in section 2, we present in section 3 an analysis of the initial requirements for the knowledge representation infrastructure both from a knowledge representation and a multimedia analysis point of view. In section 4 we present the general ontology infrastructure design focusing on the multimedia related ontologies and structures. This presentation is complemented by a description of an annotation process needed for initializing the knowledge base with prototype instances of domain concepts in question, including a description of the actual implementation of a user friendly tool to assist this annotation process. Initial results from the knowledge-assisted analysis process, which exploits the developed infrastructure and annotation framework are presented in section 5. We conclude with a summary of our work in section 6.

2 Related Work

In the *multimedia analysis* area, knowledge about multimedia content domains, as for example reported in [8], is a promising approach by which higher level semantics can be incorporated into techniques that capture the semantics through automatic parsing of multimedia content.

Such techniques are turning to knowledge management approaches, including Semantic Web technologies to solve this problem [9]. In [10], semantic entities, in the context of the MPEG-7 standard, are used for knowledge-assisted video analysis and object detection, thus allowing for semantic level indexing. In [11] a framework for learning intermediate level visual descriptions of objects organized in an ontology is presented that aid the system to detect domain objects.

In [12], a-priori knowledge representation models are used as a knowledge base that assists semantic-based classification and clustering. MPEG-7 compliant low-level descriptors are automatically mapped to appropriate intermediate-level descriptors forming a simple vocabulary termed object ontology. Additionally, an object ontology is introduced to facilitate the mapping of low-level to high-level features and allow the

definition of relationships between pieces of multimedia information. This ontology paradigm is coupled with a relevance feedback mechanism to allow for precision in retrieving the desired content.

Work in *semantic annotation* [13] has so far mainly focused on textual resources [4] or simple annotation of photographs [5, 6]. A presentation of an earlier version of *M-OntoMat-Annotizer* can be found in [14].

3 Requirements

The challenge in building a knowledge infrastructure for multimedia analysis and annotation arises from the fact that multimedia data comes in two separate though intertwined layers which need to be appropriately linked. On the one hand, *multimedia layer* deals with the semantics of properties and phenomena related to the presentation of content within the media-data itself, e.g. its spatio-temporal structure or visual features for analysis and is typically hard to understand for people who aren't trained in multimedia analysis. The *content layer*, on the other hand, deals with the semantics of the actual content contained in the media data as it is perceived by the human media consumer. This section analyzes a number of requirements for an integrated knowledge infrastructure and annotation environment for multimedia description, analysis and reasoning. To illustrate some of the requirements, we first present a simple scenario, with focus on direct exploitation:

Multimedia content manager Samantha is working on a project on historic tennis matches. She has to prepare both the metadata infrastructure and the multimedia content. Samantha loads existing general sports ontologies into M-OntoMat-Annotizer and extends them by adding missing concepts of major interest. Next, she points M-OntoMat-Annotizer to images from the project, which are loaded and depicted in the user interface. One after another, Samantha then selects different objects in the images and drags them to the corresponding concepts in the domain ontology. The system extracts visual descriptors for these concepts and stores them in the application memory. Thus, Samantha has used M-OntoMat-Annotizer to describe the tennis domain and to describe the shape and the texture of tennis balls, rackets, nets, or courts.

Note that this simple scenario has focused on simply providing conceptual information and the corresponding visual characteristics to the knowledge base which might be exploited directly in the context of the first mode described in section 1.

However, at the same time, the generated data would serve as a valuable a-priori source of information for multimedia analysis tools. These tools would use the descriptions in order to learn how to tag and relate segments of images and video keyframes with the domain ontology concepts in the next step, i.e. in the second mode sketched in section 1, initial results of which are presented in section 5.

3.1 Requirements from Multimedia Analysis

In order to support linking between low level visual information and the higher level content domain, the above example scenario implicitly requires a suitable knowledge infrastructure tailored to multimedia descriptions:

Low-Level Description Representation. In order to represent the visual characteristics associated with a concept, one has to employ several different visual properties, depending on the concept at hand. For instance, in the tennis domain as was described in the scenario, the tennis ball might be described using its shape (“round”), color (“white”), or, in some cases of video sequences, motion. Similarly, a tennis racket has a distinctive and easily recognizable shape.

Support for Multiple Visual Descriptions. Visual Characteristics of domain concepts can not be described using one single instance of the visual descriptors in question. For example, while the net of a tennis racket might be described in terms of its texture only once, its shape heavily depends on the viewing angle and occlusions (e.g. by the player in front of the net). The required conceptualization thus has to provide means for *multiple* prototypical descriptions of a domain concept.

Spatiotemporal Relation Representation. Simple visual properties may be used to model simple concepts. In some cases, however, decomposition of more complex concepts in terms of simpler object parts is desirable. A tennis player, for instance, is difficult to describe using a single shape, motion or texture description; it is more efficient to model and describe the characteristic parts (head, tennis shirt, racket) in terms of their visual properties first, and then define the human player as a spatial configuration of these parts. In other domains like beach holidays, it is more appropriate to describe the entire scene of a picture in terms of its color layout, depicting e.g. the sky at the top, the sand in the middle and the sea at the bottom. In such cases, modelling of spatiotemporal and partonomic relations is required apart from simple visual properties.

Multimedia Structure Representation. The result of the annotation (or content analysis in a next step) should be able to express the structure of a multimedia document itself, depending on the type of document, e.g. image, video, audio, or multimedia presentation. For instance, an image is usually decomposed into a number of still regions corresponding to some semantic objects of interest, while a video clip may be decomposed into shots, each of which into associated moving regions. A hierarchical structure of multimedia segments is thus needed in order to capture all possible types of spatiotemporal or media decompositions and relations.

Alignment with MPEG-7 Standard. The MPEG-7 multimedia content description standard already provides tools for representing fragments of the above information. For instance, the MPEG-7 Visual Part [15] supports color (e.g. dominant colors, color layout), texture, shape (e.g. region/contour-based), and motion (local or global) descriptors. Similarly, the MPEG-7 Multimedia Description Schemes (MDS) [16] supports spatial (directional or topological) and temporal multimedia segment relations, as well as hierarchical structures for multimedia segment decomposition. Given the importance of MPEG-7 in multimedia community, it is evident that in the design of an associated ontology, a large part of its structures should be appropriately captured, aligned and used.

Support for Basic Data Types. Finally, based on the previous requirement, and on the fact that MPEG-7 is built on XML Schema and supported by English-text semantic description but no associated data models, the implementation of an MPEG-7 ontology

using an appropriate formalism like RDF Schema would have to deal with the representation of basic data types like numeric types (integer, float etc.), dates, vectors, arrays and so on. This is a challenging task that is even more important when feature matching algorithms are employed on such data as part of the reasoning process during knowledge-assisted analysis.

3.2 Requirements from Semantic Annotation

The described infrastructure requires appropriate authoring of the domain ontologies with respect to the domain and visual descriptor ontologies.

Associate Visual Features with Concept Descriptions. Visual descriptions are made on the conceptual level, i.e. certain visual descriptors should describe how a certain domain concept is expected to look like. The ontology and annotation framework should model this link in a way that is consistent with current semantic web standards and should avoid 2nd order statements, while

- preserving the ability to use reasoning on the ontology and the knowledge base respectively and
- providing a clear distinction between the visual descriptions of a concept and its instances.

User Friendly Annotation. Domain ontologies are typically edited by trained indexers with little experience in multimedia analysis using standard ontology editing tools. Additionally, maintaining metadata about extracted low level features is cumbersome and error-prone. An annotation framework thus has to integrate:

- management of reference multimedia content (images and videos)
- extraction of suitable low level features for objects depicted in the reference content
- automatic generation of fact statements describing the correspondence between a selected concept and the low level features
- while at the same time hiding the details of these mechanisms to the user behind an easy-to-use user interface.

Modularization. The links between domain ontology concepts and low level feature descriptions should form separate modules of the overall knowledge infrastructure. Specifically, updates of these fact statements should be possible without touching the integrity of the domain ontologies.

Linking into Multimedia. Visual Descriptors contain no information about their location in the original content. This becomes a problem if existing visual descriptors need to be visualized, e.g. to check them for appropriateness or to identify redundant descriptors. Additionally, in order to be able to exploit spatial relationships between objects within multimedia content, the objects have to be linked to the respective regions, they are depicted in. This combines to the more general requirement to provide means to describe regions in terms of their location within the content, i.e. to describe their spatial features, and to link them with objects representing both concepts from the domains and visual descriptors.

Table 1. Matrix of Design Rationales

Components		Knowledge Infrastructure				M-OntoMat-Annotizer						
Requirement	Design Features	Core Ontology	Visual Descriptor Ontology	Multimedia Structure Ontology	Domain Ontologies	Prototyping Approach	Core OntoMat	Annotation Server	Domain Visual Database	Feature Extraction Toolbox	VDE Visual Editor & Media Viewer	VDE Plugin
	Ontology Extensions	•			•	•	•	•		•		
	Low-Level Description Representation		•			•				•		•
	Support for Multiple Descriptors		•			•		•				•
	Spatiotemporal Relation Representation	•		•		•		•				
	Multimedia Structure Representation			•			•	•	•			
	Alignment with MPEG-7 Standards		•	•						•		•
	Support for Basic Data Types		•					•		•		
	Associate Visual Features with Concept Descriptions					•	•	•			•	•
	User Friendly Annotation				•		•		•		•	•
Modularization				•	•		•					
Linking into Multimedia		not yet dealt with										

4 Multimedia Annotation Infrastructure Design

Based on the requirements collected in the preceding section, we propose a comprehensive Multimedia Annotation Infrastructure the components of which will be described in this section. Table 1 plots the collected requirements against the infrastructure components.

4.1 Knowledge Infrastructure Design

The requirements presented in the last section point to the challenge that the hybrid nature of multimedia data must be necessarily reflected in the ontology architecture that represents and links both layers. Fig. 1 summarizes the developed knowledge infrastructure¹.

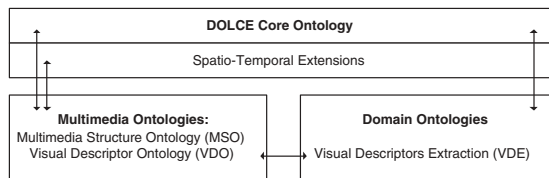


Fig. 1. Ontology Structure Overview

Knowledge Representation Formalisms. Several knowledge representation languages have been developed during the last years as ontology languages in the context of the Semantic Web, each with varying characteristics in terms of their expressiveness, ease of use and computational complexity.

Our framework uses *Resource Description Framework Schema (RDFS)* as modelling language. While RDFS offers sufficient primitives for defining domain models, other parts of the ontology infrastructure either are already encoded in OWL (like DOLCE and the spatio-temporal extensions) or are likely to be leveraged to an appropriate sub-language of OWL at a later stage. This decision also reflects the fact that a full usage of the increased expressiveness of OWL requires specialized and more advanced inference engines, especially when dealing with large numbers of instances with slot fillers, while TBox reasoning is no specific focus of this framework at this point in time.

Core Ontology. The role of the core ontology in this overall framework is to serve as a starting point for the construction of new ontologies, to provide a reference point for comparisons among different ontological approaches and to serve as a bridge between existing ontologies. In our framework, we have used DOLCE [17] for this purpose. DOLCE was explicitly designed as a core ontology, is minimal in that it includes only the most reusable and widely applicable upper-level categories, rigorous in terms of axiomatization and extensively researched and documented.

¹ We intend to make these ontologies publicly within 2005.

In a separate module, we have extended the *Region* concept branch of DOLCE to accommodate topological and directional relations between regions of different types, mainly *TimeRegion* and *2DRegion*. Directional spatial relations describe how visual segments are placed and relate to each other in 2D or 3D space (e.g., left and above). Topological spatial relations describe how the spatial boundaries of the segments relate (e.g., touches and overlaps). In a similar way, temporal segment relations are used to represent temporal relationships among segments or events.

Visual Descriptor Ontology. The Visual Descriptor Ontology (VDO) contains the representations of the MPEG-7 visual descriptors, models Concepts and Properties that describe visual characteristics of objects. By the term descriptor we mean a specific representation of a visual feature (color, shape, texture etc) that defines the syntax and the semantics of a specific aspect of the feature. For example, the *dominant color* descriptor specifies among others, the number and value of dominant colors that are present in a region of interest and the percentage of pixels that each associated color value has. Although the construction of the VDO is tightly coupled with the specification of the MPEG-7 Visual Part [18], several modifications were carried out in order to adapt to the XML Schema provided by MPEG-7 to an ontology and the data type representations available in RDF Schema.

The `VDO:VisualDescriptor` concept is the top concept of the Visual Descriptor Ontology and subsumes all modelled visual descriptors. It consists primarily of six subconcepts, one for each category that the MPEG-7 standard specifies. These are: color, shape, texture, motion, localization and basic descriptors. Each of these categories includes a number of relevant descriptors that are correspondingly defined as concepts in the VDO. The only MPEG-7 descriptor category that was modified and does not contain all the MPEG-7 descriptors is the `VDO:BasicDescriptors`.

Multimedia Structure Ontology. The Multimedia Structure Ontology (MSO) models basic multimedia entities from the MPEG-7 Multimedia Description Scheme [16] and mutual relations like decomposition. Within MPEG-7, multimedia content is classified into five types: Image, Video, Audio, Audiovisual and Multimedia. Each of these types has its own segment subclasses. MPEG-7 provides a number of tools for describing the structure of multimedia content in time and space. A number of specialized subclasses are derived from the generic Segment Description Scheme, describing the specific types of multimedia segments, such as video segments, moving regions, still regions and mosaics, which result from spatial, temporal and spatiotemporal segmentation of the different multimedia content types.

Domain Ontologies. In the multimedia annotation framework, the domain ontologies are meant to model the content layer of multimedia content with respect to specific real-world domains, such as sports events like tennis. All domain ontologies are explicitly based on or aligned to the DOLCE core ontology, and thus connected by high-level concepts, what in turn assures interoperability between different domain ontologies at a later stage.

In the context of our work, domain ontologies are created and maintained by content managers or indexers. They are defined in a way to provide a general model of the domain, with focus on the users' specific point of view. In general, the domain ontology

needs to model the domain in a way, that on the one hand the retrieval of pictures becomes more efficient for a user of a multimedia application and on the other hand the included concepts can also be automatically extracted from the multimedia layer. In other words, the concepts have to be recognizable by automatic analysis methods, but need to remain comprehensible for a human.

Prototype Approach. Describing the characteristics of *concepts* for exploitation in multimedia analysis naturally leads to a meta-concept modeling dilemma. This issue occurs in the sense that using concepts as property values is not directly possible while avoiding 2nd order modelling, i.e. staying within the scope of established standards like OWL DL².

In our framework, we propose to enrich the knowledge base with instances of domain concepts that serve as *prototypes* for these concepts. This status is modelled by having these instances also instantiate an additional VDO-EXT:Prototype concept from a separate *Visual Annotation Ontology (VDO-EXT)*. Each of these instances is then linked to the appropriate visual descriptor instances. The approach we have adopted is thus pragmatical, easily extensible and conceptually clean.

4.2 Design of M-OntoMat-Annotizer

While using and referencing the described knowledge representation infrastructure, we have extended the CREAM (CREating Metadata for the Semantic Web) framework [4] and its reference implementation, OntoMat-Annotizer³, in order to allow low-level feature annotation. Figure 2 shows the integrated architecture the modules of which are explained in the following in more detail.

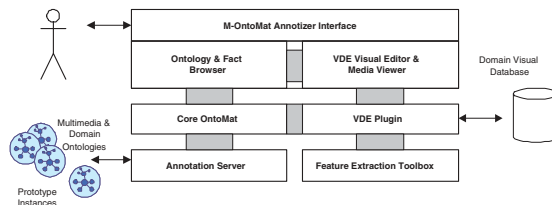


Fig. 2. M-OntoMat-Annotizer and VDE plug-in design architecture

Core OntoMat-Annotizer. OntoMat-Annotizer supports two core applications: (i) it is used as an annotation tool for web pages and (ii) it acts as the basis of an ontology engineering environment. Also, by providing a flexible plug-in interface it offers the pos-

² The issue of representing concepts as property values is under constant discussion in the Semantic Web Community. As a resource on this topic see Natasha Noy et al.: *Representing Classes As Property Values on the Semantic Web*, W3C Working Draft 21 July 2004, <http://www.w3.org/TR/2004/WD-swbp-classes-as-values-20040721/>.

Note that our approach best resembles approach 2 in this document.

³ see <http://annotation.semanticweb.org/ontomat/>

sibility to implement new components and extend the core functionality of OntoMat-Annotizer.

Annotation Server. The annotation server acts in the background and stores the entities of the knowledge base, maintains their mutual references and is responsible for maintaining the overall integrity of the stored entities.

Domain Visual Database. As easy content access is crucial for annotation and content analysis processes, a visual database containing content related to the domain examined and analyzed is always necessary. In aceMedia⁴ project, appropriate images and videos are primarily supplied by commercial partners, who actually serve as content providers.

Feature Extraction Toolbox. The actual extraction of the visual descriptors is performed using a feature extraction toolbox, namely the *aceToolbox*, a content pre-processing and feature extraction toolbox developed inside aceMedia project. The aceToolbox saves the extracted MPEG-7 Descriptors in XML format.

VDE Visual Editor and Media Viewer. The VDE Visual Editor and Media Viewer presents a graphical interface for loading and processing of visual content (images and videos), visual feature extraction and linking with domain ontology concepts. The interface, as shown in Figure 3, seamlessly integrates with the common OntoMat interfaces. Usually, the user needs to extract the features (multimedia descriptors) of a specific object inside the image/frame. For this reason, the VDE application lets the user draw a region of interest in the image/frame and apply the multimedia descriptors extraction procedure only to the specific selected region. By selecting a specific concept in the OntoMat ontology browser and selecting a region of interest the user can extract and link concepts with appropriate prototype instances by means of the underlying functionalities of the VDE plugin.

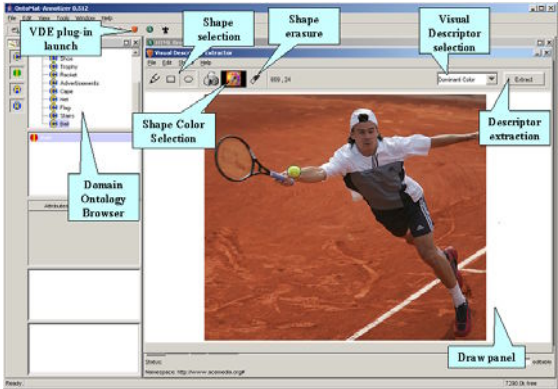


Fig. 3. The VDE plugin into M-OntoMat-Annotizer user interface

⁴ see <http://www.acemedia.org/>

VDE Plug-in. The *Visual Descriptor Extraction* (VDE) tool is implemented as a plug-in to OntoMat-Annotizer and is the core component for extending its capabilities and supporting the initialization of ontologies with low-level multimedia features. The VDE plugin manages the overall low-level feature extraction and linking process by communicating with the other components.

4.3 VDE Extraction and Annotation Process

Triggered by the users extraction command, the VDE plugin *extracts* the requested MPEG-7 Descriptors through calls to the Feature Extraction Toolbox. The VDE plug-in supports the *transformation* of the extracted XML multimedia resources into instances of the visual descriptors defined in the VDO, by means of an XSL transformation specification that creates a corresponding descriptor instance for each extraction, which is handed to the knowledge base on the annotation server:

```
<!-- Values of "ScalableColorDescriptor" instance "VDE_INST_0001"-->
<vdo:ScalableColorDescriptor
  rdf:ID="http://www.acemedia.org/ontologies/VDO#VDE_INST_0001">
  <vdo:coefficients> 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 </vdo:coefficients>
  <vdo:numberOfBitPlanesDiscarded> 6 </vdo:numberOfBitPlanesDiscarded>
  <vdo:numberOfCoefficients> 0 </vdo:numberOfCoefficients>
</vdo:ScalableColorDescriptor>
```

Then, the VDE automatically links the newly created visual descriptor instance with the selected domain concept prototype instance on the annotation server:

```
<!-- A prototype instance "Ball_Prototype_1" definition of tennis
concept "Ball". "Ball_Prototype_1" instance "hasDescriptor"
"VDE_INST_0001", which corresponds to the scalable color descriptor
instance above -->
<vdoext:Prototype
  rdf:ID="http://www.acamedia.org/ontologies/VDO-EXT#Ball_Prototype_1">
  <rdf:type rdf:resource="http://www.acamedia.org/ontologies/TENNIS#Ball"/>
  <vdoext:hasDescriptor
    rdf:resource="http://www.acamedia.org/ontologies/VDO#VDE_INST_0001"/>
</vdoext:Prototype>
```

As seen above, these prototype instances are not only instances of the domain concept in question but are also stated to be instances of a separate Prototype concept. The created statements are added to the knowledge base and can be retrieved in a flexible way during analysis. The necessary conceptualizations can be seen as extensions to the VDO (*VDO-EXT ontology*) that link to the core ontology and are implemented in RDF:

```
<!-- Definition of concept "Prototype" as a subclass of Dolce's
"Physical-Object" -->
<rdfs:Class
  rdf:ID="http://www.acemedia.org/ontologies/VDO-EXT#Prototype">
  <rdfs:subClassOf rdf:resource=
    "http://ontology.ip.rm.cnr.it/ontologies/DOLCE-Lite#Physical-Object"/>
</rdfs:Class>

<!-- Definition of relation "hasDescriptor" having Dolce's
"Physical-Object" as domain and VDO's "VisualDescriptor" as range-->
<rdf:Property
  rdf:ID="http://www.acemedia.org/ontologies/VDO-EXT#hasDescriptor">
  <rdfs:domain rdf:resource=
    "http://ontology.ip.rm.cnr.it/ontologies/DOLCE-Lite#Physical-Object"/>
  <rdfs:range
    rdf:resource="http://www.acemedia.org/ontologies/VDO#VisualDescriptor"/>
</rdf:Property>
```

All the prototype instances can be saved in a RDFS file. The VDE tool saves the domain concept prototype instances together with the corresponding transformed descriptors, *separately* from the ontology file, thus leaving the original domain ontology unmodified.

5 Knowledge-Assisted Analysis

The extracted knowledge base presented above, is playing a central role in automatic semantic multimedia analysis process, through tools currently being developed in ace-Media that automatically analyze content, generate metadata and annotation, and support intelligent content search and retrieval services. Currently, three spatial relations and three low-level descriptors are supported. These include the adjacency (*ADJ*), below (*BEW*), and inclusion (*INC*) relations, and the dominant color (*DC*), motion (*MOV*) and compactness (*CPS*) descriptors.

During preprocessing, color segmentation and motion segmentation are combined to generate a set of over-segmented atom-regions. After preprocessing, assuming for a single image N_R atom regions and a domain ontology of N_O objects, there are $N_R^{N_O}$ possible scene interpretations. A genetic algorithm is used to overcome the computational time constraints of testing all possible configurations [19].

The degree of matching between regions, in terms of low-level visual and spatial features respectively, is defined in an interpretation function used for the genetic algorithm fitness function and is based on a back-propagation neural network. When the task is to compare two regions based on a single descriptor, several distance functions can be used; however, there is not a single one to include all descriptors with different weight on each. This is a problem that is handled by the neural network. Its input consists of the low-level descriptions of both an atom region and an object model, while its

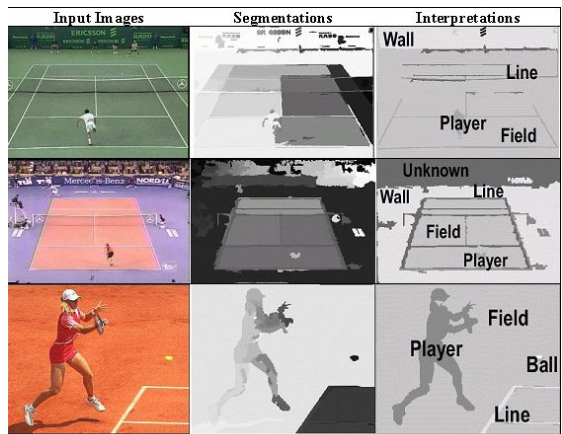


Fig. 4. Tennis domain results

response is the estimated normalized distance between the atom region and the model. A training set is constructed using the descriptors of a set of manually labelled atom regions and the descriptors of the corresponding object models. Figure 4 illustrates example results from the sports domain, where the system output is a segmentation mask outlining the semantic description of the scene.

6 Summary

In this paper, an integrated infrastructure for semantic multimedia content annotation was presented. This framework comprises ontologies for the description of low-level audio-visual features and for linking these descriptions to concepts in domain ontologies based on a prototype approach. This prototype approach avoids the well-known problems introduced by Meta-Concept Modelling, and thus preserves the ability to use OWL DL compliant reasoning techniques on the annotation meta-data.

The generation of the visual descriptors and the linking with the domain concepts is embedded in an user-friendly tool, which hides analysis-specific details from the user. Thus, the definition of appropriate visual descriptors can be accomplished by domain experts, without the need to have a deeper understanding of ontologies or low-level multimedia representations. In allowing annotation and linking of concept prototype instances with more than one extracted descriptors, the system is flexible with respect to analysis requirements. In allowing multiple prototypical instantiations of the a concept, the system is flexible with respect to varying visual characteristics of objects.

An important issue in the actual annotation procedure, is the selection of appropriate descriptors for extraction, valuable for the further analysis process. Depending on the results, the knowledge-assisted analysis process adjusts its needs and guides the extraction procedure, providing constant feedback on the concepts that have to be populated, how many prototype instances are necessary for each concept, which descriptors are helpful for the analysis of a specific concept etc.

Finally, despite the early stage of multimedia analysis experiments, first results based on the ontologies presented in this work are promising and show that it is possible to apply the same analysis algorithms to process different kinds of images or video, by simply employing different domain ontologies. Apart from visual descriptions and relations, future focus will concentrate on the creation of rules to assist reasoning in order to detect more complex events. The examination of the interactive process between ontology evolution and use of ontologies for content analysis will be the target of our future work, in the direction of handling the semantic gap in multimedia content interpretation.

Acknowledgements. This research was partially supported by the European Commission under contract FP6-001765 aceMedia. The expressed content is the view of the authors but not necessarily the view of the aceMedia project as a whole.

References

1. S.-F. Chang, T. Sikora, and A. Puri. Overview of the MPEG-7 standard. *IEEE Trans. on Circuits and Systems for Video Technology*, 11(6):688–695, June 2001.
2. O. Mich R. Brunelli and C.M. Modena. A survey on video indexing. *Journal of Visual Communications and Image Representation*, 10:78–112, 1999.
3. A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12).
4. Siegfried Handschuh and Steffen Staab. Cream - creating metadata for the semantic web. *Computer Networks*, 42:579–598, AUG 2003. Elsevier.
5. J. Wielemaker A.Th. Schreiber, B. Dubbeldam and B.J. Wielinga. Ontology-based photo annotation. *IEEE Intelligent Systems*, May/June 2001.
6. L. Hollink, A.Th. Schreiber, J. Wielemaker, and B. Wielinga. Semantic annotation of image collections. In *Proceedings of the K-CAP 2003 Workshop on Knowledge Markup and Semantic Annotation, Florida*, 2003.
7. P. Wittenburg D. Thierry and H. Cunningham. The Automatic Generation of Formal Annotations in a Multimedia Indexing and Searching Environment. In *Proc. ACL/EACL Workshop on Human Language Technology and Knowledge Management*, Toulouse, France, 2001.
8. J. Hunter, J. Drennan, and S. Little. Realizing the hydrogen economy through semantic web technologies. *IEEE Intelligent Systems Journal - Special Issue on eScience*, 19:40–47, 2004.
9. A. Yoshitaka, S. Kishida, M. Hirakawa, and T. Ichikawa. Knowledge-assisted content-based retrieval for multimedia databases. *IEEE Multimedia*, 1(4):12–21, Winter 1994.
10. R. Tansley, C. Bird, W. Hall, P. Lewis, and M. Weal. Automating the linking of content and concept. In *Proc. ACM Int. Multimedia Conf. and Exhibition (ACM MM-2000)*, Oct./Nov. 2000.
11. Nicolas Maillot, Monique Thonnat, and Céline Hudelot. Ontology based object learning and recognition: Application to image retrieval. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 15-17 November 2004, Boca Raton, FL, USA, pages 620–625, 2004.
12. I. Kompatsiaris, V. Mezaris, and M. G. Strintzis. *Multimedia content indexing and retrieval using an object ontology*. Multimedia Content and Semantic Web - Methods, Standards and Tools, Editor G.Stamou, Wiley, New York, NY, 2004.
13. Siegfried Handschuh and Steffen Staab, editors. *Annotation for the Semantic Web*. IOS Press, 2003.
14. Stephan Bloehdorn, Steffen Staab, Siegfried Handschuh, Yannis Avrithis, Vasilis Tzouvaras, Nikos Simou, Michael G. Strintzis, Yiannis Kompatsiaris, and Kosmas Petridis. Knowledge representation for semantic multimedia content analysis and reasoning. In *Proceedings of the European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology (EWIMT)*, NOV 2004.
15. T. Sikora. The MPEG-7 Visual standard for content description - an overview. *IEEE Trans. on Circuits and Systems for Video Technology, special issue on MPEG-7*, 11(6):696–702, June 2001.
16. ISO/IEC 15938-5 FCD Information Technology - Multimedia Content Description Interface - Part 5: Multimedia Description Scemes, March 2001, Singapore.

17. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, Proceedings of the 13th International Conference on Knowledge Acquisition, Modeling and Management, EKAW 2002*, volume 2473 of *Lecture Notes in Computer Science*, Siguenza, Spain, 2002.
18. ISO/IEC 15938-3 FCD Information Technology - Multimedia Content Description Interface - Part 3: Visual, March 2001, Singapore.
19. N. Voisine, S. Dasiopoulou, V. Mezaris, E. Spyrou, T. Athanasiadis, I. Kompatsiaris, Y. Avrithis, and M.G. Strintzis. Knowledge-Assisted Video Analysis Using A Genetic Algorithm. In *Proc. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2005)*, April 2005.

RELFIN - Topic Discovery for Ontology Enhancement and Annotation^{*}

Markus Schaal, Roland M. Müller, Marko Brunzel, and Myra Spiliopoulou

Otto-von-Guericke-University Magdeburg
forename.name@iti.cs.uni-magdeburg.de

Abstract. While classic information retrieval methods return whole documents as a result of a query, many information demands would be better satisfied by fine-grain access inside the documents. One way to support this goal is to make the semantics of small document regions explicit, e.g. as XML labels, so that query engines can exploit them. To this purpose, the *topics* of the small document regions must be discovered from the texts; differently from document labelling applications, fine-grain topics cannot be listed in advance for arbitrary collections. Text-understanding approaches can derive the topic of a document region but are less appropriate for the construction of a *small set of topics* that can be used in queries.

To address this challenge we propose the coupling of text mining, prior knowledge explicated in ontologies and human expertise and present the system RELFIN, which is designed to assist the human expert in the *discovery of topics* appropriate for (i) ontology enhancement with additional concepts or relationships, (ii) semantic characterization and tagging of document regions. RELFIN performs *data mining* upon linguistically preprocessed corpora to group document regions on *topics* and *constructing the topic labels* for them, so that the labels are characteristic of the regions and thus helpful in ontology-based search. We show our first results of applying RELFIN on a case study of text analysis and retrieval.

Keywords: Topic Discovery, Label Construction, Ontology Enhancement, Text Clustering.

1 Introduction

Ontologies over document corpora and semantic labels inside the documents can greatly enhance information acquisition: Ontologies describe concepts and the relationships among them and map them into their textual representations in the documents. A semantic label reflects the topic of a small part of a document, e.g. a paragraph or a sentence; if it is implemented as an annotation tag, it can

^{*} Work partially funded under the EU Contract IST-2001-39023 Parmenides.
<http://www.crim.co.umist.ac.uk/parmenides>

be exploited by a query engine. The topics corresponding to the semantic labels may or may not consist of terms from the ontology, so the two instruments support information acquisition in complementing ways.

In this paper, we present the interactive system RELFIN for the discovery of region-level topics in documents. RELFIN is part of the PARMENIDES integrated environment, which encompasses tools for linguistic pre-processing, ontology enhancement through concepts and relations, semantic text annotation and extraction of entities and events. RELFIN uses data mining techniques to analyse and group semantically similar document regions and to derive labels as topics from them. At the same time, RELFIN interacts with the human expert who provides the context knowledge and assists her by proposing topics for text annotation or ontology enhancement.

Central to RELFIN is the notion of *topic cluster*. A topic cluster is a non-marginal set of similar document regions, where similarity is given by the cosine-distance between vectors. Each document region is represented by a vector over a feature space of concepts from an ontology. The weights are computed by the TF-IDF value of the term count for a feature. A cluster label is constructed as a concatenation of features with a high support within the cluster - the so-called *frequent features*. We term the generation of topic clusters as *topic discovery*. Topic discovery can be used both for ontology enhancement and text annotation but it must be stressed, that these tasks are distinct. We propose the use of additional corpus specific terms for ontology enhancement and a novel quality criterium for region annotation.

For *ontology enhancement*, new terms are proposed as new concepts, while groups of co-occurring terms (concepts) are indicative of the need to create links among them. Thus, the human expert is supported in creating added value by juxtaposing her background knowledge to corpus content and exploiting both to enrich the ontology. It should be noted that even the richest ontology may need this type of enrichment: A corpus focusses on specific aspects of a universe of discourse, which may or may not be explicit in the ontology. Moreover, the individual document regions may refer to topics that turn to be a posteriori of importance for the ontology. However, topics at region level may be too fine-grain for ontology enhancement but nonetheless appropriate for search inside the specific corpus. Hence, RELFIN supports *text annotation*, i.e. the tagging of document regions with the derived topic labels, as complementary task to ontology enhancement.

A major challenge for topic clustering is the specification of the label. Obviously, a label consisting of the single term “be” is not informative for most corpora. A label consisting of 100 terms appearing in some of the cluster members is not very useful towards a search engine either. For RELFIN, we propose a novel criterium for cluster labelling - the *Residuum* of non-frequent features within the topic cluster: First, RELFIN clusters document regions on similarity and derives labels for the clusters. Then, a residuum threshold is set and clusters not satisfying the threshold are rejected. The retained topic clusters with their labels are proposed to the domain expert as *labelled topic clusters*.

In the next section, we discuss related work. In section 3 we introduce the PARMENIDES framework, in which RELFIN operates and then elaborate on RELFIN in section 4. Section 5 contains a first set of experiments for interactive ontology enhancement and text annotation and a discussion of the findings. The last section concludes the study.

2 Related Work

There is increasing research on the discovery of semantic labels from text data. Some methods put their emphasis on the formulation of appropriate labels [MB01, RM99, GSW01, WS01b], while others further consider the establishment of schemata or other semantic descriptions from those labels [HSS03, KVM00, MS00a, MS00b, WS01a, WS02]. We elaborate on these two types of methods in the following. Moore and Berman propose an algorithm that converts textual pathology reports into XML documents: Natural Language Processing (NLP) techniques are applied upon the texts; the identified terms and noun groups are mapped upon concepts of a medical thesaurus; these concepts become XML tags that annotate the corresponding pieces of text [MB01]. This approach can achieve an extensive annotation of the corpus at a great level of detail.

Rauber and Merkl derive document labels by a clustering technique [RM99]: The documents are modelled as vectors of weighted terms and clustered on similarity. For each cluster thus established, a label is derived by considering the terms characterizing the cluster. The core methodology is conceptually the same as in our previous work on the derivation of labels for sentences by similarity-based clustering of sentence contents [GSW01, WS01b].

Handschuh et. al. [HSV03] presents a system that learns information extraction rules from manually tagged input. In contrast to our approach they focus on entities extraction and not on topic discovery and they need pre-labelled documents that we don't need.

The subject of deriving an appropriate semantic label for a set of similar texts is also addressed in [HSS03]: Hotho et al use text clustering to derive text clusters. However they subsequently use formal concept analysis to construct a concept lattice and don't use metrics to check the validity of a cluster for topic enhancement.

The extraction of a domain-specific ontology from texts with data mining techniques is discussed in [KVM00, MS00a, MS00b], whereby [KVM00, MS00b] concentrate on the core mechanism, which relies on the frequency of concepts in the texts, while the emphasis in [MS00a] is on the discovery of semantic relations by using association rules. The semantic richness and diversity of corpora does not lend itself to full automation, so that the involvement of a domain expert becomes necessary [MS00c].

The ASIUM system [FN99] uses unsupervised concept clustering methods to learn semi-automatically subcategorization frames of verbs and ontologies. However they haven't used text unit clustering and don't use cluster quality criteria.

In this study, we extend our previous work on the “DIAsDEM Workbench” for the formulation of semantic labels for text fragments [GSW01, WS02]. Similarly to the original DIAsDEM Workbench, we perform clustering over the document corpus to establish a set of clusters, for which semantic labels can be derived. However, we replace the original rudimentary criteria on cluster cardinality and number of representative features in a cluster with a more sophisticated measure of cluster quality, the so-called *residuum* of non-frequent features, thereby enabling the automatic selection and labelling of high-quality clusters.

3 Parmenides Framework

PARMENIDES is an EU-funded project in the area of knowledge extraction and management. One of its goals is the realization of an ontology-driven systematic approach for integrating the entire process of information gathering, processing and analysis (cf. [SRB⁺04]).

One task within this goal is the extraction of knowledge from texts. Knowledge extraction is directed towards (a) the establishment of ontologies which reflect the universe of discourse and (b) the semantic annotation of documents with the concepts, entities and events depicted in the ontologies.

The RELFIN module is responsible for ontology enhancement with new concepts and with concept groups, as well as the semantic annotation of texts with such concepts/groups. As explained in the introduction, we use the collective term “topic” for them. This knowledge extraction process involves at least one human expert, who aligns knowledge extraction to the business objectives by:

- providing an initial ontology
- enhancing the ontology with concepts and relations found by the “RELFIN Learner”
- reviewing the topic clusters and proposed labels for the annotation to be performed by the “RELFIN Annotator”

The software components RELFIN Learner and RELFIN Annotator can be seamlessly integrated into “PARMENIDES workflows”: A workflow is a series of component invocations that can be specified graphically and then executed on the fly as shown in Fig. 1. The components process documents and enrich them with annotations at different levels of complexity and semantic [RDH⁺03a], i.e. by representing linguistic as well as conceptual knowledge. The XML-based ParDoc format [RDH⁺03b] is used as reference format. The components interact with each other via a document queue, depicted in the figure under the label `NormalQueue`.

The example workflow of Fig. 1 consists of the following components¹ (named by their labels):

¹ Printed with the kind agreement of the responsible PARMENIDES partners

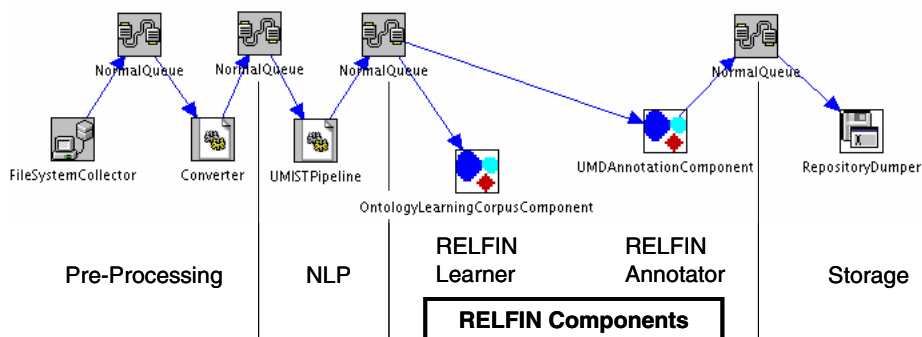


Fig. 1. An example PARMENIDES workflow (screenshot + annotation)

FileSystemCollector: This component collects documents from a repository on a hard disk. For collecting documents from the web, the component **WebSystemCollector** should be used instead.

Converter: This component is responsible for creating **ParDoc** documents from other document types, such as HTML, plain text, pdf, word and ppt.

UMISTPipeline: This component is a four-step analysis process that performs basic NLP and information extraction functionality. Its functionality consists of Tokenization, POS Tagging, Sentence Splitting, Ontology Lookup operations and ontology-based Information Extraction using the Cafetiere software ².

OntologyLearningCorpusComponent: This component is the RELFIN Learner component for text clustering and interactive expert involvement, as described in section 4. It takes as input a whole collection of documents (a seed collection) and outputs clusterings, expert-approved topic labels and the documents with annotations.

UMDAnnotationComponent: This component is the RELFIN Annotator component. It reads as input a document, as well as the clusterings and the approved cluster labels output by the RELFIN Learner. It assigns the regions of the document into clusters and annotates them with the corresponding topic labels.

RepositoryDumper: This component stores **ParDoc** documents into a Document Repository.

4 The RELFIN Learner

In Section 4.1, we give a formal introduction to the concepts used with the RELFIN Learner including the novel cluster quality criterium *Residuum*. We

² For a documentation on the UMIST Pipeline, cf. Vasilakopoulos et al. [VBB04]

show the procedure of the RELFIN Learner in Section 4.2 and present some details about the human expert interaction with the RELFIN GUI in Section 4.3.

4.1 Formal Concepts

A *text unit* is an arbitrary text fragment produced by a linguistic tool, e.g. by a sentence-splitter. Text units consist of words. A *term* is a stemmed or lemmatized word. Thus, text units can be measured according to the frequency of the terms contained in it. Text units correspond to documents or document regions, e.g. paragraphs or sentences. A *text corpus* $\mathcal{A} = \{1, \dots, n\}$ is a set of text units.

A term is a textual representation of a *concept*. Generally, there is a m-to-n mapping between terms in a text corpus and a set of concepts that describe a universe of discourse due to synonyms and homonyms. Terms, concepts and the mapping between them are part of the *ontology* of this universe.

A *feature space* $\mathcal{F} = (1, \dots, d)$ is a sequence of features, where each feature corresponds to a single concept. A *vectorization* $\mathcal{X}$ of the text corpus is obtained by counting all terms that are mapped to each of the features in the feature space for all text units of a text corpus. Subsequently, TF-IDF weighting is applied³. $\mathcal{X}$ is a 2-dimensional matrix given by values x_{ij} per text unit $1 \leq i \leq n$ and feature $1 \leq j \leq d$. Thus, each text unit i is represented by its vector $x_i = (x_{i1}, \dots, x_{id})$ over the feature space.

A *cluster* $C \subseteq \mathcal{X}$ is a set of vectors. A *cluster label* is a term or term combination that is given by the *frequent features* of a cluster. A *frequent feature* is a feature whose in-cluster support is above a certain threshold τ_{ics} .

Definition 1 (In-Cluster Support of a Feature). Let $C \subseteq \mathcal{X}$ be a cluster, where $\mathcal{X}$ is the vector space over the text corpus $\mathcal{A}$ for the feature space $\mathcal{F}$. Let $k \in \mathcal{F}$ denote a feature. The in-cluster support of feature k in C is the count of vectors $x \in C$ that contain feature k (i.e. $x_k \neq 0$) divided by the cardinality of C .

$$\text{ics}(k, C) = \frac{|\{x \in C \mid x_k \neq 0\}|}{|C|} \quad (1)$$

One criteria for clusters having a good label is newly introduced here, the so-called *Residue* of the in-cluster support of infrequent labels. *Topic Clusters* with a residue lower than a certain residue threshold τ_{res} are *pure*.

Definition 2 (Residue). Let $C \subseteq \mathcal{X}$ be a cluster and let τ_{ics} be the lower boundary to the in-cluster support of features, thus determining which features are frequent. Then, the residue of C subject to this threshold is the relative in-cluster support for infrequent features:

$$\text{residue}(C, \tau_{\text{ics}}) = \frac{\sum_{k \in \text{nonfreq}(C, \tau_{\text{ics}})} \text{ics}(k, C)}{\sum_{k \in \mathcal{F}} \text{ics}(k, C)} \quad (2)$$

where $\text{nonfreq}(C, \tau_{\text{ics}}) = \{k \in \mathcal{F} \mid \text{ics}(k, C) \leq \tau_{\text{ics}}\}$.

³ For a documentation on Vectorization and TF-IDF weighting, cf. Salton and Buckley [SB88].

4.2 Procedure

The RELFIN procedure is described by a Data Flow Diagram (DFD) in Fig. 2. *Processes* are represented by circles, *external input* is represented by squares and *data stores* are represented by open boxes (over- and underlined). The software proceeds as follows (cf. Fig. 2):

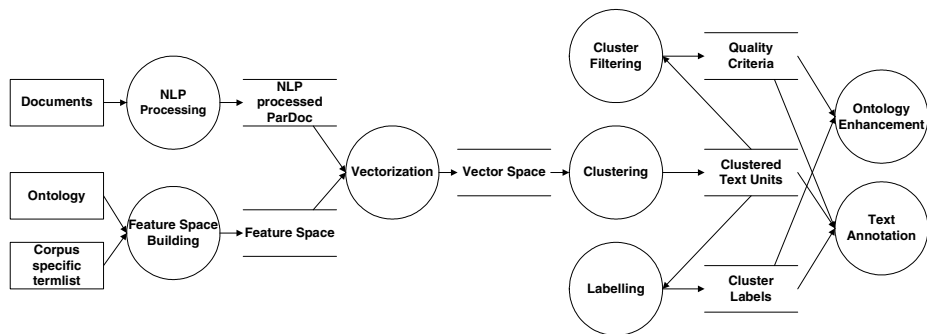


Fig. 2. Data Flow Diagram of RELFIN Learner

NLP Processing. The RELFIN application relies on a NLP processed document collection. These documents are provided in the ParDoc format.

Feature Space Building. The feature space is build from an ontology and/or a list of corpus specific terms⁴. When using the ontology⁵, each class including its synonyms and its instances⁶ (and their synonyms) becomes a feature of the feature space.

Vectorization. Different text granularities are available from the ParDoc format for vectorization: (1) documents as a whole, (2) paragraphs or (3) sentences. Here we use sentences as the chosen granularity. Each text unit is represented by its vector computed from the feature space. Only text units with two or more non-zero values are used for further processing.

Clustering. The text units are clustered by use of a Bi-Secting k-Means algorithm [SKK00], which partitions the instances in k clusters. The parameter k is specified by the user. The Bi-Secting k-Means is a variation of the k-means and showed great success in the text clustering problem[SKK00]. The algorithm starts with a single cluster which is split into two clusters by a k-means algorithm with $k = 2$. Then, the biggest cluster is choosen and it is again split in two clusters. This is done until the desired cluster number

⁴ Corpus specific terms are ordered by their rank position ratio with respect to a general language corpus, here the British National Corpus.

⁵ The PARMENIDES project incorporates an ontology editor for the maintenance of ontologies.

⁶ In the used ontology format, instances are maintained as special concepts to be subsumed together with their class concepts.

is reached. Alternatively, the cluster with highest residuum (cf. paragraph *Cluster Filtering* below) is chosen instead of the biggest cluster. The cosine metric is used as the distance function.

Cluster Filtering. After the clustering, the quality measures of the clusters are calculated. For a cluster to be considered as a *labelled topic cluster* (and thus be accepted), we require the cluster

- to be *non-marginal*, i.e. to have a cardinality above a certain threshold $c_{\min}$ and
- to be *pure*, i.e. to have a residuum lower than a given threshold τ_{res} (with respect to frequent feature threshold τ_{ics} , cf. Section 4.1).

The label of a *pure* and *non-marginal* cluster is given by the set of its frequent features, i.e. features with high in-cluster support, whereas there are only few instances not covered by the frequent features.

Labelling. For each cluster, the set of frequent features is concatenated and proposed as the cluster label.

Ontology Enhancement. RELFIN can be used for ontology enhancement, cf. Section 4.3.

Text Annotation. RELFIN can be used for semi-automatic annotation, cf. Section 4.3.

4.3 Human Expert Interaction

Figure 3 shows the RELFIN GUI, displaying a table of clusters on the left side and details of the selected cluster on the right side. The table allows sorting the clusters on certain attributes, associated with the clusters. The bar chart diagram shows the percentage of instances which have a certain feature, for the ten most frequent features of the selected cluster. In the lower right corner examples of text units in the current cluster are displayed, whereas terms, included in the feature space are highlighted. These example text units help the user to justify the appropriateness of a cluster label.

For ontology enhancement, an ontology editor is opened on the desktop next to the RELFIN-GUI, where the domain expert can edit an existing ontology of his choice. Good candidates for enhancement are homogeneous⁷ clusters with respect to the cosine distance, it is on experts choice whether to include a certain feature - as a new (1) class (concept) or (2) instance (concept), as a (3) synonym of an existing concept or as a (4) attribute type/ attribute value.

For annotation, a domain expert is required to browse (at least) the accepted clusters in order to deny acceptance by deselecting the check mark next to **accept Cluster** and to optionally edit the **Cluster Label** in the cluster information section of the RELFIN GUI, cf. upper-right corner of Fig. 3.

⁷ The term *homogeneous* refers to the cluster criteria *average instance to centroid distance (AICD)* and *average instance-to-instance distance (AIID)*, which are based on the cosine distance used for clustering and also shown in the cluster table.

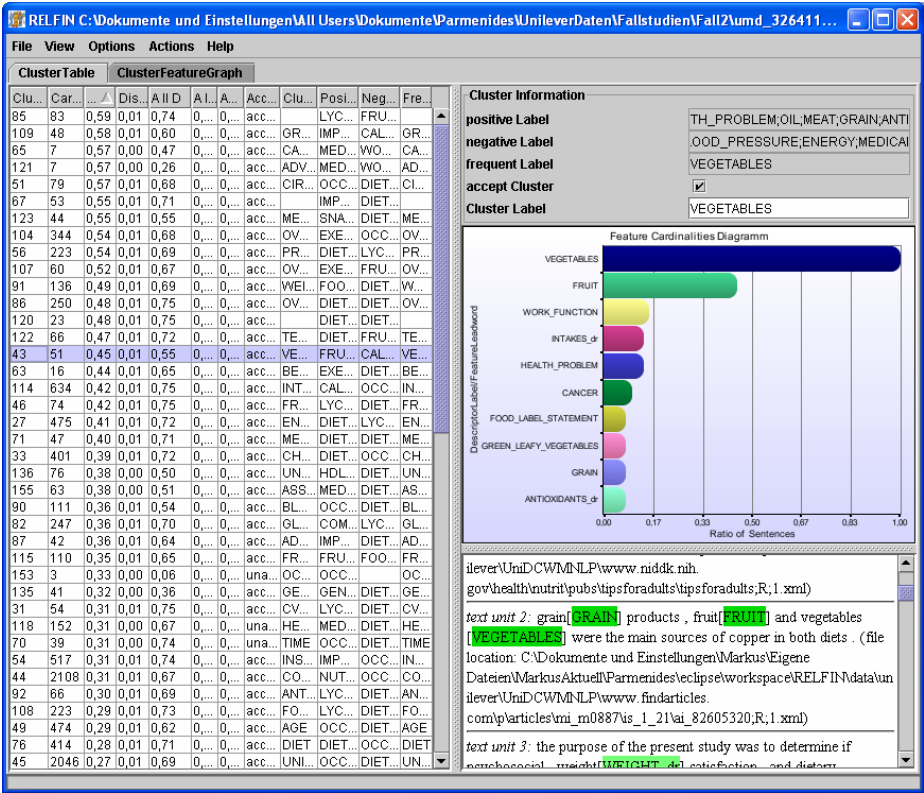


Fig. 3. Presentation of the clustering results in RELFIN

5 Experiments

For our experiments, we employed a text corpus collected from the internet for the weight management domain. This text corpus and an initial ontology were provided by an Unilever domain expert. An objective of the weight management case study of Unilever is the improvement of information retrieval and decision support. The corpus contains 1394 documents and has been split into more than 20.000 sentence-level text units. For the experiments, not all but only text units with a certain feature support were used, cf. Section 4. We performed two investigations on the text corpus:

1. Topic Clusters generated by clustering and their frequent features were presented to a human expert. The expert was asked to report on the benefits for ontology enhancement.
2. Labelled Topic Clusters were automatically accepted according to their size and residuum. We compared different clustering parameters with respect to their coverage of the text corpus.

The first investigation examined the usability of text clustering for *ontology enhancement*, the second investigation examined the ability of text clustering to find *labelled topic clusters*. Note, that the underlying corpus does not have a gold-standard ontology or annotation.

5.1 Ontology Enhancement

For the purpose of human expert ontology enhancement, 80 clusters were generated. For building the feature space, the features from the ontology were complemented with terms from the corpus-specific term list, so that 500 features were used altogether. The ic-support threshold for frequent features was set to $\tau_{cs} = 0.2$ and all clusters were presented to the human expert. By creating 80 clusters only, we got good topic clusters for ontology enhancement, but the topic clusters weren't pure enough for annotation.

In order to evaluate the use of proposed clusters for ontology enhancement, the Unilever domain expert was asked to evaluate the topic clusters according to the following criteria:

- Do the given term or term combinations (the frequent features) make sense and is it of relevance in the use case? Please indicate by Accepting/ Rejecting each cluster.
- Are some of the delivered terms or term combinations appropriate for ontology enhancement? A term or term combination is appropriate if you would decide to put it in the ontology. A combination can be put into the ontology as a combined concept or by establishing a link between concepts.

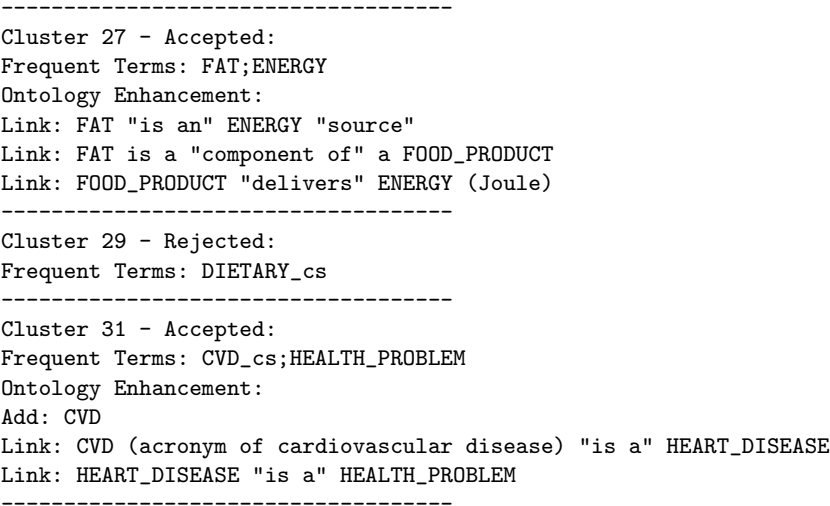


Fig. 4. Expert Contribution for Ontology Enhancement (for 3 sample clusters)

Note that the review of unlabelled topic clusters is not of use with such a low ic-support threshold. Moreover, features below the threshold are of no interest. Therefore, the expert was provided with a report on the frequent labels per cluster only, without asking him to browse the cluster table.

The whole clustering was evaluated by the Unilever domain expert according to the above criteria, a sample of the results is shown in Fig. 4. Frequent features with suffix ”_cs” are the ones originating from the corpus specific term list.

Out of 79 clusters, the human expert accepted 30 clusters. Based on the frequent feature combinations of the accepted clusters, he proposed 21 new concepts, 14 new synonyms and 10 new links between concepts for ontology enhancement.

5.2 Labelled Topic Clusters

By our approach of filtering topic clusters for purity by setting a threshold on the residuum, we have deliberately surrendered a full coverage of all text units. Here we study the coverage of text units by labelled topic clusters for different splitting criteria, residuum thresholds, cluster counts and different feature spaces.

In a first experiment, only the initial ontology was used for building the feature space, resulting in 12990 text units to be selected for clustering. Fig. 5 shows the text units covered by topic clusters over the amount of generated clusters. Two different splitting criteria for the Bi-Secting k-Means algorithm have been used, namely *Splitting the Cluster with Highest Cardinality* (Card-Split) and *Splitting the Cluster with the Highest Residuum* (Res-Split). Clusters have been accepted as topic clusters with residuum threshold $\tau_{\text{Res}} = 0.5$ (threshold=0.5) and $\tau_{\text{Res}} = 0.3$ (threshold=0.3) respectively. In all cases, the minimum cardinal-

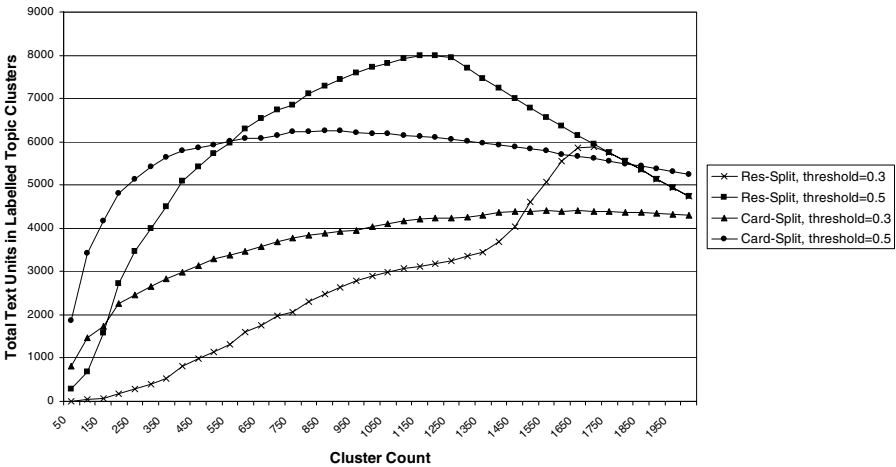


Fig. 5. Text Units in Topic Clusters (total 12990)

ity threshold was $c_{\min} = 15$ and the ic-support threshold for frequent features was $\tau_{\text{ics}} = 0.8$.

For the case of *Splitting the Cluster with the Highest Residuum* (Res-Split), the maximum is reached quite late, i.e. 8.000 text units at 1.150 clusters (147 topic clusters) for the case of $\tau_{\text{res}} = 0.5$ and 5.880 text units at 1700 clusters (133 topic clusters) for the case of $\tau_{\text{res}} = 0.3$.

For the case of *Splitting the Cluster with Highest Cardinality* (Card-Split), the maximum is considerably lower (most likely due to the creation of marginal clusters), but better residue can be reached with less clusters.

In a second experiment, we compared the use of an initial ontology with the use of corpus specific terms. Fig. 6 shows the result for using 300 terms of an ontology (Ontology) juxtaposed against using the first 300 corpus specific and using both (600 words). All computations have been performed with *Res-Split* and residuum threshold $\tau_{\text{res}} = 0.5$ (threshold=0.5). Note that the total size of text units for the clustering varies, since different feature spaces are build and only vectors with two or more non-zero values are accepted.

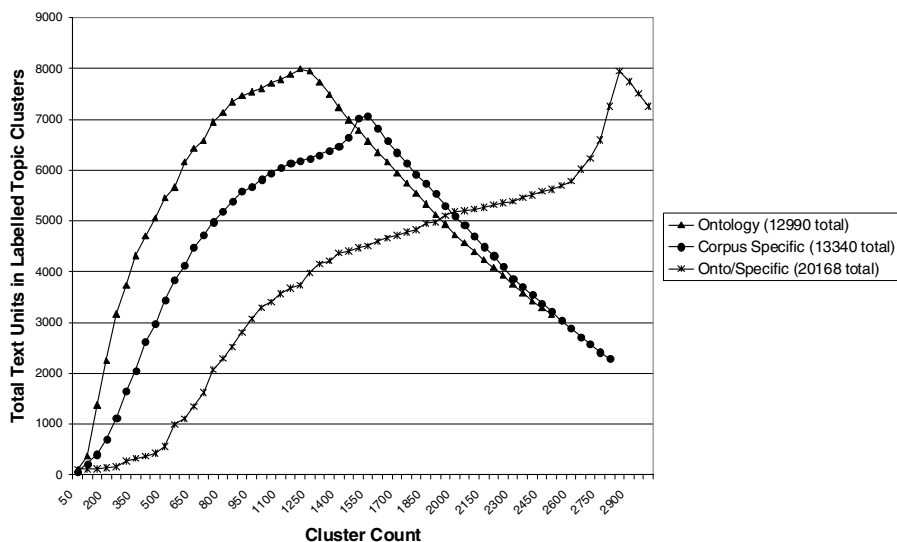


Fig. 6. Text Units in Topic Clusters (Res-Split, threshold=0.5)

For the case of the ontology, the maximum is higher than with the corpus specific term-list, while the maximum of the combination of both is reached later. It should be noted, that the usage of 600 words in the combined case allowed for a total of 20.168 text units to be represented by a vector with two or more non-zero values. The word-list's weakness when compared with the ontology (while actually processing more text units, 13.340 vs. 12.990) might be caused by the ontology being properly tailored towards the text corpus by the domain expert.

6 Conclusions

We presented the fully-fledged RELFIN application as an integrated component within the PARMENIDES framework and showed its ability to support semi-automatic ontology enhancement and a novel approach of filtering for pure topic clusters.

We applied the RELFIN methodology to a real use case without pre-existing gold standards for ontologies or text annotation and learned from first experiments:

- The stimulation of the human expert by looking at topic clusters is manifold and leads to added value by knowledge explication. The use of an ontology editor in parallel to the RELFIN software is suggested.
- The two-phase approach for discovering labelled topics may reach a good coverage of the text corpus, at least with a domain-specific ontology that is tailored towards supporting annotation.

In the future, we intend to further improve the integration of technologies and expert interaction models for semi-automatic ontology enhancement and annotation - possibly with focus on semantic web evolution.

Acknowledgments: We would like to thank the Parmenides consortium and especially the partner Unilever for their contribution to the experiment.

References

- [FN99] David Faure and Claire Nédellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system ASIUM. In Dieter Fensel and Rudi Studer, editors, *Knowledge Acquisition, Modeling and Management: 11th European Workshop, EKAW '99, Dagstuhl Castle, Germany, May 1999: Proceedings*, volume 1621 of *Lecture Notes in Computer Science*, pages 329–334. Springer-Verlag, Heidelberg, 1999.
- [GSW01] Henner Graubitz, Myra Spiliopoulou, and Karsten Winkler. The DI-AsDEM framework for converting domain-specific texts into XML documents with data mining techniques. In *Proc. of the 1st IEEE Intl. Conf. on Data Mining.*, pages 171–178, San Jose, CA, Nov. 2001. IEEE.
- [HSS03] Andreas Hotho, Steffen Staab, and Gerd Stumme. Explaining text clustering results using semantic structures. In *Proc. of ECML/PKDD 2003*, LNAI 2838, pages 217–228, Cavtat-Dubrovnik, Croatia, Sept. 2003. Springer Verlag.
- [HSV03] Siegfried Handschuh, Steffen Staab, and Raphael Volz. On deep annotation. In *Proceedings of the Twelfth International Conference on World Wide Web*, pages 431–438, Budapest, Hungary, May 2003. ACM Press.

- [KVM00] Jörg-Uwe Kietz, Raphael Volz, and Alexander Maedche. Extracting a domain-specific ontology from a corporate intranet. In Claire Cardie, Walter Daelemans, Claire Nédellec, and Erik Tjong Kim Sang, editors, *Proc. of 4th Conf. on Computational Natural Language Learning and of the 2nd Learning Language in Logic Workshop*, pages 167–175, Somerset, New Jersey, 2000. Association for Computational Linguistics.
- [MB01] G. William Moore and Jules J. Berman. Medical data mining and knowledge discovery. In *Anatomic Pathology Data Mining*, volume 60 of *Studies in Fuzziness and Soft Computing*, pages 72–117, Heidelberg, New York, 2001. Physica-Verlag.
- [MS00a] Alexander Maedche and Steffen Staab. Discovering conceptual relations from text. In *Proc. of ECAI'2000*, pages 321–325, 2000.
- [MS00b] Alexander Maedche and Steffen Staab. Mining ontologies from text. In *Proc. of Knowledge Engineering and Knowledge Management (EKAW 2000)*, LNAI 1937. Springer, 2000.
- [MS00c] Alexander Maedche and Steffen Staab. Semi-automatic engineering of ontologies from text. In *Proc. of 12th Int. Conf. on Software and Knowledge Engineering*, Chicago, IL, 2000.
- [RDH⁺03a] F. Rinaldi, J. Dowdall, M. Hess, J. Ellman, G. P. Zarri, A. Persidis, L. Bernard, and H. Karanikas. Multilayer annotations in parmenides. In *Proceedings of the K-CAP2003 workshop on "Knowledge Markup and Semantic Annotation"*, October 2003.
- [RDH⁺03b] Fabi Rinaldi, James Dowdall, Michael Hess, Kaarel Kaljurand, Andreas Persidis, Babis Theodoulidis, Bill Black, John McNaught, Haralampos Karanikas, Argyris Vasilakopoulos, Kelly Zervanou, Luc Bernard, Gian Piero Zarri, Hilbert Bruins Slot, Chris van der Touw, Margaret Daniel-King, Nancy Underwood, Agnes Lisowska, Lonneke van der Plas, Veronique Sauron, Myra Spiliopoulou, Marko Brunzel, Jeremy Ellman, Giorgos Orphanos, Thomas Mavrouidakis, and Spiros Taraviras. Parmenides: an opportunity for ISO TC37 SC4? In *ACL-2003 workshop on Linguistic Annotation*, Sapporo, Japan, July 2003.
- [RM99] Andreas Rauber and Dieter Merkl. Mining text archives: Creating readable maps to structure and describe document collections. In *Principles of Data Mining and Knowledge Discovery*, pages 524–529, 1999.
- [SB88] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
- [SKK00] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000.
- [SRB⁺04] Myra Spiliopoulou, Fabio Rinaldi, William J. Black, Gian Piero Zarri, Roland M. Mueller, Marko Brunzel, Babis Theodoulidis, Giorgos Orphanos, Michael Hess, James Dowdall, John McNaught, Maghi King, Andreas Persidis, and Luc Bernard. Coupling information extraction and data mining for ontology learning in parmenides. In *RIAO'2004, April 26th-28th*, Avignon, 2004.
- [VBB04] A. Vasilakopoulos, M. Bersani, and W.J. Black. A suite of tools for marking up textual data for temporal text mining scenarios. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-2004)*, Lisbon, 2004.

- [WS01a] Karsten Winkler and Myra Spiliopoulou. Extraction of semantic XML DTDs from texts using data mining techniques. In *Proceedings of the K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation*, pages 59–68, Victoria, BC, Canada, October 2001.
- [WS01b] Karsten Winkler and Myra Spiliopoulou. Semi-automated XML tagging of public text archives: A case study. In *Proceedings of EuroWeb 2001 “The Web in Public Administration”*, pages 271–285, Pisa, Italy, December 2001.
- [WS02] Karsten Winkler and Myra Spiliopoulou. Structuring domain-specific text archives by deriving a probabilistic XML DTD. In *6th European Conf. on Principles and Practice of Knowledge Discovery in Databases, PKDD’02*, pages 461–474, Helsinki, Finland, Aug. 2002. Springer Verlag.

Semantic Web-Based Document: Editing and Browsing in AktiveDoc

Vitaveska Lanfranchi¹, Fabio Ciravegna¹, and Daniela Petrelli²

¹ Department of Computer Science, University of Sheffield, Regent Court,
211 Portobello Street, S1 4DP, Sheffield, United Kingdom
{vita, fabio}@dcs.shef.ac.uk
<http://www.dcs.shef.ac.uk/>

² Department of Information Studies, University of Sheffield,
Regent Court, 211 Portobello Street, S1 4DP,
Sheffield, United Kingdom
d.petrelli@shef.ac.uk
<http://www.shef.ac.uk/~is/>

Abstract. This paper presents a tool for supporting sharing and reuse of knowledge in document creation (writing) and use (reading). Semantic Web technologies are used to support the production of ontology based annotations while the document is written. Free text annotations (comments) can be added to integrate the knowledge in the document. In addition the tool uses external services (e.g. a Semantic Web harvester) to propose relevant content to writing user, enabling easy knowledge reuse. Similar facilities are provided for readers when their task does not coincide with the author's one. The tool is specifically designed for Knowledge Management in organisations. In this paper we present and discuss how Semantic Web technologies are designed and integrated in the system.

1 Introduction

In the current form of the Web, content is designed and published for human readers and it is not typically tractable by machines; the Semantic Web, SW, is expected to make content processable in an automatic way via the addition of annotations. However, besides supporting automatic processing, the rich annotations behind the SW can improve the user's experience when dealing with documents and knowledge. Several methods of enriching Semantic Web documents have been proposed. One is to insert ontology-driven annotations that identify ontological instances in the document [8]. This type of annotation enables the capturing of document content, empowering better retrieval and reasoning.

A second method of enriching a document is to attach services: they can be associated to ontological instances and made available directly from the document in an automatic way [5]. Annotations and services can create a personalised view of the document, so that the reader can directly access its content (via ontology based annotation) and the additional information concerning it (i.e. the ontology and its

knowledge base can connect concepts in the document with external knowledge or documents -provided by the ontological-based services [7]).

A further way to enrich documents is to incorporate free text annotations in the form of comments [9]. They have become quite a standard feature¹, especially in the Knowledge Management (KM) world. Comments are used to integrate the text, adding information and knowledge not explicitly mentioned within the document: this is called *braindump*. For example, a lawyer could add explanations about referring to a specific regulation in a document (e.g. a EU directive), rather than others that could seem more relevant in the context of the document. In this case, comments are used for explaining the reasons that led to a specific formulation of the document itself, i.e. they are used to complement the knowledge in the document with knowledge about the process that generated it. A fundamental difference between *braindump* and ontology-based annotation is related to privacy. Typically, *braindump* is confined within an organisation's boundary as it contains the history, methodologies and motivations of a document; these are generally considered internal knowledge not to be shared with the outside world. As an example during the writing of this paper many comments were introduced by the authors as a way to discuss the paper content; however the form you are reading does not include it. The reader's *braindump* can comment not just the document itself, but even the author's annotations and comments. In this way it adds a further layer to the document knowledge.

All types of annotations, besides their different nature, share the same view of supporting "knowledge addition" by the different agents involved in the document lifecycle: e.g., the author(s) and the reader(s). Differences in the agent's role imply, in our view, a difference in management. Reader's tasks are different from the author's ones; for example, the ontology used for annotation can differ: inside an organisation a document may be written by the legal department using a legal ontology and accessed by the commercial department; the two departments are unlikely to share the same ontology.

In this paper, we propose to adapt and use SW technologies in order to support users during the lifecycle of a document, from production (writing), to consumption (reading) and maintenance (revision). By integrating the modalities of knowledge sharing offered by Semantic Web technologies, it is possible to create new opportunities for supporting users that can dramatically change the way document are written and read.

First and foremost, we claim that annotations (and especially ontological-based ones) must be generated as part of the document production step (i.e. editing) rather than during a separated step, as it happens in many of current approaches [8][11][5]. As a matter of fact, if annotations are added while documents are written, it is possible to use them to retrieve relevant content, moving from a situation in which they passively mark up the document content for future use (e.g. retrieval and reasoning), to one in which they contribute to reuse and sharing of knowledge during writing. This direction of research was preliminary explored by Carr et al [1] in the WickOffice editor, where knowledge about the domain of academics aided filling a

¹ Editorial tools like Microsoft Word and Adobe Acrobat provide tools to add comments.

pre-defined form (part A of an EPSRC project proposal). In that case, the knowledge used was static, i.e. its use was specified a priori by the application via the definition of the domain (knowledge about academics) and the form to be filled. The support was limited to filling pre-determined fields, while no additional knowledge was provided for writing the rest of the project proposal, especially its free text parts.

The system and the methodology we discuss here go one step further than that initial proposal. Conversely from the current technology our approach:

1. Supports all types of enrichment mentioned above (comments, ontologically-based and associated services, hyperlinking) both for author(s) and reader(s). Annotations can be added in layers, i.e. on top of other annotations.
2. Is able to automatically suggest ontological-based annotation so that annotations are immediately available and no separate annotation step is required.
3. Is able to monitor user actions while editing and to provide automatic suggestions about relevant content; support is not limited to filling forms and other pre-determined structures, but it is extended to free text as well; this enables timely reuse of existing knowledge when available.

The result is, in our view, a system that helps sharing and reusing existing knowledge. Its intended use is mainly for KM, in order to support sharing and reusing knowledge within an organisation. The system is called AktiveDoc and is discussed in the next section.

2 AktiveDoc

AktiveDoc is a system for supporting knowledge management in the process of document editing and reading. Its main feature is to support users (both readers and writers) in timely sharing and reusing relevant knowledge. In particular, document content and existing annotations are considered in the context of the user's previous knowledge; then further annotations and content are suggested for insertion. Proposals are gathered from different sources, i.e., from:

- *Information extraction processes applied on the document itself*: possible ontology-based annotations are automatically identified and proposed to the user. If accepted, they become part of the enriched document. Such annotations can be used again to connect to the KB and ontology as mentioned below.
- *Available structured knowledge*: existing annotations enable connections to knowledge bases and ontologies, so that part of the knowledge stored there can be proposed for inclusion in the document;
- *Querying the Web or other external repositories*, including querying both other documents in a repository and their annotations. Examples of tasks covered here are using a search engine or a SW harvester (e.g. Armadillo [2]) or retrieving pictures.

While many current systems modify the original document to add annotations, AktiveDoc saves them in a separate database. This is in order to allow levels of security and privacy: as mentioned above, there can be different levels of

confidentiality associated to the document enrichment. Therefore, annotations are stored in a database and superimposed to the document during the process of rendering according to a user profile. Annotations can be (1) public, (2) private or (3) confidential: only the ones for which the user has permission are actually displayed. An export facility allows producing a version of the document containing the required level of enrichment. For example, when the document is published (like the document you are reading now), only the annotations labelled as public are included. Documents are saved in a KM system that acts as a knowledge base: every document is logically associated to its annotations.

ActiveDoc’s architecture is composable so that it can accommodate different user scenarios. Composition is done by integrating different Semantic Web technologies and functionalities via SW Services. The rest of the section presents the system architecture and its main functionalities.

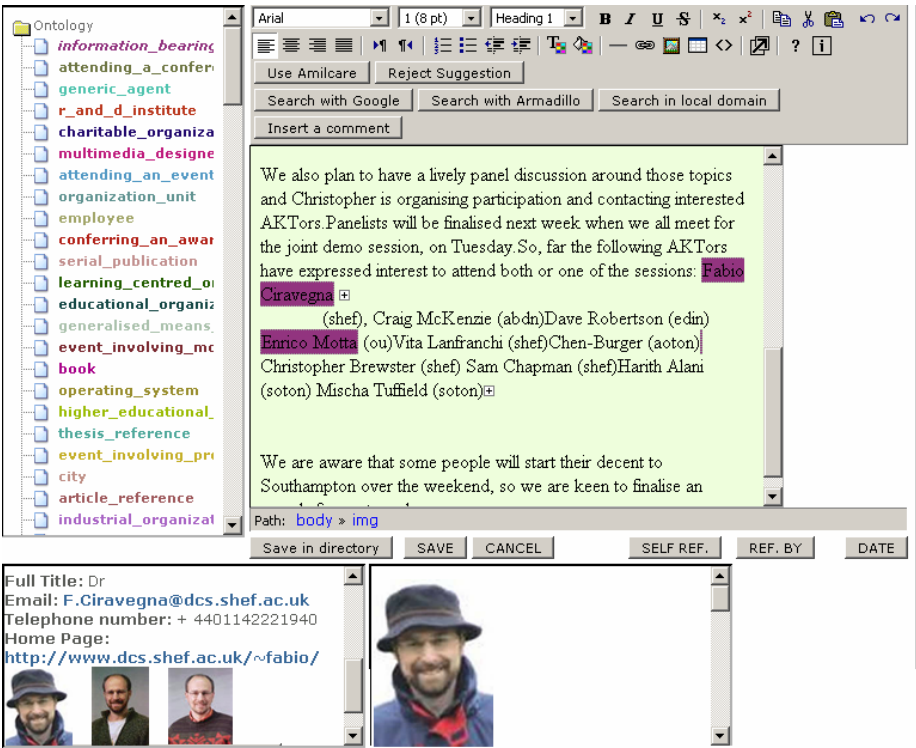


Fig. 1. The ActiveDoc interface during editing: the name (Fabio Ciravegna) has been annotated (in the centre); content suggestions from the system are displayed in the lower frames

2.1 User Interface

The actual appearance of the editor depends on the application it is used for. In general, the interface is composed by (Figure 1):

- An editing window (top right) with formatting commands organized in toolbars;
- The ontology on a side panel (left);
- A set of lower frames that visualise system suggestions, contributions and proposed annotations (in Figure 1 the results of searching a proper name with a search engine, and a set of pictures from a database);
- Braindumps presented in a way similar to MS Word (shown in Figure 3).

The actual appearance for a specific application is decided during a setup phase where the different services are connected (mainly using Semantic Web Services) and assigned a portion of the editor for outputting results or receiving input. Input is provided by highlighting portion of text in the main pane and activating a service (e.g. search with a search engine).

2.2 Ontology-Based Enrichment

Concerning ontology based enrichment, the following kinds of services are provided: manual and semi-automatic annotation, and association of services. Manual annotations requires users to associate (portions of) documents to the ontology or KB in a way similar to what required by tools like Cream. Like in Melita [3], a graphical interface is provided where colours are associated to entities; a mouse click is needed to associate the selected text to a concept. The ontology used to annotate the document is chosen by the user uploading it in RDF format.

Learning To Harvest Information For The Semantic Web

Fabio Ciravegna

Professor
University of Sheffield

Sam Chapman

Alexiei Dingli

Yorick Wilks

Copyright © 2004 f.ciravegna@dcs.shef.ac.uk

Abstract

In this paper we describe a methodology for harvesting information from large distributed repositories (e.g. large Web sites) with minimum user intervention. The methodology is based on a combination of information extraction, information integration and machine learning techniques. Learning is seeded by extracting information from structured sources (e.g. databases and digital libraries) or a user-defined lexicon. Retrieved information is then used to partially annotate documents. Annotated documents are used to bootstrap learning for simple Information Extraction (IE) methodologies, which in turn will produce more annotation to annotate more documents that will be used to train more complex IE engines and so on. In this paper we describe the methodology and its implementation in the

[View Related Services](#) [hide](#)

[Google Search](#)
[View Related Item](#)
[View Personal Information](#)

[Author Related Information](#) [hid](#)

Name: Fabio
Surname: Ciravegna
Title: Dr
Mail: F.Ciravegna@dcs.shef.ac.uk
Telephone Number: +44011422219
Home Page:
[Http://www.Dcs.Shef.Ac.Uk/~Fabio](http://www.Dcs.Shef.Ac.Uk/~Fabio)

Fig. 2. (part of the) the interface for browsing a document showing one suggestion for additional content. No editing facilities are provided, except for comments and adding annotations

When working in a semi-automatic approach, the system learns from previous annotations how to suggest annotations while the document is edited. Again, the

reference model is Melita's. Users can accept or reject annotations. User reactions to suggestions are used for further learning. The automatic detection provides efficiency (i.e. annotation is instantly available) while user corrections provide precision (i.e. only the truly important information is actually included in the annotation). Learning is based on a machine learning system connected to the interface. The current implementation connects to Amilcare [4]. The cycle annotation, correction and retrain is imported from Melita and the same strategy for avoiding intrusivity is used here [3]. The difference is that in the original implementation, Melita did not allow editing of document. This required implementing a specific annotation step. Moreover, it prevented the implementation of the strategies for content suggestion during editing mentioned below.

2.3 Inserting Unstructured Annotations (Braindumps)

AktiveDoc accommodates insertion of free text comments into the document. As mentioned before, braindump can be done at any stage of the document lifecycle, both while editing and while reading. As in other tools, to insert a comment, the portion of document is highlighted and a button "Add Comment" is pressed. The comment will then be shown as a "plus" in the editor window (Figure 3). The comment can be expanded by clicking on the "plus".

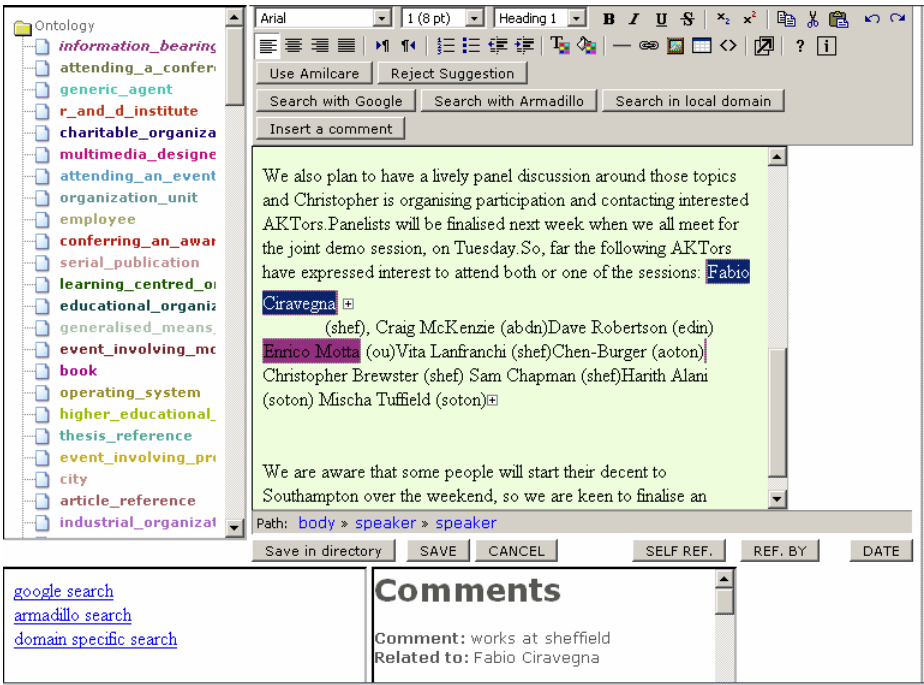


Fig. 3. The AktiveDoc interface while comments are inserted: a "+" marks a comment (related to the concept Fabio Ciravegna); in the lower-right frame all the comments are listed

Comments are organized in layers: a user may add comments to other users' comments; they can also add annotations to their or other people's comments. Finally, they can add System-suggested content (see next subsection) directly into comments. As for any type of annotation comments are stored in the database with authorship and level of confidentiality, to guarantee privacy and security.

2.4 Supporting Content Generation

As mentioned, one of the main features of AktiveDoc are active suggestions of relevant content to authors and readers in order to enable knowledge sharing and timely reuse. Knowledge is retrieved by external composable Web Services that exchange knowledge with the editor via the ontology.

When a portion of document is selected, services are made available depending on the annotations contained in the portion of text (if any) and on the string. For example, in Figure 3 in the lower-left frame the services associated to the annotated string "Fabio Ciravegna" are shown. Information about the individual are retrieved from a KB and other services are made available. Services are associated to types in the ontology and depend on the specific application.

A similar process was proposed in Magpie [5]. The difference with Magpie is that in Magpie annotation is not provided while editing but only in displaying the document. As mentioned, providing services during editing enables retrieving new content to be added to the document, therefore knowledge sharing and reuse is possible for the author and not only for the reader. Also, in Magpie annotations are generated only using a named entity recognizer (eSpotter [12]), therefore they are quite shallow; moreover, a rule based Named Entity Recognizer is used, and any addition of coverage to the recognizer requires rule writing by an expert. In AktiveDoc, annotations are either produced automatically by a system that learns from examples (Amilcare) or manually by the user. Also annotation is not limited to generic named entity recognition. In the current implementation connections to Armadillo (knowledge harvester) [2], Search engines (e.g. Google) and to structured resources (databases and knowledge bases [6]) are provided as shown in Figure 3 in the lower-left.

The activation of services is not automatic, but it requires a user action. This is done in order both to avoid spending CPU time on irrelevant tasks (that is one of the requirements for non intrusivity for automatic annotation [2]) and to avoid overwhelming users with (possibly irrelevant) suggestions (another requirement for automatic annotation).

Suggestions are presented to the user in a frame different from the one used for the document (and on which the user is working), so to avoid distracting the user attention when they appear. The interface currently allows both textual content (as in content retrieved from Google Web Service or Armadillo RDF repository) and multimedia content to be displayed (e.g. images retrieved by Armadillo see Figure 1).

Visualised suggestions can be inserted either directly into the document or as comment by dragging them in the wanted position.

3 Architecture and Implementation

ActiveDoc is a client-server application integrated in a Web Based KM System. The system is based on an interface that interacts with user's actions and timely calls the appropriate modules for executing the actions. The main system's components are: (1) Annotation module and (2) Information module.

The information module (IM) is in charge of connecting to the appropriate information source and retrieving content to be suggested to the user; when the user selects a portion of document, the IM extracts the string and the contained annotations (if any) and sends them to the available services. Then it collects their results and presents it into the user interface. The annotations module is responsible for saving the annotations inserted by the user and for retrieving the automatic annotations provided by the system via Amilcare. Both modules are integrated in the user interface and are active only when a user's action requests them.

The system contains also an Ontology Module that is in charge of interpreting the RDF ontology the user loads and of visualizing it using appropriate style sheets. The editor interface is based on HTMLArea, a free, open source utility developed by interactivetools.com to convert a <textarea> field into a WYSIWYG editor. In this way HTML documents can easily be opened and visualized correctly in the system and new documents can be created using the formatting facilities offered by HTML. Several frames are inserted in the main interface to allow displaying the ontology, the available services and the retrieved content. The interface is based on Javascript and JSP functions. Documents and annotations are saved in a MySQL database. Support for Semantic Web Services is provided by the Armadillo infrastructure [10].

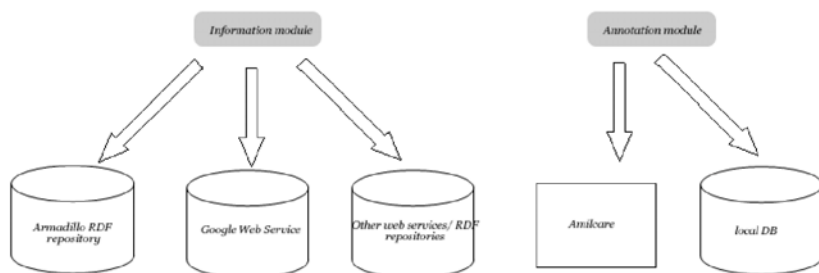


Fig. 5. An example of ActiveDoc application architecture

4 Discussion and Conclusions

In this paper we have presented a tool for supporting users during the lifecycle of a document, from production (writing) to consumption (reading and publishing). The tool integrates Semantic Web technologies to support users in adding knowledge to the document. In particular, writers can receive suggestions on relevant content (enabling reuse of knowledge) and can be supported in producing ontology-based annotations (that will empower the Semantic Web). Also textual comments are

enabled in order to add further knowledge to documents. Potentially² the system is also able to suggest relevant content after the document is published, in order to allow document maintenance (e.g. newly available content could be communicated to the author even after the release of the paper).

Readers are able to access the enriched document produced by the authors using AktiveDoc. Moreover, in case their task differs from the author's one (e.g. they use a different ontology), they receive the same kind of support authors receive (relevant content suggestion, ontology based annotation, etc.); this enables reading in the context of their knowledge rather than in the author's one.

AktiveDoc has been designed mainly with Knowledge Management in mind, specifically in order to help reusing and sharing knowledge. These are fundamental needs in enterprises: according to some recent statistics, knowledge workers spend between 15% and 35% of their time in searching for knowledge³. Also, "lack of efficient publishing capabilities for digital content costs organizations \$750 billion annually due to wasted time spent by knowledge workers seeking and capturing information necessary for them to do their jobs⁴". By providing both ontology-based annotations and the suggestion of relevant content, we enable knowledge reusing. Knowledge sharing is empowered by layered comments and also by the searching capabilities provided by ontology-based annotations.

Annotations and services are stamped with authorship and are not saved in the document, so to allow confidentiality when needed. Marking authorship has also two other positive side effects. On the one hand it can contribute to identifying experts, a well known problem in large organizations. Associating annotations to documents means having coped with a problem, therefore it is possible to identify who works on specific problems by inspecting what documents a person has worked on. In traditional environments only the author can be tracked, not the readers. On the other hand, it allows implementing strategies of company management that rewards who shares knowledge within the company. The amount of sharing can be counted starting from the value and quantity of annotations provided to documents.

Future work on AktiveDoc will include the extension of the base of services provided and a field user test in a KM environment.

References

1. L. Carr, T. Miles-Board, A. Woukeu, G. Wills and W. Hall. The Case for Explicit Knowledge in Documents. In Proceedings of ACM Symposium on Document Engineering , pages pp. 90-98, Milwaukee, Wisconsin.
2. Fabio Ciravegna, Sam Chapman, Alexiei Dingli, Yorick Wilks. Learning to Harvest Information for the Semantic Web. In Proceedings of the 1st European Semantic Web Symposium, Heraklion, Greece, May 10-12, 2004

² This is a potential feature because it has not been implemented yet.

³ KMWorld Volume 13, Issue 3, March 2004.

⁴ A.T. Kearney, Network Publishing study, April 2001.

3. Fabio Ciravegna, Alexiei Dingli, Daniela Petrelli and Yorick Wilks. User-System Coopera-tion in Document Annotation based on Information Extraction. In Asuncion Gomez-Perez, V. Richard Benjamins (eds.): Knowledge Engineering and Knowledge Management (Ontologies and the Semantic Web), Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), 1-4 October 2002 - Sigüenza (Spain), Lecture Notes in Artificial Intelligence 2473, Springer Verlag
4. Fabio Ciravegna and Yorick Wilks: Designing Adaptive Information Extraction for the Semantic Web in Amilcare, in S. Handschuh and S. Staab (eds), Annotation for the Seman-tic Web, in the Series Frontiers in Artificial Intelligence and Applications by IOS Press, Amsterdam, 2003.
5. M. Dzbor, J. Domingue and E. Motta: Magpie: Towards a Semantic Web Browser. In Proc. of the 2nd Intl. Semantic Web Conf. (ISWC). 2003. Florida, USA.
6. Hugh Glaser, Harith Alani, Les Carr, Sam Chapman, Fabio Ciravegna, Alexiei Dingli, Nicholas Gibbins, Stephen Harris, M.C. Schraefel, and Nigel Shadbolt CS AKTiveSpace: Building a Semantic Web Application. In Proceedings of the 1st Euro-pean Semantic Web Symposium, Heraklion, Greece, May 10-12, 2004
7. C. A.Goble, S. Bechhofer, L. Carr, D. De Roure, W. Hall. Conceptual Open Hypermedia = The Semantic Web?. In Proc. Of The Second International Workshop on the Semantic Web, Hong Kong, 2001.
8. S. Handschuh, S. Staab. CREAM - CREATing Metadata for the Semantic Web. Computer Networks. 42, pp. 579-598, Elsevier 2003
9. J. Kahan, M. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An Open RDF Infra-structure for Shared Web Annotations. In Proc. of the WWW10 International Conference. Hong Kong, 2001.
10. Barry Norton, Sam Chapman and Fabio Ciravegna. Developing a Service-Oriented Archi-tecture to Harvest Information for the Semantic Web. In Proc. Of 1st AKT Workshop on Semantic Web Services, 2004.
11. Maria Vargas-Vera, Enrico Motta, John Domingue, Mattia Lanzoni, Arthur Stutt and Fabio Ciravegna "MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup", The 13th International Conference on Knowledge Engineering and Management (EKAW 2002), ed Gomez-Perez, A., Springer Verlag, 2002.
12. Jianhan Zhu, Victoria Uren, and Enrico Motta. ESpotter: Adaptive Named Entity Recogni-tion for Web Browsing. To appear in Proc. of Workshop on IT Tools for Knowledge Management Systems at WM2005 Conference, Kaiserslautern, Germany, April 11-13, 2005.

Semantic-Based Automated Composition of Distributed Learning Objects for Personalized E-Learning

Simona Colucci^{1,3}, Tommaso Di Noia¹, Eugenio Di Sciascio¹,
Francesco M. Donini², and Azzurra Ragone¹

¹ Politecnico di Bari, Via Re David, 200, I-70125, Bari, Italy
{s.colucci, t.dinoia, disciascio, a.ragone}@poliba.it

² Università della Tuscia, via San Carlo, 32, I-01100, Viterbo, Italy
donini@unitus.it

³ Knowledge Media Institute, The Open University, MK7 6AA, United Kingdom

Abstract. Recent advances in e-learning technologies and web services make realistic the idea that courseware for personalized e-learning can be built by dynamic composition of distributed learning objects, available as web-services. To be assembled in an automated way, learning objects metadata have to be exploited, associating unambiguous and semantically rich descriptions, to be used for such an automated composition. To this aim, we present a framework and algorithms for semantic-based learning objects composition, fully compliant with Semantic Web technologies. In particular our metadata refer to ontologies built on a subset of OWL-DL, and we show how novel inference services in Description Logics can be used to compose dynamically, in an approximated –but computationally tractable– way learning resources, given a requested courseware description.

1 Introduction

Since the beginnings, the World Wide Web has played a key role in changing the way learning and teaching were delivered. The term e-learning has become common, describing several concepts, from complete web-based courses to distance learning and tutoring.

Recently, also thanks to various standardization efforts [2], emphasis has been placed on the concept of learning object *i.e.*, small and easily reusable educational resources to be composed to allow personalized instruction and courseware creation [21, 9, 4, 30].

Obviously, discovery and composition –according *e.g.*, to prerequisite material– of such learning objects in an automated way requires the association of unambiguous and semantically rich metadata, defined in accordance with shared ontologies. The LOM (Learning Object Metadata) [1] standard, though limited in the basic annotation items, allows to freely define annotated metadata describing a learning resource.

The semantic-based annotation of educational resources is hence fully in the stream of the Semantic Web initiative [29], and it can share with it both techniques and approaches [26, 8, 17]. In particular, as more and more learning objects become available on the Web as services with well-defined machine interpretable interfaces as described

e.g., in OWL-S [28, 22], personalized learning units can be built by scratch, by retrieving learning resources, which are in this scenario semantic-enabled web services themselves [25, 19, 15]. Automated composition of learning resources, exposed as web services for example, can then match a personalized learning need.

In this paper we propose a framework and an approach where, given a specification of courseware described using a subset of OWL-DL [24], we are able to discover and compose, using semantics of descriptions, learning resources covering as much as possible the learning need, and orchestrating resources according to specified prerequisites. Furthermore, thanks to recently introduced inference services, when available resources are unable to fulfill the needs, our approach provides an explanation of what is missing to fully cover the request. We show how this is obtained by presenting a greedy algorithm for Concept Covering in a subset of OWL-DL and exploit it for semantic service discovery. Our framework also allows to carry out the assembly and integration of learning objects. The greedy algorithm takes into account approximate solutions and is computed in polynomial time.

The remaining of the paper is structured as follows: next section describes basics of Description Logics and the subset of OWL-DL we concentrate on; then we present our extension of Concept Covering [18] definition and a greedy algorithm, which uses Concept Abduction [13], to determine a Concept Cover. In Section 3 we exploit the previously defined Concept Covering algorithm in a general framework to carry out a semantic learning objects assembly and orchestration. The approach, and behavior of the related algorithms, are thoroughly explained with the aid of an example in Section 5. Last Section draws the conclusions and outlines future research directions.

2 Basic of Description Logics

We start with a brief guided tour of Description Logics (DLs) and their interaction with novel languages for the Semantic Web. DLs[5] are a family of logic formalisms whose basic syntax elements are *concept* names, e.g., *WebService*, *ProceduralLanguage*, *Java*, and *role* names, such as *allowedTechnologies*, *hasProgrammingLanguages*. Intuitively, concepts stand for sets of objects, and roles link objects in different concepts. Formally, concepts are interpreted as subsets of a domain of interpretation Δ , and roles as binary relations (subsets of $\Delta \times \Delta$). Basic elements can be combined using *constructors* to form concept and role *expressions*, and each DL has its distinguished set of constructors. Every DL allows one to form a *conjunction* of concepts, usually denoted as $\sqcap$; some DL include also disjunction $\sqcup$ and complement $\neg$ to close concept expressions under boolean operations. Roles can be combined with concepts using *existential role quantification*, e.g., $\text{WYSIWYGtool} \sqcap \exists \text{allowedTechnologies.WebService}$, which describes a WYSIWYG tool allowing Web services development, and *universal role quantification*, e.g., $\text{IDE} \sqcap \forall \text{hasProgrammingLanguage.Java}$, which describes a tool to handle only java code. Other constructs may involve counting, as number restrictions: $\text{Tool} \sqcap (\leq 2 \text{ hasProgrammingLanguage}) \sqcap \forall \text{hasProgrammingLanguage.OOL}$ expresses a tool allowing at most two different kinds of object oriented languages. Many other constructs can be defined,

increasing the expressive power of the DL, up to n-ary relations [10]. Concept expressions can be used in *inclusion assertions*, and *definitions*, which impose restrictions on possible interpretations according to the knowledge elicited for a given domain. For example, we could impose that a IDE is a WYSIWYG tool using the following inclusion: $\text{IDE} \sqsubseteq \exists \text{develTool} \sqcap \forall \text{develTool.WYSIWYGtool}$. Definitions are useful to give a meaningful name to particular combinations, as in $\text{OOP} \equiv \forall \text{programming}.\forall \text{language.OOL}$. Historically, sets of such inclusions are called TBox (Terminological Box). The basic reasoning problems for concepts in a DL are satisfiability, which accounts for the internal coherency of the description of a concept (no contradictory properties are present), and subsumption, which accounts for the more general/more specific relation among concepts, that forms the basis of a taxonomy. More formally, a concept C is satisfiable if there exists an interpretation in which C is mapped into a nonempty set unsatisfiable otherwise. If a TBox $\mathcal{T}$ is present, satisfiability is relative to the models of $\mathcal{T}$, that is, the interpretation assigning C to a nonempty set must be a model of the inclusions in $\mathcal{T}$. A concept C subsumes a concept D if every interpretation assigns to C a subset of the set assigned to D . Also subsumption is usually established relative to a TBox, a relation that we denote $\mathcal{T} \models C \sqsubseteq D$. Also a TBox can be said satisfiable if there exist at least one model (i.e., an interpretation fulfilling all its inclusions in a nontrivial way).

It is easy to see that $\mathcal{T}$ in DLs represents what is called an ontology in a knowledge representation system. In the rest of the paper we refer to the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) DL, a subset of OWL-DL. Constructs allowed in an $\mathcal{ALN}$ DL are:

- $\top$ *universal concept*. All the objects in the domain.
- $\perp$ *bottom concept*. The empty set.
- A *atomic concepts*. All the objects belonging to the set represented by A .
- $\neg A$ *atomic negation*. All the objects not belonging to the set represented by A .
- $C \sqcap D$ *intersection*. The objects belonging both to C and D .
- $\forall R.C$ *universal restriction*. All the objects participating to the R relation whose range are all the objects belonging to C .
- $\exists R$ *unqualified existential restriction*. There exists at least one object participating in the relation R .
- $(\geq n R) | (\leq n R) | (= n R)$. Respectively the minimum, the maximum and the exact number of objects participating in the relation R .

We use a *simple-TBox* in order to express the relations among objects in the domain. With a *simple-TBox* in all the axioms (for both inclusion and definition) the left side is represented by a concept name.

Ontologies using the above logic can be easily modeled using languages for the Semantic Web [12, 23, 24]. These languages have been conceived to allow for representation of machine understandable, unambiguous, description of web content through the creation of domain ontologies, and aim at increasing openness and interoperability in the web environment. The strong relations between DLs and the above introduced languages for the Semantic Web [7] is also evident in the definition of the three OWL sub-languages. OWL-Lite allows class hierarchy and simple constraints on relation between classes; OWL-DL is based on Description Logics theoretical studies, it allows a

Table 1. Correspondence between OWL-DL and $\mathcal{ALN}$ DL syntax

OWL syntax	DL syntax
$\langle owl : Thing / \rangle$	$\top$
$\langle owl : Nothing / \rangle$	$\perp$
$\langle owl : Classrdf : ID = "C" / \rangle$	C
$\langle owl : ObjectPropertyrdf : ID = "R" / \rangle$	R
$\langle rdfs : subclassOf / \rangle$	$\sqsubseteq$
$\langle owl : equivalentClass / \rangle$	$\equiv$
$\langle owl : disjointWith / \rangle$	$\neg$
$\langle owl : intersectionOf / \rangle$	$\sqcap$
$\langle owl : allValuesFrom / \rangle$	$\forall$
$\langle owl : someValuesFrom / \rangle$	$\exists$
$\langle owl : maxCardinality / \rangle$	$\leq$
$\langle owl : minCardinality / \rangle$	$\geq$
$\langle owl : cardinality / \rangle$	$=$

great expressiveness keeping computational completeness and decidability; OWL-Full: using such a language, there is a huge syntactic flexibility and expressiveness. This freedom is paid in terms of no computational guarantee.

The subset of OWL-DL Tags allowing to express an $\mathcal{ALN}$ DL is presented in Table 1. In the rest of the paper we will use DL syntax instead of OWL-DL syntax, to make expressions much more compact.

3 Concept Covering in DLs

Standard inference services in DLs include subsumption and satisfiability. These are enough when a yes/no answer is needed. However there are scenarios that require explanation. In [13] the Concept Abduction Problem (CAP) was introduced and defined as a non standard inference problem for DLs to provide an explanation when subsumption does not hold.

Definition 1. Let C , D , be two concepts in a Description Logic $\mathcal{L}$, and T be a set of axioms, where both C and D are satisfiable in T . A Concept Abduction Problem (CAP), denoted as $\langle \mathcal{L}, C, D, T \rangle$, is finding a concept H such that $T \not\models C \sqcap H \equiv \perp$, and $T \models C \sqcap H \sqsubseteq D$.

We use $\mathcal{P}$ as a symbol for a CAP, and we denote with $SOL(\mathcal{P})$ the set of all solutions to a CAP $\mathcal{P}$.

In [13] also minimality criteria for H and a polynomial algorithm to find solutions which are irreducible, for an $\mathcal{ALN}$ DL, have been proposed.

Given a CAP, if H is a conjunction of concepts and no sub-conjunction of concepts in H is a solution to the CAP, then H is an **irreducible solution**. The *rankPotential* algorithm [14] allows to numerically compute the *length* of H .

The solution to a CAP can be interpreted as *what has to be hypothesized in C , and in a second step added to, in order to make C more specific than D ?* In other words

H is what is expressed, explicitly or implicitly, in D and is not present in C , or again which part of D is not covered by C . On the basis of the latter remark in the following we will show how to use concept abduction to perform a "concept covering".

In [18] the *best covering problem* in DLs was introduced as "...a new instance of the problem of rewriting concepts using terminologies".

That is, given a concept C and a set of concept definitions in a terminology $\mathcal{T}$, find concepts defined in $\mathcal{T}$ such that their conjunction can be an approximation of C .

In order to define a concept covering two non standard inference services were there used: the least common subsumer (*lcs*)[6] and the *difference* or *subtraction* operation [27]. Unfortunately, as the authors admitted, the difference operator makes sense only for a limited set of DLs, and surely not for the $\mathcal{ALN}$ (we do not delve into details, for a complete description see [27]).

In a more formal way the authors of [18] defined *cover* as follows.

Definition 2. Let $\mathcal{L}$ be a Description Logic with structural subsumption, $\mathcal{T}$ be a terminology using operator allowed by $\mathcal{L}$, $\mathcal{R}$ be the set of concept definitions in $\mathcal{T}$, $\mathcal{R} = \{S_i, i \in [1..n]\}$, and D be a concept in $\mathcal{L}$ such that $\mathcal{T} \not\sqsubseteq D \equiv \perp$. A cover of a D using $\mathcal{T}$ is finding a set $\mathcal{R}_c \subseteq \mathcal{R}$ such that $\sqcap S_i$, conjunction of all the $S_i \in \mathcal{R}_c$ is such that $D - lcs_{\mathcal{T}}(D, \sqcap S_i) \not\equiv D$.

That is, a cover is finding a set of concepts defined in $\mathcal{T}$ such that they contain the information in D . Notice that a DL with structural subsumption is needed in order to use *concept difference*. In [18] also an hypergraphs based methodology is presented to compute best covers.

We extended the previous definition, in terms of a Concept Covering Problem, both eliminating limitations on $\mathcal{L}$ to be used and rewriting it in terms of Concept Abduction.

Definition 3. Let D be a concept, $\mathcal{R} = \{S_1, S_2, \dots, S_k\}$ be a set of concepts in a Description Logic $\mathcal{L}$, and $\mathcal{T}$ be a set of axioms, where D and $S_i, i = 1..k$ are satisfiable in $\mathcal{T}$.

1. A Concept Covering Problem (CCoP), denoted as $\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$, is finding, if it exists, a set $\mathcal{R}_c \subseteq \mathcal{R}$, such that both for each $S_j \in \mathcal{R}_c$, $\mathcal{T} \not\sqsubseteq \sqcap S_j \equiv \perp$, and $H \in SOL(\langle \mathcal{L}, \sqcap S_j, D, \mathcal{T} \rangle)$ is such that $H \not\sqsubseteq D$.
2. We call $\langle \mathcal{R}_c, H \rangle$ a solution for the CCoP $\langle \mathcal{L}, \mathcal{R}, D, \mathcal{T} \rangle$.

In the above definition the elements for the solution $\langle \mathcal{R}_c, H \rangle$ of a CCoP represent respectively:

- $\mathcal{R}_c$. Which concepts in $\mathcal{R}$ represent the cover for D w.r.t. $\mathcal{T}$.
- H . What is still in D and is not covered by concepts in $\mathcal{R}$.

We say that $\mathcal{R}_c$ **covers** D and we use the symbol $\mathcal{V}$ for CCoP and $SOLCCoP(\mathcal{V})$ for the set of all the solution to a CCoP $\mathcal{V}$. Actually, there are several solution for a single CCoP, depending also on the strategy adopted for searching concepts belonging to $\mathcal{R}_c$. Based on the definition of Concept Covering Problem we now define the *best cover* and the *exact cover*.

Definition 4. Given $\mathcal{V}$, a best cover for $\mathcal{V}$, w.r.t. an order $\prec$ for $\mathcal{L}$, is a solution $\langle \mathcal{R}_c, H_b \rangle \in \text{SOLCCoP}(\mathcal{V})$ such that there is no other $\langle \mathcal{R}'_c, H' \rangle \in \text{SOLCCoP}(\mathcal{V})$ with $H' \prec H_b$.

There is no solution $\langle \mathcal{R}'_c, H' \rangle$ for $\mathcal{V}$ such that H' , the remaining part of D yet to be covered, is *smaller* than H_b .

Definition 5. Given $\mathcal{V}$, a full cover for $\mathcal{V}$ is a solution $\in \text{SOLCCoP}(\mathcal{V})$ such that $H_e \equiv \top$.

Having a set $\mathcal{R}$ of concepts $S_i, i = 1..k$, we want to find a subset $\mathcal{R}_c$ of $\mathcal{R}$, if it exists, such that the conjunction of all the concepts in $\mathcal{R}_c$ is more specific than, i.e., it is subsumed by, D . In other words, we are looking for a set of concepts that completely cover D .

3.1 An Algorithm to Solve a CCoP

It is well known that the general set-covering problem is NP-Hard. Here we adapt a tractable greedy set-covering algorithm [11] to compute a CCoP.

Algorithm *GREEDYsolveCCoP*($\mathcal{R}, D, \top$)

input concepts $D, S_i \in \mathcal{R}, i = 1..k$, where D and S_i are satisfiable in $\mathcal{T}$

output $\langle \mathcal{R}_c, H \rangle$

begin algorithm

$\mathcal{R}_c = \emptyset$;

$D_{uncovered} = D$;

$H_{min} = D$;

do

$S_{min} = \top$;

/* ♣ Perform a greedy search among $S_i \in \mathcal{R}$ */

for each $S_i \in \mathcal{R}$

if $\mathcal{R}_c \cup \{S_i\}$ is a cover for $D_{uncovered}$ **then**

$H = \text{solveCAP}(\langle \mathcal{L}, S_i, D_{uncovered}, \top \rangle)$;

/* ⋄ Choose S_i based on an order */

if $H \prec H_{min}$ **then**

$S_{min} = S_i$;

$H_{min} = H$;

end if

end if

end for each

/* ♠ If a new S_i is found then add S_i to $\mathcal{R}_c$

and remove it from $\mathcal{R}$ */

if $S_{min} \not\equiv \top$ **then**

$\mathcal{R} = \mathcal{R} \setminus \{S_i\}$;

$\mathcal{R}_c = \mathcal{R}_c \cup \{S_i\}$;

$D_{uncovered} = H_{min}$;

end if

/* ♥ Continue searching until no S_i is found */

while($S_{min} \not\equiv \top$);

return $\langle \mathcal{R}_c, D_{uncovered} \rangle$;

end algorithm

The algorithm tries to cover D *as much as possible*, using the concepts $S_i \in \mathcal{R}$.

♥ If no new useful $S_i \in \mathcal{R}$ is found, that is any S_i such that it covers D more, then the algorithm terminates.

♣ A greedy approach is used to choose the *candidates* for $\mathcal{R}_c$.

- ◇ Choose among the candidates the one such that H , solution for the local CAP, is minimal w.r.t. an order $\prec$.
- ♠ If the greedy search returns a new S_i , it is removed from $\mathcal{R}$ and added to $\mathcal{R}_c$.

In [11] it is proved that, for a set covering problem, the solution grows logarithmically in the size of the set to be covered with respect to the minimal one. Hence the complexity source is in the solution of the CAPs and the comparison in [◇]. For the $\mathcal{ALN}$ DL, in [13] a polynomial algorithm (*findIrred*) is proposed to find irreducible solutions for a CAP, and in [14] the tractable *rankPotential* is presented to rank concepts. Using such algorithms it can be easily proved that also *GREEDYsolveCCoP* can be solved in polynomial time. Obviously we are not claiming that we solve a covering problem polynomially. The algorithm returns *a cover**, not the best one.

4 Semantic Enabled Learning Object Composition

In the following we present how to exploit the DL standard and non-standard inference services in order to perform an automated composition of Learning Objects to assemble personalized learning objectives. Here we do not refer to a particular model specification but we propose a general framework based on OWL technologies for composition which can be easily integrated in existing metadata specifications, such as SCORM [3], LOM [1], IMS [20], Dublin Core [16].

Hereafter we will refer only to a specific portion of the Learning Object (λ) model, but the approach can be easily extended taking into account all the information in the model.

In our view, the discovery of Learning Objects to be composed is a sub-problem of the more generic resource retrieval one. In this perspective, if there is a **learning request** and a repository of learning objects potentially satisfying the learner specifications, a solution to a λ -retrieval problem is:

retrieve (a sequence of) some λ s from the repository such that their composition satisfies the request as far as possible.

Notice that we are not necessarily interested in a full satisfaction of the user request; we want to satisfy it *as much as possible*.

If the system is not able to extract a set of objects from the repository such that they completely fulfill the search goal, an approximate solution has to be taken into account, possibly explaining the approximation.

To introduce and motivate the approach, we start with a model where both the learning request, denoted as ρ , and the λ information needed for discovery are represented by an OWL-DL annotation. Then we will enrich the model adding features to compose the discovered λ s.

In the initial model we define both ρ and the description of each learning object λ_D simply as DL concept descriptions w.r.t. an ontology $\mathcal{T}$. We assume the existence of a λ s repository, where the all the information related to each λ is stored with respect to a generic learning object model.

Given a λ request ρ modeled w.r.t. an ontology $\mathcal{T}$, the steps needed in order to obtain a set of λ s satisfying ρ as much as possible are hence the following:

1. query the repository in order to obtain all the λ descriptions, λ_D , referring to the same $\mathcal{T}$. That is, they could perform the task required by the user.
2. create the set $\mathcal{R}$ collecting all the retrieved λ_D .
3. compute the solution $\langle \mathcal{R}_c, H \rangle$ for the *CCoP* $\langle \mathcal{L}, \mathcal{R}, \rho, \mathcal{T} \rangle$ using the algorithm *GREEDYsolveCCoP*($\mathcal{R}, \rho, \mathcal{T}$).
4. referring to $\langle \mathcal{R}_c, H \rangle$ computed in the previous step, return to the user both $\mathcal{R}_c = \{\lambda_i\}$, representing the set of learning objects in the repository satisfying ρ , and H as an explanation of what is not specified in any $\lambda_i \in \mathcal{R}_c$.

$\mathcal{R}_c$ and H respectively represent the set of λ corresponding to an approximate solution to the retrieval problem and the explanation why the solution is not an exact one.

Using the above approach a **discovery** process is performed for the λ s in the repository, which can be composed in order to satisfy ρ as far as possible. Obviously, the discovery of all the services in the repository is a trivial solution to the problem, of no interest.

The **composition** of the discovered $\lambda_i \in \mathcal{R}_c$ requires further information to be taken into account. Some λ may require background knowledge from the learner. If the user does not hold specific knowledge, then she/he is not able to benefit from the use of λ . The learner can get such information using the previous λ in a composition flow.

In order to benefit from the use of a λ , the user knowledge must satisfy the background knowledge required from λ .

4.1 Background Knowledge for Automated Lesson Composition

In order to deal with the execution information, we extend the previous model and define:

Learning Request: $\rho = \langle \rho_D, \rho_{BK} \rangle$, where ρ_D is the description of the requested lesson and ρ_{BK} represents the background knowledge owned by the requester before looking for the courseware.

Learning Object:¹ $\lambda = \langle \lambda_D, \lambda_{BK} \rangle$. λ_D describes the knowledge the user will acquire after she/he uses λ . Using a language endowed with a well-defined syntax and semantics, it models the offered knowledge. λ_{BK} is a representation of prerequisites in order to benefit from λ .

ρ_D , ρ_{BK} , λ_D and λ_{BK} are modeled using OWL DL statements referring to an OWL DL task ontology. Notice that $\rho_{BK} = \top$ or $\rho_{BK} = \perp$ means, respectively, that the user has not any knowledge related to the field she/he wants to learn about and that no background knowledge is needed in order to benefit from the learning object.

For the sake of clarity, from now on we will model OWL DL expressions using their equivalent DL formulation.

¹ Without loss of generality here we consider only the information needed for a semantic discovery and composition.

A simple covering solution, as the one proposed above, cannot deal with the ρ_{BK} , λ_{BK} specifications of the background knowledge respectively owned by the user and required to benefit from λ . To compose Learning Objects dealing with their required background knowledge, we introduce a definition of composite courseware.

A **courseware flow** with respect to some initial background knowledge ρ_{BK} , denoted as $\overline{\Lambda}(\rho_{BK})$, is a finite sequence of learning objects $(\lambda^1, \lambda^2, \dots, \lambda^i, \dots, \lambda^n)$, where for each learning object λ^i belonging to the courseware, all the following conditions hold:

1. the background knowledge owned by the learner before benefiting from the lesson, ρ_{BK} , is at least λ_{BK}^1 , that is the background knowledge required by λ^1 , the first Learning Object of the sequence. In order to learn from a sequence of learning objects (LOs), the user must have at least the knowledge required to learn from the starting LOs.
2. after using λ^{i-1} , the user has a background knowledge which is at least λ_{BK}^i , *i.e.*, the one required by the *i*-th Learning Object. While benefiting of the composite LOs, the user acquires new knowledge which becomes part of her/his background. Such an *updated* background knowledge must satisfy the λ^i requirements.

Now the question is: "What is the background knowledge of the user after she/he benefits from the (*i*-1)-th learning object"?

The background knowledge of the learner before the fruition of λ^i is the conjunction of all the knowledge provided by λ_D^j , with $j < i$, and the initial background knowledge ρ_{BK} .

Indicating with BK_i the background knowledge before using λ^i , using the DL syntax, the following relation ensues:

$$BK_i = \rho_{BK} \sqcap \lambda_D^1 \sqcap \lambda_D^2 \sqcap \dots \sqcap \lambda_D^{i-1}$$

We can now define formally a **courseware flow**.

Definition 6. A **courseware flow** with respect to some initial background knowledge ρ_{BK} is a finite sequence of learning objects $\overline{\Lambda}(\rho_{BK}) = (\lambda^1, \lambda^2, \dots, \lambda^i, \dots, \lambda^n)$ with $i = 1..n$, where for each $\lambda^i \in \overline{\Lambda}(\rho_{BK})$ all the following conditions hold:

1. $\rho_{BK} \sqsubseteq \lambda_{BK}^1$.
2. $BK_i \sqsubseteq \lambda_{BK}^i$.

– We indicate with $\mathcal{D}_{\overline{\Lambda}}$, the set of learning objects descriptions in $\overline{\Lambda}(\rho_{BK})$. $\mathcal{D}_{\overline{\Lambda}} = \{\lambda_D^i | \lambda^i \in \overline{\Lambda}(\rho_{BK})\}$.

Based on the previous definition of **courseware flow**, it possible to define a **composite courseware** with respect to a request ρ .

Definition 7. Let $\mathcal{R} = \{\langle \lambda_D^i, \lambda_{BK}^i \rangle\}$, with $i=1..k$, be a set of learning objects λ^i , and $\langle \rho_D, \rho_{BK} \rangle$ be a request for a courseware, such that λ_D^i , λ_{BK}^i , ρ_D and ρ_{BK} are modeled as concept descriptions in a DL w.r.t. an ontology $\mathcal{T}$.

A **composite courseware** for $\rho = \langle \rho_D, \rho_{BK} \rangle$ with respect to $\mathcal{R}$, denoted $\Lambda(\rho, \mathcal{R})$, is a courseware flow such that for each λ_j in the courseware flow, $\mathcal{D}_{\Lambda} = \{\lambda_D^j | \lambda^j \in \Lambda(\rho, \mathcal{R})\}$, covers ρ_D .

Within a resource retrieval scenario, a composite courseware is a sequence of learning objects such that both the following conditions hold: it can be started using some background knowledge the requester owns (ρ_{BK}) and the provided composite courseware covers the user request description (ρ_D).

4.2 Automated Composite Courseware Generation

We adapt now the previously introduced *GREEDYsolveCCoP* to cope with background knowledge and present an algorithm to automatically compute a composite courseware.

For such purpose we need to define an *usable learning object* and an *usable set*.

Definition 8. Given a courseware flow $\bar{A}(\rho_{BK}) = (\lambda^1, \lambda^2, \dots, \lambda^n)$, we say that a learning object is a **usable learning object** λ_u for $\bar{A}(\rho_{BK})$ if and only if

1. $\lambda_u \notin \bar{A}(\rho_{BK})$.
2. $\bar{A}(\rho_{BK}) = (\lambda^1, \lambda^2, \dots, \lambda^n, \lambda_u)$ is a courseware flow.

A **usable learning object** λ_u for $\bar{A}(\rho_{BK})$ is a learning object which can be used after the user benefits from $\bar{A}(\rho_{BK})$, i.e., its required background knowledge is provided by $\bar{A}(\rho_{BK})$.

Actually, given a courseware flow, several usable learning objects exist.

Definition 9. Given a courseware flow $\bar{A}(\rho_{BK})$ and a set of learning objects $\mathcal{R} = \{\lambda^i\}$ we call **usable set** for $\bar{A}(\rho_{BK})$, the set of all the $\lambda^i \in \mathcal{R}$ such that λ^i is a usable learning object for $\bar{A}(\rho_{BK})$. $\mathcal{U}_{\bar{A}(\rho_{BK})} = \{\lambda_u^i | \lambda_u^i \text{ is a usable learning object for } \bar{A}(\rho_{BK})\}$

The *usable set* is hence the set of all the learning objects that can be used after the user benefits from a courseware flow.

Algorithm *teacher*($\mathcal{R}, \langle \rho_D, \rho_{BK} \rangle, T$)

input a set of learning objects $\mathcal{R} = \{\lambda^i = \langle \lambda_D^i, \lambda_{BK}^i \rangle\}$, a request $\rho = \langle \rho_D, \rho_{BK} \rangle$, where $\lambda_D^i, \lambda_{BK}^i, \rho_D$ and ρ_{BK} are satisfiable in T

output $\langle A, H \rangle$

begin algorithm

$A(\rho, \mathcal{R}) = \emptyset$;

$\rho_{D_{uncovered}} = \rho_D$;

$H_{min} = \rho_D$;

do

compute $\mathcal{U}_{A(\rho, \mathcal{R})}$;

$\lambda_{D_{min}} = \top$;

for each $\lambda^i \in \mathcal{U}_{A(\rho, \mathcal{R})}$

if $\mathcal{D}_{A(\rho, \mathcal{R})} \cup \{\lambda_D^i\}$ covers $\rho_{D_{uncovered}}$ **then**

$H = \text{solveCAP}(\langle \mathcal{L}, \lambda_D^i, \rho_{D_{uncovered}}, T \rangle)$;

if $H \prec H_{min}$ **then**

$\lambda_{D_{min}} = \lambda_D^i$;

$H_{min} = H$;

end if

end if

end for each

if $\lambda_{D_{min}} \neq \top$ **then**

$\mathcal{R} = \mathcal{R} \setminus \{\lambda^i\}$;

$A(\rho, \mathcal{R}) = (A(\rho, \mathcal{R}), \lambda^i)$;

$\rho_{D_{uncovered}} = H_{min}$;

end if

while ($\rho_{D_{min}} \neq \top$);

return $\langle A(\rho, \mathcal{R}), \rho_{D_{uncovered}} \rangle$;

end algorithm

The algorithm returns the composite courseware $A(\rho, \mathcal{R})$ and the uncovered part, $\rho_{D_{uncovered}}$, of the request description ρ_D . The main difference between *GREEDYsolveCCoP* and *teacher* is that the learning objects to be added to the lesson flow are searched for only within the current *usable learning objects*.

5 Example

In this section we show, with the aid of an example, the behavior of *teacher* with respect to a scenario related to Computer Science teaching. In the example we will refer to the toy ontology in Figure 1 in order to model ρ_D , ρ_{BK} , λ_D and λ_{BK} . We refer to a

```

WYSIWYGtool  $\sqsubseteq$  Tool
IDE  $\sqsubseteq \exists \text{develTool} \sqcap \forall \text{develTool.WYSIWYGtool}$ 
OOL  $\sqsubseteq$  ProgrammingLanguage
ProceduralLanguage  $\sqsubseteq$  ProgrammingLanguage
OOL  $\sqsubseteq \neg \text{ProceduralLanguage}$ 
Java  $\sqsubseteq$  OOL
C++  $\sqsubseteq$  OOL
Java  $\sqsubseteq \neg \text{C++}$ 
OOP  $\equiv \exists \text{language} \sqcap \forall \text{language.OOL}$ 
WebService  $\sqsubseteq$  DistributedTechnology
CORBA  $\sqsubseteq$  DistributedTechnology
WebService  $\sqsubseteq \neg \text{CORBA}$ 

```

Fig. 1. The ontology used as reference in the example

student wishing to learn about Java programming language with the aim of web service developing on a Unix platform, if specified. She/he has not any knowledge in that field.

```

 $\rho_D = \exists \text{language} \sqcap \forall \text{language.Java} \sqcap \exists \text{allowedTechnologies} \sqcap$ 
 $\forall \text{allowedTechnologies.WebService} \sqcap \forall \text{OS.Unix}$ 
 $\rho_{BK} = \top$ 

```

The repository contains six learning objects related to the Computer Science domain. $\mathcal{R} = \{\lambda^a, \lambda^b, \lambda^c, \lambda^d, \lambda^e, \lambda^f\}$.

λ^a relates to Object Oriented Languages. It is addressed to beginners.

$\lambda_D^a = \exists \text{language} \sqcap \forall \text{language.OOL}$

$\lambda_{BK}^a = \top$

λ^b relates to Java language programming with the aid of an Integrated Development Tool. It is addressed to Object Oriented programmers.

$\lambda_D^b = \exists \text{language} \sqcap \forall \text{language.Java} \sqcap \text{IDE}$

$\lambda_{BK}^b = \text{OOP}$

λ^c relates to C++ Languages programming. It is addressed to Object Oriented programmers.

$$\lambda_D^c = \exists \text{language} \sqcap \forall \text{language.C++}$$

$$\lambda_{BK}^c = \text{OOP}$$

λ^d relates to the CORBA technology. It is addressed to Object Oriented programmers.

$$\lambda_D^d = \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.CORBA}$$

$$\lambda_{BK}^d = \text{OOP}$$

λ^e relates to Web Services development with the aid of a WYSIWYG tool. It is addressed to Java programmers.

$$\lambda_D^e = \exists \text{develTool} \sqcap \forall \text{develTool.WYSIWYGtool} \sqcap \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService}$$

$$\lambda_{BK}^e = \exists \text{language} \sqcap \forall \text{language.Java}$$

λ^f is an introduction to distributed programming for beginners.

$$\lambda_D^f = \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.DistributedTechnology}$$

$$\lambda_{BK}^f = \top$$

The first step in order to compute $\Lambda(\rho, \mathcal{R})$ is to identify the initial $\mathcal{U}_{\Lambda(\rho, \mathcal{R})}$ with respect to an empty courseware flow $\Lambda(\rho, \mathcal{R}) = \emptyset$ ($\mathcal{U}_0$ for short). As $\rho_{BK} = \top$ then

$$\mathcal{U}_0 = \{\lambda^a, \lambda^f\}$$

At this initial step $\rho_{D_{uncovered}} = \rho_D$ then:

$$\begin{aligned} H_{\lambda^a} &= \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService} \sqcap \\ &\forall \text{language.Java} \sqcap \forall \text{OS.Unix} \\ |H_{\lambda^a}| &= 5 \end{aligned}$$

$$\begin{aligned} H_{\lambda^f} &= \exists \text{language} \sqcap \forall \text{allowedTechnologies.WebService} \sqcap \\ &\forall \text{language.Java} \sqcap \forall \text{OS.Unix} \\ |H_{\lambda^f}| &= 6 \end{aligned}$$

$\Lambda(\rho, \mathcal{R})$, $\rho_{D_{uncovered}}$ and BK_1 are now respectively:

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a)$
- $\rho_{D_{uncovered}} = \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService} \sqcap \forall \text{language.Java} \sqcap \forall \text{OS.Unix}$
- $BK_1 = \exists \text{language} \sqcap \forall \text{language.OOL}$

With respect to BK_1 , the new usable set $\mathcal{U}_1$ is:

$$\mathcal{U}_1 = \{\lambda^b, \lambda^c, \lambda^d, \lambda^f\}$$

With respect to $\mathcal{U}_1$ we have the following values:

$$\begin{aligned} H_{\lambda^b} &= \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService} \sqcap \\ &\forall \text{OS.Unix} \\ |H_{\lambda^b}| &= 4 \end{aligned}$$

$$H_{\lambda^c} = \text{NOT COMPUTED}$$

$$|H_{\lambda^c}| = \text{NOT COMPUTED}$$

$$H_{\lambda^d} = \text{NOT COMPUTED}$$

$$|H_{\lambda^d}| = \text{NOT COMPUTED}$$

$$H_{\lambda^f} = \forall \text{allowedTechnologies.WebService} \sqcap \forall \text{language.Java} \sqcap \forall \text{OS.Unix}$$

$$|H_{\lambda^f}| = 5$$

Both H_{λ^c} and H_{λ^d} are not computed because the information provided by λ^c and λ^d is not compatible with the uncovered one and this situation is not allowed by the Concept Abduction definition. Respectively, λ^c offers knowledge on C++ and the user is looking for Java while λ^d is about CORBA while the user is looking for WebService. The updated $\Lambda(\rho, \mathcal{R})$, $\rho_{D_{\text{uncovered}}}$ and $\mathcal{BK}_2$ are

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a, \lambda^b)$
- $\rho_{D_{\text{uncovered}}} = \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService} \sqcap \forall \text{OS.Unix}$
- $\mathcal{BK}_2 = \exists \text{language} \sqcap \forall \text{language.Java} \sqcap \text{IDE}$

For the next step the new usable set is:

$$\mathcal{U}_2 = \{\lambda^c, \lambda^d, \lambda^e, \lambda^f\}$$

Now, with respect to $\mathcal{U}_2$ we have :

$$H_{\lambda^c} = \text{NOT COMPUTED}$$

$$|H_{\lambda^c}| = \text{NOT COMPUTED}$$

$$H_{\lambda^d} = \text{NOT COMPUTED}$$

$$|H_{\lambda^d}| = \text{NOT COMPUTED}$$

$$H_{\lambda^e} = \forall \text{OS.Unix}$$

$$|H_{\lambda^e}| = 1$$

$$H_{\lambda^f} = \forall \text{allowedTechnologies.WebService} \sqcap \forall \text{OS.Unix}$$

$$|H_{\lambda^f}| = 2$$

Again H_{λ^c} and H_{λ^d} are not computed, but this time the reason is related to the definition of Concept Covering. In fact the description of both λ^c and λ^d is not compatible with the one belonging to the conjunction of λ^a and λ^b .

- $\Lambda(\rho, \mathcal{R}) = (\lambda^a, \lambda^b, \lambda^e)$
- $\rho_{D_{\text{uncovered}}} = \forall \text{OS.Unix}$
- $\mathcal{BK}_2 = \exists \text{language} \sqcap \forall \text{language.Java} \sqcap \text{IDE} \sqcap \exists \text{allowedTechnologies} \sqcap \forall \text{allowedTechnologies.WebService}$

It is easy to show that, at this point $teacher(\mathcal{R}, \langle \rho_D, \rho_{BK} \rangle, T)$ stops and it returns:

$$\langle A, H \rangle = \langle (\lambda^a, \lambda^b, \lambda^e), \forall OS.Unix \rangle$$

The proposed courseware, with respect to the available λ^i in the repository, for the request ρ is then $(\lambda^a, \lambda^b, \lambda^e)$, but ρ is not completely satisfied by the returned A because nothing is specified about the platform that will be used ($\forall OS.Unix$).

A prototype system integrated in the MAMAS framework (<http://193.204.59.227:8080/MAMAS-devel>) has been developed implementing *teacher* algorithm and exploiting standard (Semantic) Web technologies. The message exchanging is performed using SOAP messages. The semantic information in the body of such messages is formatted using OWL as explained in Section 2.

6 Conclusion and Future Work

In this work we proposed a tractable greedy algorithm to perform an automated courseware composition compliant with the standard Semantic Web technologies and exploiting standard and novel non-standard inference services for Description Logics. We presented motivations and examples for the approach.

We showed how a semantic specification, formatted in OWL-DL, of the learning objects (LOs) can be used both to retrieve from a repository LOs satisfying a user request and to compose such discovered LOs in a courseware.

The proposed approach also copes with non-exact solutions to the courseware composition. That is, if it is not possible to compose, using available LOs, a courseware which completely satisfies the user request, approximate solutions are proposed endowed with an explanation for the non-exact match.

Currently we are developing ontologies related to different tasks, in order to perform further experiments on real scenarios. Under development is also the integration of the implemented prototype with the SCORM specifications.

References

1. *IEEE Standard for Learning Object Metadata*, std 1484.12.1-2002 edition, 2002.
2. *IEEE standard for learning technology-learning technology systems architecture (LTSA)*, std 1484.1-2003 edition, 2003.
3. Advanced Distributed Systems (ADL) Lab, Sharable Content Object Reference Model (SCORM). <http://www.adlnet.org/index.cfm?fuseaction=scormabt>.
4. K. Ajami. Specifying and implementing interoperable and reusable learning objects: one step beyond. In *Proc. of Intl. Conf. on Information and Communication Technologies: From Theory to Applications*, pages 111–112, 2004.
5. F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
6. Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI2003)*, pages 319–324, 2003.

7. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In Dieter Hutter and Werner Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.
8. N. Bennacer, Y. Bourda, and B. Doan. Formalizing for querying learning objects using OWL. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 321–325. IEEE, 2004.
9. A.S. Cabezuelo and J.M.D. Beardo. Towards a model of quality for learning objects. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 822–825. IEEE, 2004.
10. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.
11. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The Massachusetts Institute of Technology, 1990.
12. DAML+OIL Specifications. www.daml.org/2001/03/daml+oil-index.html, 2001.
13. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 337–342, Acapulco, Mexico, August 9–15 2003. Morgan Kaufmann, Los Altos.
14. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. A system for principled Matchmaking in an electronic marketplace. In *Proc. International World Wide Web Conference (WWW '03)*, pages 321–330, Budapest, Hungary, May 20–24 2003. ACM, New York.
15. Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Student tracking and personalization: Personalization in distributed e-learning environments. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
16. Dublin Core Metadata Element Set, Version 1.1: Reference Description . <http://dublincore.org/documents/1999/07/02/dces/>.
17. D. Gasevic, J. Jovanovic, and V. Devedzic. Enhancing learning object content on the semantic web. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 714–716. IEEE, 2004.
18. Mohand-Sad Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. Computing Concept Covers: a Preliminary Report. In *Proc. of the 15th Intl. Workshop on Description Logics (DL'02)*, volume 53 of *CEUR Workshop Proceedings*, 2002.
19. Marek Hatala, Griff Richards, Timmy Eap, and Jordan Willms. Sharing educational resources: The interoperability of learning object repositories and services: standards, implementations and lessons learned. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
20. IMS, Learning Resource Meta-data Best Practices and Implementation Guide Version 1.1 - Final Specification . <http://www.imsproject.org/metadata/mdbestv1p1.html>.
21. A. Ip, A. Young, and I. Morrison. Learning objects - Whose are they? In *Proc. of 15th Conf. of the National Advisory Committee on Computing Qualifications*, pages 315–320, 2002.
22. Sycara Katia, Paolucci Massimo, Ankolekar Anupriya, and Naveen Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1, December 2003.
23. D.L. McGuinness, R. Fikes, J. Hendler, and L.A. Stein. DAML+OIL: An Ontology Language for the Semantic Web . *IEEE Intelligent Systems*, 17(5):72–80, 2002.
24. OWL. www.w3.org/TR/owl-features/.

25. C. Pahl and R. Barrett. A web services architecture for learning object discovery and assembly. In *Proc. International World Wide Web Conference (WWW '04)*, 2004. Alternate track papers and posters.
26. S. Sanchez and M. Sicilia. On the semantics of aggregation and generalization in learning object contracts. In *Proc. of Intl. Conf. on Advanced Learning Technologies*, pages 425–429. IEEE, 2004.
27. G. Teege. Making the difference: A subtraction operation for description logics. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 540–550. MK, 1994.
28. The OWL Services Coalition . www.daml.org/services/owl-s/1.0/owl-s.html, 2004.
29. James Hendler Tim Berners-Lee and Ora Lassila. The semantic web. *Scientific American*, 248(4), 2001. (34-43).
30. G. Vossen and P. Jaeschke. Learning objects as a uniform foundation for e-learning platforms. In *Proc. of Intl. Symp. on Database Engineering and Applications Symposium*, pages 278–287, 2003.

Orchestration of Semantic Web Services for Large-Scale Document Annotation

Barry Norton, Sam Chapman, and Fabio Ciravegna

Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
{B.Norton, S.Chapman, F.Ciravegna}@dcs.shef.ac.uk

Abstract. Armadillo is a tool that provides automatic annotation for the Semantic Web using unannotated resources like the existing Web for *information harvesting*, that is: combining a *crawling* mechanism with an extensible architecture for *ontology population*. The latter is achieved via largely unsupervised machine learning, boot-strapped from oracles, such as web-site wrappers. It is backed up by ‘evidential reasoning’, which allows evidence to be gained from the redundancy in the Web as well as inaccuracies in information, also characteristic of today’s Web, to be circumvented. In this paper we sketch how the architecture of Armadillo has now been reinterpreted as workflow templates that compose semantic web services and show how the porting of Armadillo to new domains, and furthermore the application of new tools, has thus been simplified and benefits from semantic discovery and automatic orchestration.

1 Introduction

The vision of the Semantic Web (SW) is centred around sharing knowledge in order to acquire and reuse it [1]. Recently, it has become apparent that it is possible to share more than static knowledge, moving towards sharing operational and active knowledge, i.e. towards Semantic Web Services (*SWs*). In the future SW it will be possible to compose large distributed systems by composing existing SW Services. One example of large scale services necessitating service reuse are automatic annotation systems, helping harvesting knowledge from existing unannotated documents [6], [11]. Harvesting is guided by an ontology. Ontologies can be overlapping or evolving from common roots, and therefore - we claim - parts of existing harvesters should be reused also when ontologies are reused. As a simple example, many harvesters will share the need to recognise generic base types like people and organisations, and appropriate services are very likely to be existing on the Web. The more ontologies will be reused, the more the need of reusing harvesters will grow.

We claim also that a SWS-based architecture provides a clear place to separate (and apply) domain-dependent functionality as opposed to domain independent functionality. Also, such an architecture should provide a means to discover already existing implementations of functionalities for reuse.

Tailoring to a domain should therefore be realised by defining or reusing independent SWS's. All of the advantages of a distributed implementation will therefore hold, i.e.:

- the ability to build a new application, coordinating third-party services, without the need for major computing infrastructure;
- conversely, the ability to provide such services while keeping the implementation in-house for management and maintenance;
- the speed-up that can be achieved through parallelism.

This paper focuses on Armadillo [4], a system for the definition of ontology-specific harvesters for large repositories (e.g. the Web), and its organisation in terms of SWS's. Armadillo uses an ontology to define an annotation task. Annotation is defined as extraction and integration of information (harvesting). The system is based on the Information Food Chain metaphor [9], where information processing is defined as an ecosystem where basic search tools (herbivores) retrieve documents (raw matter), while information processors (carnivores) digest them to produce progressively more sophisticated information. The ecosystem in Armadillo is geared towards the production of knowledge and its implementation is based on integration of SWS. Practically, Armadillo annotates by extracting information from different sources (documents or repositories such as databases) and carrying out 'evidential reasoning' to validate the classifications of, and relations between, instances. This evidence is then integrated and the knowledge entered into a repository summarising the integrated knowledge.

The orchestration of SWS's is realised by providing workflows in BPEL [8], which allows us to express workflows where 'partners', *i.e.* workflow actors to be realised by web services, are parametrically typed but unbound to services. We therefore say that we define the Armadillo architecture as workflow templates where *service parameters* must be instantiated, *i.e.* 'filled in', for orchestration to effect a given task. We allow the process to be understood abstractly in terms of the OWL [15] concepts they deal with by describing the parts of this architecture as *semantic* web services, in OWL-S [7]. This furthermore allows *semantic* discovery to be engaged to help with finding the appropriate services to use to instantiate parameters; again we pragmatically decide to adapt the existing web services, *i.e.* UDDI¹, tools to semantic purposes in the spirit of [14]. Furthermore, BPEL allows us to represent graphically, rather than in code, the workflows and insofar as appropriate services can be found, a developer can apply Armadillo to a new annotation task with no coding at all. Even where coding is needed, the subtask needed is abstracted away from the Armadillo logic by service boundaries and the service produced is automatically described in OWL-S terms allowing it to be directly made available for future reuse.

In this paper we concentrate on the implementation via SWSs of the extraction and integration tasks, which are at the heart of the architecture. This paper is organised as follows: Section 2 describes the abstract harvesting strategy, Section 3 gives more details on the architecture and each subtask in detail, finally we make conclusions and sketch future work in Section 5.

¹ <http://uddi.org/>

2 Harvesting Strategy

Armadillo [4] annotates documents by harvesting knowledge from large repositories, i.e. by extracting information from different sources and finally integrating the retrieved knowledge into a repository. The repository can be used both to access the extracted information and to annotate the source(s) where the information was identified. Furthermore the information source(s) can be analysed to verify the correctness and the provenance of the information.

Unsupervised learning begins from seed data provided by a largely ‘infallible oracle’ (e.g. a list of relevant terms). Seed data are utilised by searching in the document repository for matching strings. If found, matching strings must be confirmed using some disambiguation or contextual strategies (e.g. local disambiguation as in SemTag [6] or multiple evidence for corroboration). Further annotations are identified by the process sketched in Figure 1, *e.g.* by adaptively learning from the context in which known entities were found. All new annotations must be confirmed, by the subtask called ‘evidential reasoning’, and the terms encapsulated by the annotations can be used to seed learning again. Finally the discovered knowledge is integrated (e.g. some of the new entities / facts are merged) and stored into a format for future use, typically a database.

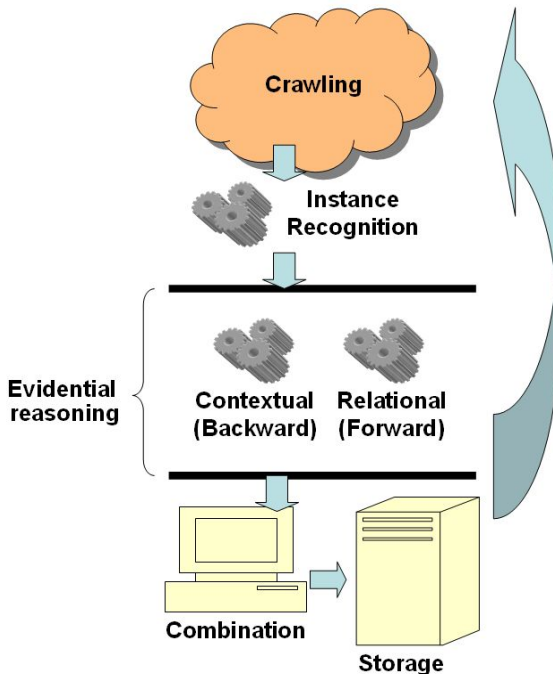


Fig. 1. Harvesting Strategy

3 Architecture

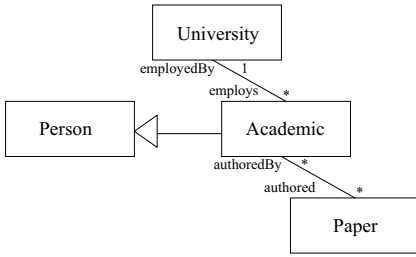
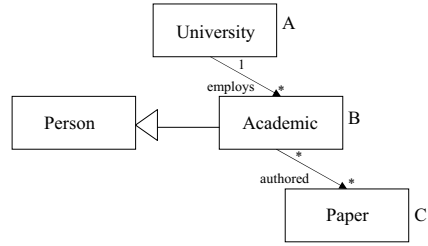
Armadillo employs the following techniques/technologies:

- *Crawling* to explore the repository in a efficient and exhaustive way.
- *Adaptive Information Extraction* from texts (IE): used for spotting information and to further learning new entities.
- Information Integration (II): used to (1) discover an initial set of annotations to be used to seed learning for IE and (2) to confirm the newly acquired (extracted) information, *e.g.* using multiple evidence from different sources and (3) to integrate knowledge *e.g.*, by merging entities.
- *Semantic Web Services*: the architecture is based on the concept of “services”. Each service is associated to some part of the ontology (*e.g.* a set of concepts and/or relations) and works in an independent way. Each service can use other services (including external ones) for performing some sub-tasks. This paper details the organisation of the SWS architecture and their orchestration.
- *RDF Repository*: where the extracted information is stored and the link with the pages is maintained.

The act of porting Armadillo to a new ontology population task begins by providing a domain-specific ontology. Population is performed by starting focusing on one concept and looking for its occurrences for it or for occurrences of its instances. These concepts tend to be characterised by unique identifiers such as proper names or rather unique descriptions² When key concepts have been identified, other entities with a minor degree of uniqueness can be identified by exploiting the context given by relations with already identified entities. For example dates tend to be too generic for unique identification in isolation. If “26 October 2004” is looked for as the date of *e.g.* a seminar, there is no way that all relevant occurrences of that date can be distinguished by other occurrences of the same date in other contexts. The context is used by the seminar instances previously identified to separate the relevant occurrences of the date from the irrelevant ones. Previously identified entities and their relations with respect to the looked for entity are used as a context. This means that the plan for ontology population is implemented as a directed search on the ontology graph: base (unique) entities are identified and the relations in the ontology are assigned a direction. The plan details the order in which the concepts and relations will be explored and therefore the ontology populated.

For each relation to be followed, we identify a concept *A* for which we will already have found instances and a concept *B* that we wish to populate by following the relation. We might also identify concepts *C* which ‘belong’ transitively, *i.e.* via further relations, to *B* so that related instances of such may also provide evidence for the instance of *B* discovered.

² Of course names can be ambiguous and descriptions not completely unique; what we mean here is that the concept looked for must have some degree of simplicity in identification).

**Fig. 2.** Example Ontology**Fig. 3.** Example with Ordering

Figures 2 and 3 illustrate the process in terms of a fragment of the example of populating an ontology for the UK academic domain used in [4] and [10]. We will use that example in the remainder of the paper. We first identify the ‘University’ concept as being one from which we can hope to obtain reliable instances from an oracle such as a list of universities and departments. Such list can be obtained for example by wrapping the Yahoo taxonomy or the RAE web pages³; university names tend to be unique terms.

We then decide that we should like to populate the ‘Academic’ concept, via the ‘employs/employedBy’ relationship. This concept is less unique than the university name (e.g. you can have many different people called “John Smith”) For this task, then, ‘University’ becomes concept *A* and we take the recognisable super-concept to ‘Academic’, ‘Person’ as concept *B*. We cannot choose directly the concept Academic because in a university there are many people that are not academic but that still are working for the university. We therefore need to discriminate a second time which recognised people are academics and which are not. Therefore we choose ‘Paper’ as an appropriate concept able to help discriminating (concept *C*) and exploit the relation ‘authored/authoredBy’ as a discriminator (in this small example the definition of academic is a person who writes papers). Papers titles tend to be largely unique. Figure 3 shows how we have added this information to the ontology fragment.

In the rest of the section we describe the details of the recognition of concept *B* using the context of *A*. As shown in Figure 1 it involves the following subtasks:

- Crawling;
- Instance Recognition;
- Evidential Reasoning;
- Combination and Storage.

Crawling will systematically retrieve documents associated with an instance of concept *A*. Instance Recognition will find candidate instances of *B* in the context of that original instance. Evidential Reasoning will (1) find support for the

³ <http://www.hero.ac.uk/rae/>

classification of the concept as B and (2) confirm the relation between A and B by looking for evidence, both within the document and outside, of the relation between the two entities. Finally, if the evidence support the initial hypothesis, the annotation is stored.

In the organisation of an SWS based architecture, A , B and C are variables ranging over concepts, as discussed above, as are Doc , $DocId$ to be tied to the documents to be analysed and $Evidence$, explained later. Moreover, additional *service parameters* must be similarly bound to create an executable process (dotted boxes in the figures below); their signatures may involve the concept variables so it is necessary to decide on this first. Having done so, the choice of services can be aided by *semantic discovery* [14]. We shall describe the choices in the remainder of the paper in terms of the following classes of service:

- *generic* services: wholly independent of domain and context;
- *context-dependent* services: where reuse depends on the context but not the domain (*e.g.* the type of documents or repository used);
- *domain-tailored* services: parameterised to be tailored to a given domain;
- *domain-trained* services: encapsulate machine-learning which can adapt semi-automatically to a given domain;
- *domain-specific* services: encapsulate techniques which are hard-coded for a given domain (but might still be re-used across applications in that domain).

We now go through each abstract subtask of the population task in the terms that these were set out in Figure 1.

3.1 Crawling

The general form of the ‘crawling’ task as a workflow template over its core services⁴ and service parameters is shown in Figure 4:

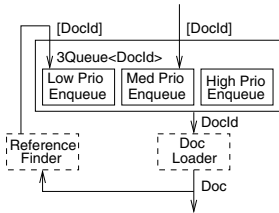


Fig. 4. Crawling Task

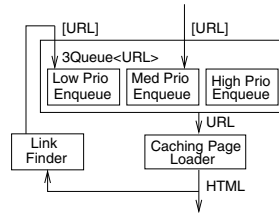


Fig. 5. Example *Crawling* Instantiation

⁴ The containing box *3Queue* means that the service, actually an instantiation in terms of the type of document identifier to be queued, consists of several operations. We omit the trigger (de-queueing) operation since it is not relevant at this level.

To reiterate, there are two levels at which this workflow is parameterised⁵: the *DocId* and *Doc* labels⁶ are type variables that must be instantiated at concrete types; the *DocLoader* and *ReferenceFinder* tasks are service variables to be instantiated with concrete services.

In these terms then, the crawling task becomes one of queueing up document references, in any form by which they can be given persistent identifiers, which are individually loaded and immediately inspected for references to related documents and also passed forward to the *instance recognition* task.

The reason that the original document list is enqueued at a medium priority level is that we will inspect first documents strongly related to the current one; as shown in Figure 14 these are fed back at a high priority level. Once a given document and strongly-related discoveries are exhausted we move on to the next from the original list. Only once this process has been completed for each member of the list do we move on to those only weakly related (there being only tentative reasons to presume those referred to in the document will relate strongly to the original instance of concept *A*). Note that the signature of the ‘Reference Finder’ service parameter (in functional terms $Doc \rightarrow [DocId]$) allows the possibility that some prioritisation may take place in the list returned, allowing for intelligent analysis.

Figure 5 shows how we could achieve a concrete instantiation of this subtask. First we choose instances for the type variables consistent with Web-oriented technology, i.e., in this example, $Doc = HTML$ and $DocId = URL$. We then choose *context-dependent* services that meet the resulting signatures, i.e. loading a page from its URL, with a ‘caching page loader’, and respectively finding list of URLs from a page with a ‘link finder’, itself an instantiation of a domain-tailored regular expression matching service but we do not show this decomposition.

3.2 Instance Recognition

As shown in Figure 6, the ‘instance recognition’ task is achieved foremost by the concurrent execution⁷ of some number⁸ of different ‘*B-Recogniser*’s, i.e. services each of which can extract from a document parameter potential occurrences of the concept *B*. We assume that the document parameter leading to the ‘split’ operation is broadcast to each service used in the instantiation of this template. Thereafter each of the lists of *candidate instances* returned by these services is

⁵ Actually three since angle brackets, for instance $3Queue\langle DocId \rangle$, mean instantiation of some generic (parameterised) service, so that $3Queue$ represents some generic queue service that stores instances of some type notated *DocId*, at three different priority levels, and supplies them to the consequent workflow one at a time.

⁶ We use square brackets to represent lists over the contained type, and so $[DocId]$ is the type of a list of elements of type *DocId*, and regular brackets to represent tuples.

⁷ We represent concurrent execution, and re-synchronisation on completion, by the solid bars, ‘split’ and ‘join’ respectively, in the diagram in the style of UML Activity Diagrams [2].

⁸ We represent multiple instantiation also in UML style with a multiple outline.

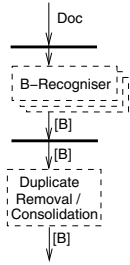


Fig. 6. Instance Recognition Task

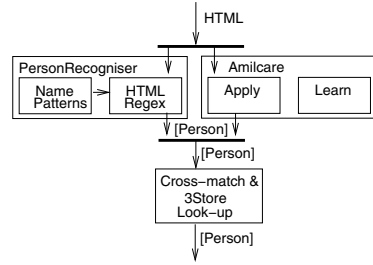


Fig. 7. Example Recognition Instantiation

concatenated to make the final list. This is then passed to a service parameter whose instance should be capable of removing internal duplicates, if multiple recognition strategies are employed, as well as carrying out a ‘pre-integration’ step. This latter step may, and usually will, involve checking in a repository to see whether the instance is already known. For simplicity we show a signature for this service that just involves some refinement of the list, $[B] \rightarrow [B]$. In fact we allow additional triples, based on the instance, to be introduced at this point so that, for instance where existing coincident repository instances are found that were introduced from some other source, we may continue to investigate them (rather than dropping them) but introducing a ‘sameAs’ relationship.

The role of B-Recogniser can be realised by both *domain-tailored*, and *domain-trained* services. This is illustrated in Figure 7 we see that both *domain-tailored* regular expression matching and an instance of the *domain-trained* IE system Amilcare [5] will be used side-by-side, Amilcare learning from the successfully validated instances produced by the former. Since the signature of this service parameter is so general, with the implementation details encapsulated behind the service boundaries, other Information Extraction tools can easily be employed.

The subsequent consolidation stage is typically domain-specific Information Integration but reuse can be made; for instance from the ‘similarity metrics’ library, *SimMetrics*, which we are developing as an open source project for string metrics and similarity-based integration, and recently released [3].

3.3 Evidential Reasoning

Having identified some consolidated list of candidate instances we then queued these to be validated both for proper classification and to verify and classify the implicit relation through which they were discovered. We call this task ‘evidential reasoning’ and its overall workflow is presented in Figure 8.

The candidate instances from the instance recognition task are first queued to be examined one by one. For each, some number of concurrent instances of reasoning tasks will be executed. Each will fit one of two general strategies described as follow.

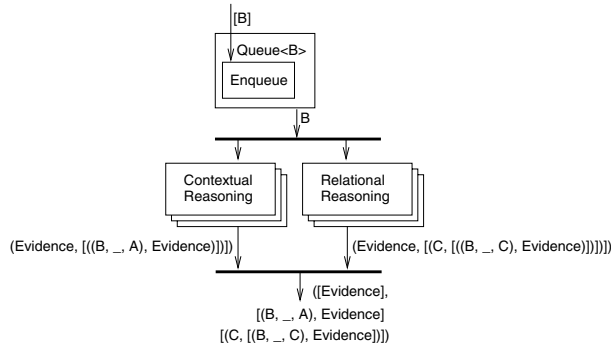


Fig. 8. Evidential Reasoning Task

Contextual Reasoning. considers each potential B instance in the context of the A instance via which it was discovered and attempts to be more specific about the relation between them than the implicit relation that instance recognition achieves. In the process more evidence for the classification of the instance being investigated may be found. As seen in Figure 9, two services will be used to find occurrences of the B instance in general, and co-located with the A instance, respectively. A third service then produces a list of potential triples relating these instances, with evidence supporting the relation.

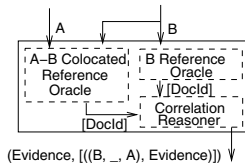


Fig. 9. Contextual Reasoning Task

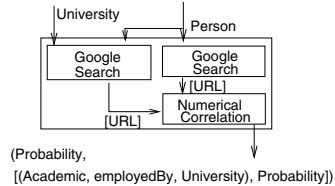


Fig. 10. Example Contextual Instantiation

Figure 10 shows a simple instance of this strategy where we ‘promote’ the candidate instance to being an academic employed by the university based on co-located references on the web, obtained by a Google wrapper which is domain-independent, where a simple probability is based on correlation.

Relational Reasoning. provides evidence for the candidate B instance being correctly classified as such, based on other relations an oracle may find. As such, for each instantiation within this class, two service parameters must be provided: as shown in Figure 11, one finds related new potential instances, another encodes this alongside some kind of evidence.

In this subtask, as shown in Figure 12, we may apply *domain-tailored* or *domain-specific* technologies such as gazetteers — we use the example of a gazetteer of people’s forenames — and site wrappers — we use the example

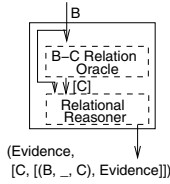


Fig. 11. *Relational Reasoning Task*

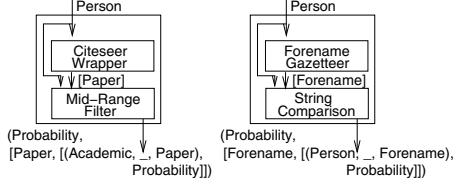


Fig. 12. *Example Relational Instantiations*

of the DBLP portal⁹ wrapped as a semantic web service and providing candidate instances for the ‘Paper’ concept and the ‘authored’ relation. Such results will be cached and bootstrap a separate workflow on this relation.

As part of the relational reasoning subtask, We may also apply *domain-trained* relation extraction, as we are developing in the tool *T-Rex*. As in the instance recognition subtask we will bootstrap learning based on oracles, for instance in this case on the DBLP results.

3.4 Combination and Storage

The essence of the combination task is to remove duplicates from the candidate instance and its candidate relations and to combine the evidence for these. This logic is encapsulated and instantiates the ‘Combination Logic’ service parameter and is domain-specific, but may decompose into some reused generic services; in particular statistical functions.

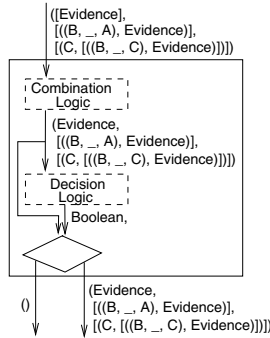


Fig. 13. *Combination Task*

On a basic level we must reduce: a list of evidence for the correct identification of an instance, $[Evidence]$, into a single combined value; all identified $(B, -, A)$ triples, *i.e.* relations back to the original contextual instance, and the separate *Evidence* values that have been found for each relation, into a non-repeating list;

⁹ <http://dblp.uni-trier.de/>

all identified $(B, -, C)$ triples, *i.e.* different relations to different new candidate instances C , and the separate *Evidence* values that have been found for each, into another non-repeating list.

The ‘Decision Logic’ task then decides which of these candidates has sufficient evidence for storage. The diamond style of the final component service in the combination task implies non-determinism: either an untyped value¹⁰ is returned (in case the evidence is insufficient) or the supported classification of the candidate and its relations are returned. Since the ‘Repository’ service parameter, implementing the storage task as seen in Figure 14, also returns an untyped value, the two possible threads of control are merged to pass such a value back to the ‘Trigger’ operation of the ‘B-Queue’ service.

4 Orchestration

The complete workflow template that orchestrates the tasks discussed in the previous section is shown in detail in Figure 14. As stated above, the orchestration is currently realised by encoding the workflow in BPEL [8]. As well as the pragmatic advantages put forward in [12], we state specifically the advantages of having an editor in the Eclipse environment¹¹ [13], where integration is possible with other plug-ins and tools described later, and having debugging and process monitoring tools¹².

Deficiencies, however, in the BPEL approach mean that it is not possible to directly encode the workflow shown in the simple dataflow manner represented in Figure 14. In particular there are two features that are incompatible with the ‘Flow’ construct: the looping behaviour shown on the far right hand side, and the non-determinism associated with choices we have illustrated via the diamond boxes. To explain what this is intended to represent, we pick up explaining the thread of control as it finished at the end of Section 3.4. Whichever service, *i.e.* instantiating ‘Combination’ or ‘Repository’, returns a value to the ‘Trigger’ operation of the ‘Queue(B)’ service, there are two possibilities, decided between only at this point in the execution: if there are remaining candidate B-instances to be considered then the evidential reasoning task is re-entered; if not another empty value triggers the document queue until this is exhausted and the flow is complete.

Since we can neither ‘cross’ the lines of a choice process, and furthermore are explicitly disallowed from having loops in a dataflow, we are forced into implementing subtasks discussed in Section 3 as different services and then mixing the imperative, explicit looping, and dataflow styles, as is typical of BPEL. Similarly we are forced into imperative style in order to mediate between tasks in the workflow, both in the one-to-one fashion normally discussed, and

¹⁰ The *empty type* is represented by $()$ as in languages like Haskell.

¹¹ <http://www.eclipse.org>

¹² Like those provided by the Collaxa tool, now part of Oracle Application Server: <http://www.oracle.com/technology/products/ias/bpel/index.html>

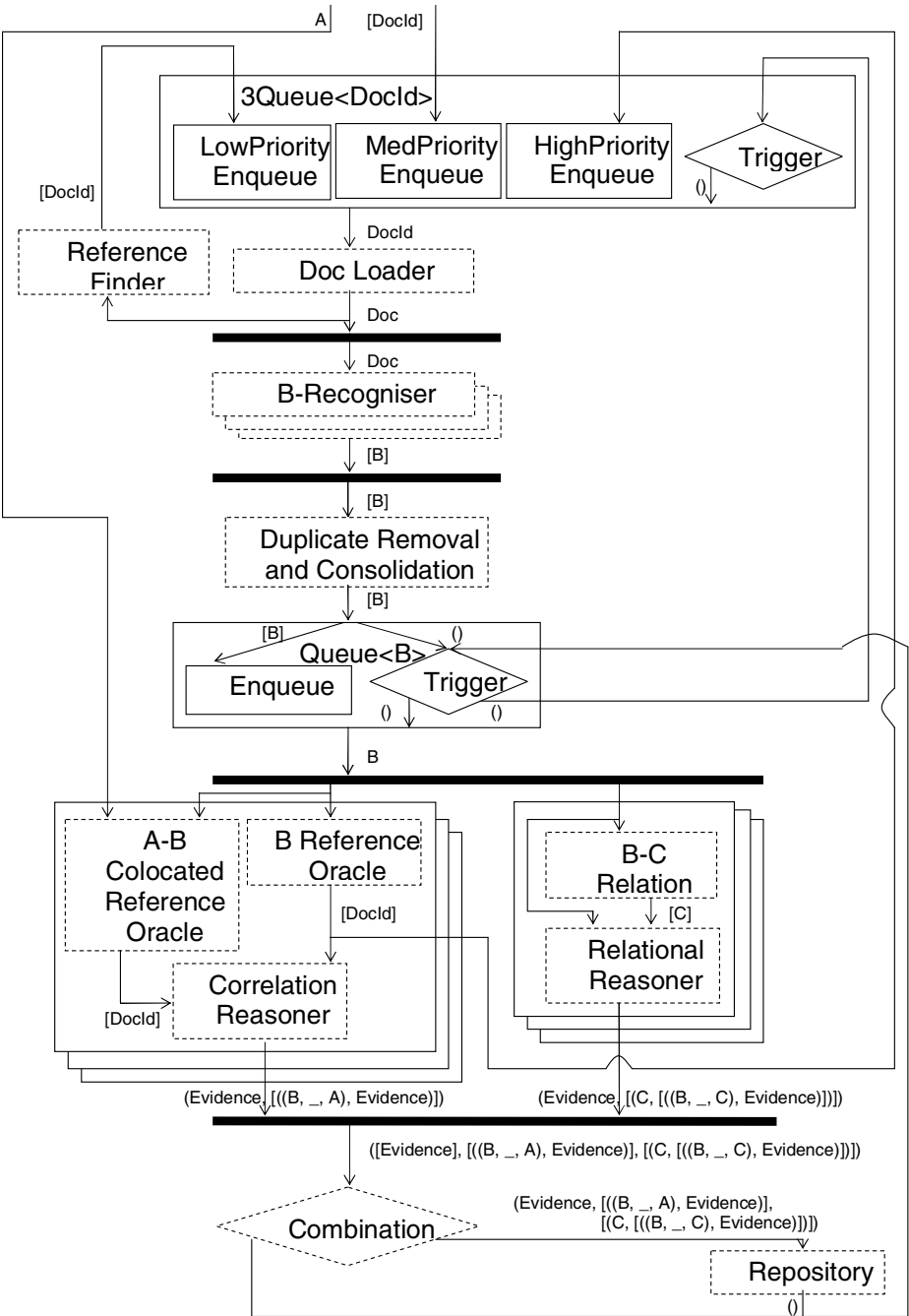


Fig. 14. Architecture in Detail

the many-to-one fashion associated with the ‘join’ operators that resynchronise concurrent activities.

We note that OWL-S [7] already allows, in the abstract, the binding of different languages to all these points in the workflows described therein. Furthermore more algebraic mixing of operators is specified as being provided. Since few implementations are currently available of OWL-S, however, our decision has been to use the Armadillo architecture as a motivation for, and a test of, a new OWL-S implementation where the language used at these points will be more declarative. For example this should allow a polymorphic functional-style ‘list concatenation’ function to be bound to the first join operator and thus not have to be tailored towards new instantiations for the parameter ‘B’. Furthermore we could allow the user more choice at this stage, for instance allowing a polymorphic ‘list cons’ function to be bound so that a different consolidation service, which needs to know which candidates came from which B-Recogniser could receive $[[B]]$.

5 Conclusions

We have described the way SWS’s are orchestrated to harvest information from the Web, and other corpora, to provide annotations for the Semantic Web. The architecture presented is based on workflow and follows an IE-oriented strategy. Initial approximations to both classification and implicit relation extraction are followed by evidential reasoning based on both context and further relations. In this way a wide variety of semantic web services may be accommodated and porting is eased since, in many cases, users can avoid coding altogether, merely using the workflow templates to guide semantic discovery and composition.

The architecture is currently implemented in BPEL in terms of services grounded in SOAP, but described both syntactically in WSDL and semantically in OWL-S; the generation of both is bootstrapped by an automatic translation. Our development process is wholly Eclipse-based with JDT support for Java coding alongside SWeDE¹³ support for OWL(-S) editing, Oracle support for BPEL editing and Ant¹⁴-hosted tasks for translation, building and deployment.

In future we would like to integrate more ‘semantic’ forms of discovery to help the process of constructing a concrete workflow, reusing from tools such as [14] as these become available. In the longer term we have hopes for the use of a full OWL-S or WSMX based orchestration solution and our own work towards the former is being carried out as an open source project¹⁵, as well as the accompanying editor¹⁶.

¹³ <http://owl-eclipse.projects.semwebcentral.org/>

¹⁴ <http://ant.apache.org/>

¹⁵ <http://savannah.nongnu.org/projects/CaSheW-s-Engine>

¹⁶ <http://savannah.nongnu.org/projects/CaSheW-s-Editor>

Using a SWS-based architecture for Armadillo provides many benefits associated with service-oriented architectures, such as speed-up from concurrency and distribution, an automatic means for reuse of any code created during an instantiation, and the ability to provide services remotely to users with little infrastructure. A number of instantiations of this architecture have already been carried out and are currently being evaluated, in particular concentrating on efficiency and scalability compared to the existing Armadillo software.

Acknowledgements

This work was carried out within the AKT project (<http://www.aktors.org>), sponsored by the UK Engineering and Physical Sciences Research Council (grant GR/N15764/01), and the Dot.Kom project, sponsored by the EU IST as part of Framework V (grant IST-2001-34038).

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, o284(5):35–43, 2001.
2. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language user guide*. Addison Wesley Longman, 1999.
3. Sam Chapman. SimMetrics. <http://sourceforge.net/projects/simmetrics/>.
4. Fabio Ciravegna, Sam Chapman, Alexiei Dingli, and Yorick Wilks. Learning to harvest information for the semantic web. In *ESWS*, pages 312–326, 2004.
5. Fabio Ciravegna and Yorick Wilks. *Annotation for the Semantic Web*. Series Frontiers in Artificial Intelligence and Applications. IOS Press, 2003.
6. Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, R. Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A. Tomlin, and Jason Y. Zien. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 178–186. ACM Press, 2003.
7. Anupriya Ankolekar et al. DAML-S: Web service description for the semantic web. In *Proc. 1st International Semantic Web Conference (ISWC)*, 2002.
8. IBM et al. Business process execution language for web services version 1.1. <http://www-128.ibm.com/developerworks/library/ws-bpel>, 2003.
9. Oren Etzioni. Moving up the information food chain: Deploying softbots on the world wide web. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1322–1326, Menlo Park, 4–8 1996. AAAI Press / MIT Press.
10. Hugh Glaser, Harith Alani, Les Carr, Sam Chapman, Fabio Ciravegna, Alexiei Dingli, Nicholas Gibbins, Stephen Harris, Monica M. C. Schraefel, and Nigel Shadbolt. CS AKTiveSpace: Building a semantic web application. In *Proc. 1st European Semantic Web Symposium*, pages 417–432, 2004.
11. P. Kogut and W. Holmes. AeroDAML: Applying information extraction to generate DAML annotations from web pages. In *First International Conference on Knowledge Capture (K-CAP 2001)*., 2001.

12. David J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The bottom-up approach to web service interoperation. In *Proc. 2nd Intl. Semantic Web Conference (ISWC2003)*, 2003.
13. Barry Norton. Eclipse as a development platform for semantic web services. Eclipse Technology Exchange (eTX04), 18th European Conference on Object-Oriented Programming (ECOOP-2004), 2004. <http://www.dcs.shef.ac.uk/~barry/CASheWs/Norton04.pdf>.
14. Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding OWL-S to UDDI: implementation and throughput. In *Proc. 1st Intl. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, pages 6–9, 2004.
15. W3C. OWL web ontology language overview. <http://www.w3c.org/TR/owlfeatures>, 2004.

Monitoring Research Collaborations Using Semantic Web Technologies

Harith Alani, Nicholas Gibbins, Hugh Glaser, Stephen Harris, and Nigel Shadbolt

Advanced Knowledge Technologies (AKT), School of Electronics and Computer Science,
University of Southampton, Southampton SO17 1BJ, UK
{ha, nmg, hg, swh, nrs}@ecs.soton.ac.uk

Abstract. In the current research environment, funding agencies are increasingly required to demonstrate that the projects they fund represent value for money. When funds are disbursed in a speculative manner, in order to stimulate interdisciplinary collaboration, the determination of value for money relies on evidence that shows the generation of new collaborations. This paper summarises the work we carried out on behalf of the Engineering and Physical Sciences Research Council (EPSRC), in which we have implemented a set of applications to enable the research council to examine the existence and nature of collaborations between researchers. We have used Semantic Web technologies to construct a flexible application framework to provide multiple complementary visualisations of the data, while separating the issues of knowledge acquisition and curation from the more user-centric interface requirements.

1 Introduction

Organisations have the need and responsibility to review and analyse their activities. The need arises not only from internal review procedures, but also from external agencies (eg government) that are trying to ensure value for money. Current and forthcoming requirements from government are imposing increasing obligations in this respect on the Research Councils (RCs), with a particular focus on research outputs and citations.

For example, one of the questions that arises is the extent to which different groups, people and programmes collaborate with each other. This is a very complex issue, as it is not even necessarily clear what it means to collaborate, and certainly no general agreement of what would be evidence of collaboration. We should expect that good analysis would not only provide qualitative and quantitative data on collaboration, but also allow users to think about and explore the nature of collaboration itself. Such analysis is challenging, in particular when it requires analysis of data from a wide variety of sources, many of which are outside the direct control of the organisation.

We can roughly define two main stages in this work. The first stage is to integrate the distributed sources of data and store it in a format suitable for further use and analysis.

Integration of databases raises several well known challenges, such as resolving the conceptual differences between database schemas, identifying data duplications and inconsistencies, etc [4]. Ontologies have been widely proposed as a major role player in information integration [7][14][15]. They provide the mechanisms to establish a

common understanding of heterogeneous data domains and help to bridge multiple data source schemas. In the case of RCs, the ontology would have concepts such as person, funding agency, grant value, etc. Data could then be gathered against those classes, and represented in a suitable form, such as RDF. It can then be kept in a suitable Knowledge-Base (KB) (3Store in our case, [9]), from which it can be queried in various fashions by other tools and applications.

The second stage of this work starts once the KB is set and populated with all the required data. This stage is concerned with building the suite of tools and applications needed to provide the type of data management and analysis of research collaborations required by the RCs. This involves the implementation of services for browsing the collecting data, and visualising research activities in interactive ways.

The following sections discuss the two stages described above. Section 2 describes the data gathering process. The architecture of the system is explained in section 3. Section 4 describes all the tools and applications we built for browsing, managing, and analysing the data. A discussion of the main issues and challenges that we came across during this work is given in section 5. Finally, section 6 concludes this work, and any major work to be done in the near future is highlighted in section 7.

2 Data Gathering

In the United Kingdom, there are a number of agencies and stakeholders who contribute towards the funding of research. The key initial activity in constructing a system which can provide an overview of this research is the gathering of appropriate data from the relevant participants. A production system that attempts to provide a full overview of the sector would need to embrace all required sources, including all RCs, and possibly publication data, academic staff data, and other funding agencies.

The sources from which data is gathered are heterogeneous, as would be expected from a group of organisations with distinct requirements and objectives, at least in terms of their research programmes. The integration of this data in a suitable manner for common browsers and visualisers requires that the data be mediated and cast into a common form. We use an ontology as the mediating construct, such that each of the heterogeneous sources is translated into the ontology.

The first requirement is to define the ontology. For this study we used an existing ontology, which was constructed in the AKT Project¹, and defined using OWL. The AKT ontology² represents general information about the academic research environment.

For the purposes of this study, we took data on projects and grants from RCs in three domains: engineering and physical sciences (EPSRC)³, biotechnology and biological sciences (BBSRC)⁴, and medicine (MRC)⁵. These RCs were chosen because their funding activities overlap in an area known as the *Life Sciences Interface*, which

¹ Advanced Knowledge Technologies <http://www.aktors.org>

² <http://www.aktors.org/ontology/>

³ <http://www.epsrc.ac.uk/>

⁴ <http://www.bbsrc.ac.uk/>

⁵ <http://www.mrc.ac.uk/>

supports interdisciplinary research between engineering and physical sciences, and the life sciences. In addition to this, we provided a small amount of publication data for selected individuals who are active within the life sciences interface for demonstration purposes.

We have adopted a hybrid approach [15] in our use of the AKT ontology, in which the ontology is used as a shared vocabulary to represent the data from each of the three RCs, with some local extensions to represent issues of local interest. These issues involved the representation of *research theme*. Each RC has its own notions of what constitute the different discrete areas of research which it funds. Such an approach allows us to easily integrate additional sources without any modifications to the rest of the system [15].

It is possible to gather data from institutions without their explicit cooperation, even when they have no intention to publish it in a machine-processable form. This is usually done by “screen-scraping” or extracting information from structured or semi-structured web pages, or even using Optical Character Recognition in extreme cases. In practice, such methods are far from satisfactory. They suffer from problems such as high error rates, and high maintenance (especially if web pages change), and are only sensible for initial experiments or for very valuable data that cannot be harvested any other way.

Far preferable is if the institution cooperates with the harvesting activity by publishing the data itself, either as web pages or other machine-readable form, against a well-specified structure. In our case, the EPSRC was able to supply us with the appropriate data from its own databases. The data was supplied in the form of tables (formatted as CSV files) which resulted from an agreed database query. We were then able to process the data to the form required for our activities (RDF, expressed in the AKT ontology) using simple scripts. EPSRC were also able to provide us with largely similar data from MRC, which was processed by using equivalent scripts. Some data from BBSRC was provided in the last minute. It is pleasing to note that the system was such that this data was incorporated within a few hours.

Resolving duplications is always a major task when integrating data from multiple sources [6][8]. We applied a set of heuristic techniques [1] for identifying duplicate entities and then consolidating them through the use of *owl:sameAs* assertions. By keeping the equality between entities as explicit assertions, rather than by making it implicit by rewriting gathered information to use canonical URIs for objects, we provide a means to roll back duplicate resolutions. In addition to that, we have also developed an editor which allows a user to vet the potential duplicates in a semi-automatic manner. Note that quality is very important in our context when it comes to resolving duplicates because any errors in doing so will almost definitely yield incorrect analysis results.

All this data is then stored in a 3Store [9] KB, ready for further action.

2.1 Statistics

In total, the information gathered from the RCs consists of some 3.1 million distinct statements (99.93% about instances), which when expressed in terms of the ontology and serialised as RDF/XML take up over 250Mb. The information is heavily weighted towards EPSRC-funded grants and postgraduate awards (and their associated investi-

gators), which comprise a big part of the total data, the remainder being evenly divided between MRC and BBSRC. The data, both raw CSV and processed RDF, are available from triplestore.aktors.org/demo/EPSRC/data/.

3 Conceptual Architecture

Figure 1 gives a general summary of the architecture of our proof-of-concept system. The data sources (EPSRC, BBSRC and MRC) are gathered by dedicated programs that take the native data in its raw form from traditional relational databases, and express it in terms of the common ontology.

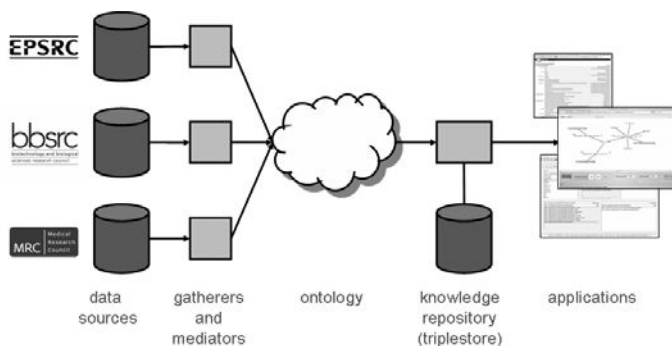


Fig. 1. Conceptual architecture

The data describing the areas covered by this system are stored in the 3store KB. Data access is provided through a web service interface to the RDQL [10] query language. Applications can make HTTP requests to the server which returns query results in an XML format.

In addition to query processing, the triplestore also performs simple inference over its data, according to the formal semantics for the RDF and RDF Schema languages.

4 Data Presentation and Analysis

We identified a range of styles that would be interesting to explore and present for the study, and have implemented different points from the spectrum. Firstly, there is simply the ability to browse the data in its raw RDF format, as well as in a rendered fashion. Secondly, we provided two visualisation tools. One shows concepts from the system (people, grants, publications, etc), and the relationships between them, and the other presents a digest of total activity between funding sources, or activities by year.

It should be noted that, particularly in the case of the visualisations, these are intended to indicate the sort of tools that can be provided to explore the data. They represent our attempts to deliver interesting utilities as a result of a short study.

4.1 Browsers

Rendered Browsing. Figure 2 shows a screenshot of the tool that allows the user to explore the data, rendered in HTML (tool located at <http://triplestore.aktors.org/browse/epsrc/>, browse for “JD Hirst”).

This is the data for Dr Hirst from the University of Nottingham, which we have chosen as the subject for our examples. This screenshot shows what it is like to browse the data, which is kept in a triplestore. Although not primarily intended for standard users, in this case we can identify some important issues.

The first is that this page offers a joint view of the data from the EPSRC, BBSRC, MRC, and also from the Research Assessment Exercise (RAE) submissions, which had already been gathered before this study. Looking at the full name data, note that the MRC knows this individual as “J D Hirst”, whereas the EPSRC knows the individual as “JD Hirst”. This is typical of the more simple variations that are seen.

Moving down the page, the sort of data we would expect for this exercise is then shown. Some publications are listed. As stated above, in this study we did not gather

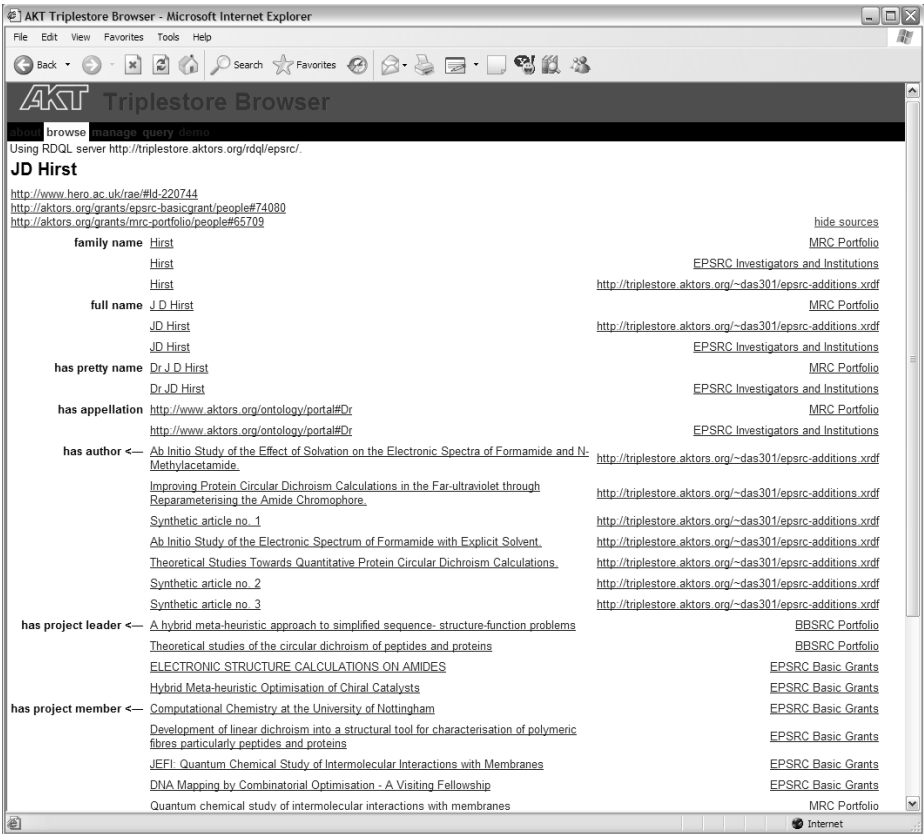


Fig. 2. Browser of HTML rendered data

publication data; however, to demonstrate how such data would look, we have found publications for this individual, as well as manufacturing some synthetic articles.

Beneath this data are the basic details of projects from the RCs. The overlap on funding is of interest; we have chosen to leave the projects that are listed by more than one council as separate projects. These projects could have been identified as the same, but the decision is one to be made in the light of the application requirements.

Raw Browsing. Figure 3 shows a prototype that gives a view that also exposes some of the more detailed workings of the system. This browsing tool is intended for the knowledge curator to get a detailed view of the data in its raw format in the triplestore.

The user has selected “Hirst” on the left, which has caused a column to appear showing the categories of knowledge the system has on people with that name. Selecting the “full-name” entry then displays the full names of all the Hirsts in the next column. Picking “JD Hirst” allows the user to find out that the data has been gathered from two sources, as the “sameAs” indicates this, and clicking on this shows the raw identifiers for the sources. The user could carry on clicking to find out details of the sources, institutions and so on. This style of interface is related to the mSpace user interfaces described in [12] and used in the application described in [13].

These interfaces are presented as sketches of different styles of low-level browsers, which gives the RCs a detailed view of their data once combined in one store.

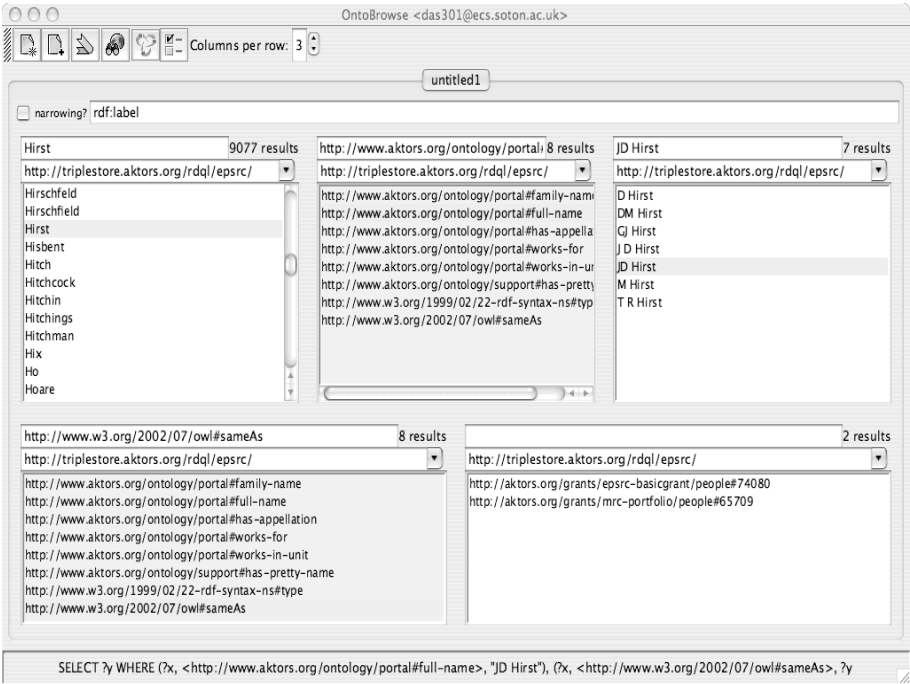


Fig. 3. Browser of raw data in triplestore

4.2 Monitoring Collaborations

We built a tool to visualise the interactions between researchers, grants, research programs, etc. This visualisation tool (named EpsrcTGViz) runs as a Java applet from any browser. It makes use of a modified version of the TouchGraph⁶ library, and it connects to a personalised ONTOCOPI web service.

ONTOCOPI (ONTOlogy-based Community Of Practice Identifier [2]) is a tool that finds sets of similar instances to a selected instance in a KB. If an ontology (i.e. both the classification structure and the KB of instantiations) represents the objects and relations in a domain, then connections between the objects can be analysed. The aim of ONTOCOPI is to extract patterns of relations that might help define a Community Of Practice (COP). COPs are informal groups of individuals interested in a particular job, procedure or work domain [16].

In the context of this work, ONTOCOPI is used to retrieve COPs of individuals or other type of object, and return the results to EpsrcTGViz. For example, the COP of the Life Science Interface research programme would include a number of individuals working on such grants, other related projects, institutions active in this research area, etc.

We can study the evolution of a scientist's collaborations by retrieving several COP sets for various dates, and monitor the rate of change of those collaborations. In other words, we can see when the scientist ceases to interact with others, and when new interactions are born. We can also find out if, and how, interactions between scientists continued once a specific grant or a project has ended.

The idea is that it should be possible to gain a sense of how research and interactions have changed over time, while keeping an eye on some level of detail. We look here at Hirst's interactions, but it would be possible to focus on other things, such as a project or a programme. Using EpsrcTGViz, we can ask for the COPs of Hirst for the years from 1998 until 2008. This is achieved by clicking the Multiple Graph button, which retrieves the data from ONTOCOPI and constructs a set of graphs that represent them, one graph per selected year. The user can then browse those graphs by selecting the year of interest, or simply move backwards and forwards through the years to view how the graphs are evolving with time. Any change in the graph is displayed incrementally, dynamically, and slowly enough to help the user perceive any transformations.

The graphs show no information for Hirst before 2001, except for a couple of papers he published with Besley in 1999. It may be tempting to assume certain things, but it is always important to look at such data in relation to the sources. It may be that the sources did not go back so far. Also, with respect to publications, if there were entries for 2001, it might be sensible to represent them in earlier years, on the basis that collaborative work takes time, and publication can be very slow. As mentioned earlier, we did not collect any publication information for this system. However, we have created some publications as illustration.

When we reach the year 2001 (figure 4), significant funding activities start to show. Hirst is now funded on three projects from three different committees. Two are BBSRC, and one is EPSRC. Logos of RCs are shown as small icons to the top right corner

⁶ <http://www.touchgraph.com>

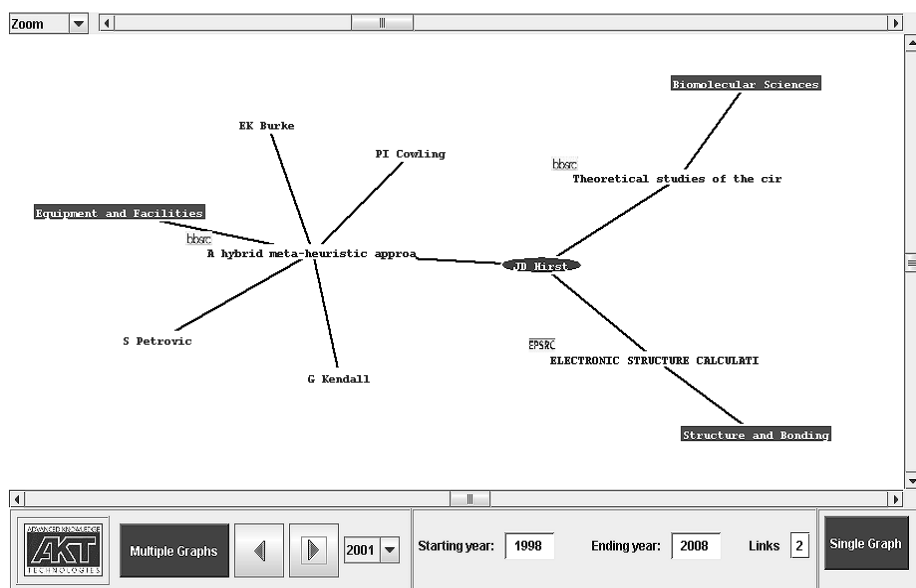


Fig. 4. Research activities of Hirst in 2001

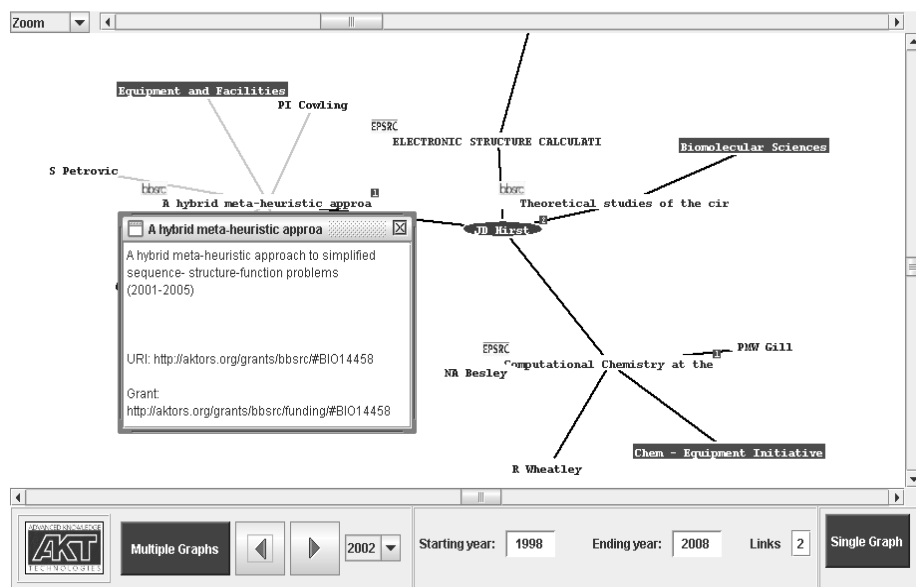


Fig. 5. Research activities of Hirst in 2002

of the nodes that represent research grants. Each grant node is linked to the nodes of people who have been identified as investigators in the RC's data, as well as to the research programme of the grant, which is displayed with dark background. As with

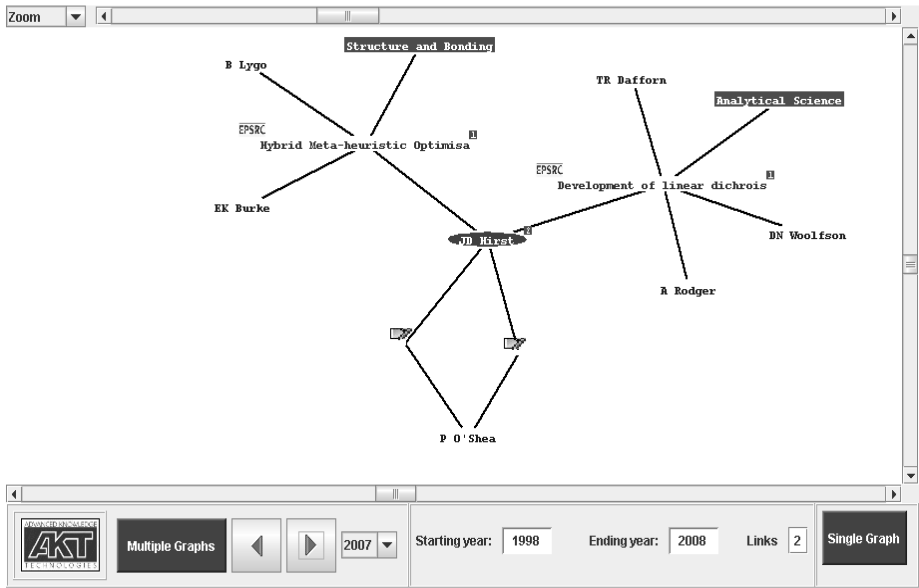


Fig. 6. Projection of Hirst’s research activities in 2007

the previous year, 2002 has full RC data (figure 5), and we can see that Hirst has gained another EPSRC project, and in fact one of the co-investigators is Beasley.

Simply viewing in this way is very interesting, but clearly a user needs to be able to explore in more detail when they find things of interest. Figure 5 shows the pop-up window that appears when the mouse hovers over the project name. Any other details could be shown, and it is possible to right click any node in the graph and select to open the relevant page in the rendered browser described in section 4.1 for full details.

Because project data has fixed durations, in some way it is possible to look into the future. In 2007, some of the earlier awards begin to end (figure 6). If we have data on publications, then one would expect to see some paper writing collaborations emerging between Hirst and the other scientists with whom he shared some grants the years before (two publications are shown in the graph as illustration).

4.3 Summarised Information

Finally we explore a representation that abstracts away from the details of individuals, projects and publications. By counting numbers of projects or level of funding or other data, interactions between groups can be explored. In the next three figures we show such “Heat Charts”.

Figure 7 looks at collaborations between different parts of the funding system, as it has changed over time. The brighter the colour, the more activity there is or was.

The metric for determining activity is the number of people who hold projects in the two areas in question, the value for each (area1, area2) combination may be determined by the number of unique solutions to this query:

This tool is intended to give a flavour of what such a knowledge system might provide, once the data is stored in a triplestore and annotated according to an ontology.

The RDF characterisation, in terms of an agreed ontology was advantageous to the development in that it provided a relatively simple common data format for integration of the data from the disparate research funding councils, and the inferential capabilities of the KB were exploited to allow general questions to be asked that would be answered using specific structures of the underlying data, without requiring the queries to be rewritten for each data source, or the data to be hand translated into a single common form.

5 Challenges and Recommendations

This study has shown that an ontology-mediated KB system can be used to gather and process data from RCs at low cost. In this section, we discuss some of the lessons learned, and describe the recommendations we made to the RCs based on this study.

5.1 Data Publishing

We realised at the start of this study the importance of requiring as little effort as possible from the data providers (ie RCs). We believe that this was a major reason why this effort succeeded. Nevertheless, data providers could publish in a more convenient form (e.g. RDF), but the model should be that they simply publish the data they wish, and processors are able to collect the data over the Internet.

Output in RDF. For this study, we received the output of database queries from the RCs and converted them to RDF with relatively simple scripts. Thus the publishing activity for the RCs is to take and maintain these small programs, or consider how they might perform the same function in other ways. The cost of doing this is very low.

Shared Ontology. We used an existing ontology (the AKT ontology) for this study. Although it is possible that each RC could publish against a different ontology which we can map to each other, we recommend that some time should be spent considering an appropriate ontology that uses widely agreed concepts, such as those provided by the Dublin Core⁷ metadata initiative.

Open Access. The primary objective of this work was to build a management tool for RC use, and to provide information to stakeholders such as government. Once the data is published in a semantic web language, it can be put into many other usages and scenarios. For example, there is renewed interest in support for an “electronic CV” for researchers, which may be seen in part as an ontologically-informed view of RC information. The RCs may consider that the provision and publication of their data is an important part of their mission. Indeed, EPSRC and the RCs have the chance to lead the knowledge-enabled society by publishing their own RDF for the Semantic Web. Doing so will encourage others to build additional tools and facilities.

⁷ www.dublincore.org

5.2 Data Acquisition

We now move from the RCs publishing data, to how that original data is acquired. Data is obviously of vital importance to this work. If the RCs are to use our system to draw any reliable conclusions about research collaborations, then the underlying data must be highly complete and accurate.

Collecting Data in RDF. The model for the RCs is that they should curate the data they need, publish it in RDF, along with mappings of their identifiers to those of other data providers. The same model should apply, in the longer term, to the institutions and any other organisations they deal with. Thus for example, instead of the RCs asking for and then curating the data for individuals from each institution, the institution should be encouraged and even required to provide the data in suitable RDF in public and private places (according to the confidentiality of the data).

Data on Publications. During this study we found that it is of great importance to have access to data on publications when tracking research collaborations. RCs do not currently hold this type of data. Institutions are now setting up publication archives, and will soon be effectively mandated to do so. It is therefore timely that RCs should require all references to publications in documents submitted to include the URL of the document in the institutional archive. This will also provide a strong spur to initiatives that are currently underway, in particular the Open Archive Initiative (OAI⁸). Indeed, the House of Commons Select Committee on Science and Technology says [11]:

“This Report recommends that all UK higher education institutions establish institutional repositories on which their published output can be stored and from which it can be read, free of charge, online.”

In the short term, however, it is sensible for RCs to gather publications data by itself, where there are areas of particular interest. This will also enable the analysis tools to include publication data at an early stage.

5.3 Referential Integrity

A key issue that we have encountered in the course of this work is that of referential integrity, which is the problem of identifying that a pair of entities in two databases are actually referring to the same object.

Multi-Sourced Data. We have taken information on projects and researchers from a number of disparate sources, as described earlier. In many cases, these sources discuss the same individuals and projects, but they each coin a different identifier to refer to these individuals and projects. For example, the subject of our earlier system description, JD Hirst, is identified by the key 74080 by EPSRC, the key 65709 by the MRC, and 220744 by the Higher Education Statistics Agency (HESA)⁹ (as used in the RAE submissions). A substantial part of our work in adapting RC databases for our use was determining whether an identifier used by one source was coreferent with an identifier used by another source.

⁸ www.openarchives.org

⁹ <http://www.hesa.co.uk/>

Imperfect Techniques. While there are heuristic techniques that can be applied to the problem of referential integrity (eg [6][5][1]), these are frequently defeasible and often require a high degree of adaptation to a particular application domain. Moreover, these techniques require sufficient high-quality data to be able to judge whether individuals are coreferent; insufficient or inconsistent data (variant name forms, for example) increase the probability of incorrect judgements [3]. When applying these techniques, the cost of both false positives (incorrectly coalescing information on two distinct individuals) and false negatives (failing to identify two individual as coreferent) must be borne in mind. In other communities, such as the library and information science community, referential integrity is managed through social means, by using name authority files, gazetteers which list the correct form of authors' names. For a system of the sort we are developing, trust is of paramount importance, and so every step should be taken to avoid false positives or negatives of any kind.

Unique Identifier Authority. The current situation in which each RC generates its own set of identifiers for referring to people, institutions, research programmes, etc., presents a significant legacy data issue, and the most appropriate way forward must take account of this. In addition, it is important to make best use of existing sources in order to minimise the duplication of effort in the alignment of these identifiers.

Our recommendation is that each RC continues to generate their own unique identifiers, but that they should also publish a mapping from their person and institution identifiers to those used by HESA, where such exist. HESA has good coverage of research staff and organisations across UK Higher Education (HE), which suggests that it is well placed to assume the role of identifier authority, but it has minimal coverage of non-HE entities. However, HESA has a high-quality dataset with properties that make it attractive for long-term use (HESA people identifiers are persistent, and do not change when personnel move between institutions). In the event that no coreferent identifier exists in HESA data, the RCs may publish pairwise mappings between their identifiers and those used by other councils.

5.4 Maintenance

Another important issue is to do with the “liveness” of the data. For this study, we chose to use data from a snapshot at a particular time, because the study did not need to keep it up to date. A production system would need to use the latest data when required. This can easily be achieved by ensuring that the RDF from the RCs is published at the required intervals (e.g. nightly), so that processors can ensure they are in synchronisation.

This approach can be practical and effective when using low cost automatic tools to format the data in RDF and publish it, with minimum or no human intervention. It is however of vital importance to couple this approach with good quality techniques and procedures for tackling the referential integrity problem discussed above.

5.5 Privacy

The RCs and others will need to have considerable regard for confidentiality and the requirements of the various Acts. We consider that the model whereby RCs choose what to publish in RDF, which will be essentially the same data as provided in systems

such as the EPSRC's Grants on the Web, gives effective control of this. By publishing the data through a single point, careful control can be maintained of what is published.

It may be that an individual RC will wish to analyse its activity using data that it does not make public, possibly using private RDF data from other sources.

6 Conclusions

Laying the proper semantic foundation for the data was the most important phase of our work. By doing so, we were able to implement a set of tools to help browsing and analysing this data. Previously, there was no joint access to this data which comes from multiple sources held in separate research agencies. There was a clear need for bridging these distributed data sources as a first step towards providing more comprehensive knowledge management tools.

We provided two tools for simple data browsing, and two other tools for analytical visualisations. These tools are meant to be simple demonstrators of what type of functionality we can add to this repository. Furthermore, they helped to gather more detailed user requirements from the RCs, some of which could be the focus of further work.

The criteria for importing data from traditional repositories into 3Store was of low cost and highly reusable. This can assure quick updates of the data whenever required. The RCs were quite pleased with the idea of keeping their traditional databases, while being able to fuse them together externally with minimum effort on their part.

The graph visualiser we developed offered the RCs a quick way to view if, and how, their grants are generating collaborations between the various scientists. This helps to make better decisions on when, and to whom, further grants should be awarded. Similarly, the heat charts provided an easy way to monitor collaborations between entire disciplines. Such a service is of great importance to RCs as it helps to quickly detect disconnections between research areas, which may feed into their future grant calls.

We believe this shows that an approach of this sort, on a wider scale, has the potential to provide EPSRC, other RCs and other stakeholders with the sort of information systems to deliver what they are now being expected to provide.

We made a set of recommendations to the RCs involved in this work to guide them through the process of building knowledge management systems. These general recommendations apply to any organisation with similar aims and requirements, and not strictly limited to any specific type of data, infrastructure, or application needs.

7 Future Work

The next phase of this study is already underway, focusing on two main objectives. The first objective is to collect data on publications to enable a finer grained analysis of collaborations. The second objective is to produce a set of data charts for the RCs, showing the rate of change of total collaborations between pairs of research programmes.

In the near future we might extend the system to display charts, similar to those presented in [2] to track changes in specific COPs over time. This type of chart can show the change in the level of n-order collaborations between researchers in relation to the duration of certain research programmes (e.g. Life Sciences Interface).

Acknowledgements. This work is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. Thanks to Daniel Smith for all his work. We are also grateful to EPSRC, and specifically to Elizabeth Hylton, Mark Hylton, and Gavin Salisbury for their time and support.

References

1. Alani H., Dasmahapatra S., Gibbins N., Glaser H., Harris S., Kalfoglou Y., O'Hara K., and Shadbolt N. *Managing Reference: Ensuring Referential Integrity of Ontologies for the Semantic Web*. Proc. 13th Int. Conf. on Knowledge Engineering and Knowledge Management, (EKAW02), Siguenza, Spain, LNAI, 317-334, 2002.
2. Alani H., Dasmahapatra S., O'Hara K., and Shadbolt, N. *ONTOCOPI - Using Ontology-Based Network Analysis to Identify Communities of Practice*. IEEE Intelligent Systems, 18(2), 18-25, 2003.
3. Alani H., Kim S., Millard D.E., Weal M.J., Hall W., Lewis P.H., and Shadbolt N. *Web based Knowledge Extraction and Consolidation for Automatic Ontology Instantiation*. Knowledge Capture (K-Cap'03), Workshop on Knowledge Markup and Semantic Annotation, Sanibel Island, FL, USA, 2003.
4. Batini C., Lenzerini M., and Navathe S.B. *A Comparative Analysis of Methodologies for Database Schema Integration*. ACM Computing Surveys, 18(4), 323-364, 1986.
5. Cohen W., Ravikumar P., and Fienberg S. *Adaptive Name Matching in Information Integration*. IEEE Intelligent Systems, Sept/Oct, 2-9, 2003.
6. Dey D., Sarkar S., and De P. *A Distance-Based Approach to Entity Reconciliation in Heterogeneous Databases*. IEEE Trans. on Knowledge And Data Eng., 14(3), 567-582, 2002.
7. Gruber T. R. *The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases*. Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, Cambridge, MA, Morgan Kaufmann, 1991.
8. Guha R. *Object Co-Identification on the Semantic Web* Proc. 13th World Wide Web Conf., New York, USA, 2004.
9. Harris S. and Gibbins N. *3Store: Efficient bulk RDF storage*. Proc. 1st Int. Workshop on Practical and Scalable Semantic Systems (PSSS'03), Sanibel Island, FL, USA, 1-20, 2003.
10. Hewlett-Packard Labs *RDQL - RDF Data Query Language*. <http://www.hpl.hp.com/semweb/rdql.htm>, 2003.
11. House of Commons Tenth Report, HC 399, July 2004. <http://www.publications.parliament.uk/pa/cm200304/cmselect/cmsctech/399/39902.htm>
12. schraefel m. c., Karam M. and Zhao S. *mSpace: interaction design for user-determined, adaptable domain exploration in hypermedia*. Proc. of AH 2003: Workshop on Adaptive Hypermedia and Adaptive Web Based Systems, 217-235, Nottingham, UK, 2003.
13. Shadbolt. N. R., Gibbins. N., Glaser. H., Harris. S., et. al. *CSAKTiveSpace or How we Learned to Stop Worrying and Love the Semantic Web*. IEEE Intelligent Systems, 2004.
14. Uschold M. and Gruninger M. *Ontologies: orinciples, methods and applications*. The Knowledge Engineering Review, 11(2), 93-136, 1996.
15. Wache H., Vögele T., Visser U., Stuckenschmidt H., et al. *Ontology-based Integration of Information - A Survey of Existing Approaches*. Proc. IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, WA, pp 108-177, 2001.
16. Wenger E., McDermott R., and Snyder W. *Cultivating Communities of Practice*. Harvard Business School Press, Cambridge, Mass, 2002

Enabling Real World Semantic Web Applications Through a Coordination Middleware

Robert Tolksdorf, Lyndon J.B. Nixon, Elena Paslaru Bontas,
Duc Minh Nguyen, and Franziska Liebsch

Free University of Berlin, Institute for Computer Science,
AG Networked Information Systems, Takustr. 9,
D-14195 Berlin, Germany
{tolk, nixon, paslaru, nguyen}@inf.fu-berlin.de
franziska@adestiny.de
<http://nbi.inf.fu-berlin.de>

Abstract. In a real world scenario Semantic Web applications must be capable to cope with the large scale, distributed, heterogeneous, unreliable and insecure environment of the World Wide Web if they are to truly represent added value to Web users. This includes issues of persistent storage, efficient reasoning, data mediation, scalability, distribution of data, fault tolerance and security. In this paper we present a coordination middleware for the Semantic Web and demonstrate its relevance to these vital issues for Semantic Web applications by elaborating a typical use case from the traffic management domain.

1 Introduction

The Semantic Web research effort is focused on providing suitable knowledge representation models and techniques for the large scale distributed environment of the Web. However there is less consideration for the particular requirements of applications which will be implemented to work with these models and techniques in order to provide intelligent Semantic Web-based functionality to users. Such applications must be equally capable to cope with the large scale, distributed, heterogeneous, unreliable and insecure environment of the World Wide Web if they are to truly represent added value to Web users. This includes issues of persistent storage, efficient reasoning, data mediation, scalability, distribution of data, fault tolerance and security.

In this paper¹ we present a coordination middleware for the Semantic Web and demonstrate its relevance to these vital issues for Semantic Web applications. We introduce a typical use case in which an intelligent traffic management system must support coordinated access to a knowledge base for a large number of

¹ This work is partially supported by the EU Network of Excellence KnowledgeWeb (FP6-507482).

agents. Through a requirements analysis and a consideration of the state of the art we note that current approaches can not adequately support such an use case and propose a new solution based on the Linda coordination model [8]. Our design and implementation approach for "Semantic Web Spaces" is described, and the operation of "Semantic Web Spaces" is illustrated through examples from the intelligent traffic management use case.

2 Overview of the Scenario

The Semantic Web is being evaluated and tested in a number of application fields in which it is expected that semantic enhancements can lead to added value for users and implementers. These fields include Web Services [17, 21, 3, 7, 18, 19], Grid Computing [10, 4, 5] and Multi-Agent Systems [24, 12, 9, 3].

In this paper we introduce a sample use case in the field of Multi-Agent Systems. We chose a traffic management scenario as it is a typical application domain for multi-agent systems which requires particular communication capabilities with support for coordination between a large number of agents. A generic scenario in this field proposes a large number of agents, both mobile (vehicular) and static (traffic controllers such as traffic lights or message systems), sending and receiving data to and from a central data store in a coordinated manner. Typically such a system, built according to the principles of multi-agent systems, has a high level architecture similar to the one illustrated in Figure 1.

Multiple sensor agents independently collect and send traffic data to the system which is persistently stored in back-end databases. In combination with route data, the system is able to interpret the data and send messages to traffic controlling agents such as traffic lights and message systems at certain locations in order to control the traffic flow (e.g. relieve congestion by holding back traffic or divert traffic around a bottleneck). In addition to this, mobile agents (i.e. vehicles) could access the system to request routing information. The system would not only calculate the possible routes from A to B on the basis of the back-end route data, but also take into account available traffic data to determine the best route at the time of request based on quantitative criteria such as

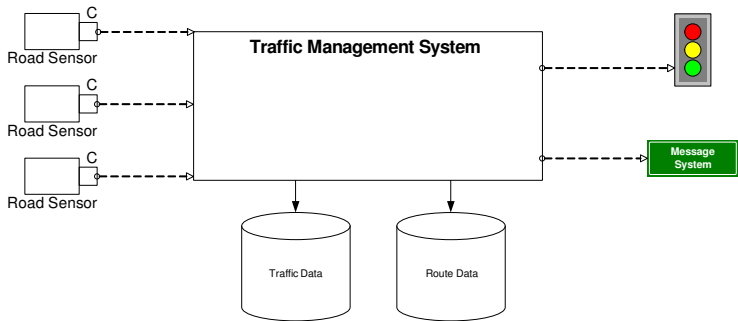


Fig. 1. Architecture of common traffic management systems

estimated journey length and duration. We expect both traffic and route data to be expressed in very low level terms, e.g. spatial position expressed using GPS coordinates and traffic conditions expressed using numeric measurements.

The Semantic Web and multi-agent systems have been identified as being complementary [12, 24]. In particular adding semantics to Web Services is likened to intelligent agent-based approaches [9, 3]. In this context, the agents are seen as self-contained applications which exhibit characteristics typical of applications on the (Semantic) Web: they are many, distributed, dynamic, heterogeneous and non-persistent. Hence we extend the traditional traffic management use case with semantics by expressing traffic and route data unambiguously in terms of a shared, common conceptualization of the domain, i.e. an ontology. This permits reasoning over the knowledge of the traffic management system to deduce new information which is of use to agents.

In the extended scenario we add ontology-based knowledge about vehicles, roadways and points of interest. Mobile agents are identified according to their location and vehicle type and can request routing information to specific points of interest. Hence the traditional traffic management system functionality is extended by the semantics and efficiently supported through Linda-like coordination. Traffic control can then take into account use cases such as restrictions on vehicle type (e.g. that a tractor can not travel on a motorway) and support queries based on reaching some specified type of service (e.g. such as a petrol station) in the most efficient means possible (e.g. inferring when and where traffic conditions are best).

3 Requirements for Semantic Web Technologies

As underlined in the previous section the usage of Semantic Web technologies in multi-agent systems provides significant advantages especially when these systems are enhanced with semantically represented domain information which can be used as shared vocabulary among agents or for inference purposes. However besides appropriate Web-compatible representation languages Semantic Web technologies need a powerful middleware for information management and agent communication, which is able to deal with typical characteristics of Web-based applications. We identified several requirements for an efficient Semantic Web middleware: 1). a decentralized and distributed architecture, in order to allow agents to publish and retrieve information efficiently and effectively; 2). scalability as a central issue because of the dimensions and the dynamics of the Web-based multi-agent scenario; 3). a high-level of abstraction to cope with inherent heterogeneity problems; and 4). support for asynchronous interaction among agents and between agents and the middleware. Interaction should be uncoupled in space and time in order to allow agents to publish and retrieve traffic information in a flexible and efficient manner. A second category of requirements relates directly to the representation and processing of scenario-relevant domain knowledge; there is a need for representation languages which are able to describe "static" domain knowledge, like types of traffic agents, points of inter-

ests, traffic conditions. Appropriate representation languages should be provided to formalize "dynamic" information e.g. recent traffic flow conditions, current events related to specific points of interest. Reasoning engines able to deal with the two types of knowledge are indispensable for the realization of intelligent multi-agent systems.

4 Semantic Web Technologies Today

This section analyzes the state of the art in Semantic Web research w.r.t. the use scenario requirements from Section 3. The Semantic Web [1] aims to provide automated information access based on machine-processable semantics of data. The first steps in this direction have been made through the realization of appropriate representation languages for Web knowledge sources like RDF(S) and OWL and the increasing dissemination of ontologies, that provide a common basis for annotation and support automatic inferencing for knowledge generation. A Description Logics-based language like OWL can be used to represent the so-called "static" domain knowledge required by the traffic management scenario (see Section 7). However formalizing "dynamic" knowledge like temporal information requires more expressive representation techniques which are not supported in a standardized manner by the Semantic Web Community. Rule languages for the Semantic Web have been proposed in several approaches [14, 15], but the interoperation between the (OWL-based) ontology layer and the consecutive rule layer is still an open issue.

The storage and processing of traffic information should be realized using a high-level, distributed middleware which permits agents to insert and retrieve information in an flexible and efficient manner. Such a middleware copes with the heterogeneity of specific storage systems for Semantic Web data [2, 11, 13] and offers a simple interface for the agents to get access to the distributed information. Currently Semantic Web technologies do not address these aspects in a satisfactory manner.

Communication and interoperation are crucial characteristics of our scenario. Currently interaction in Semantic Web applications is based on the classical client-server model and message exchange requiring strong coupling in terms of reference and time. The communication needs to be addressed to the communicating parties and it is synchronous. As mentioned in Section 3 the traffic management scenario as well as a much broader area of multi-agent applications or Web Services [6] require different communication paradigms to realize the envisioned Semantic Web.

As a conclusion of this section we underline that Semantic Web technologies do not cover the requirements of real world Web-based multi-agent systems to a satisfactory extent. We propose "Semantic Web Spaces" with an underlying tuplespace paradigm as a possible solution for an open, distributed, scalable middleware for the Semantic Web (see Section 6).

5 Linda and TupleSpaces

Before describing the key concepts of Semantic Web Spaces we introduce the foundations of our approach, the coordination language Linda and the tuplespace paradigm, and discuss how they fulfill the requirements mentioned in Section 3.

The coordination language Linda [8] has its origins in parallel computing and was developed as a means to inject the capability of concurrent programming into sequential programming languages. It consists of a shared data space (*the tuplespace*) which contains data (*the tuples*) and coordination operations (*the coordination primitives*) that are applied in the shared data space.

The tuplespace is a shared data space which acts as an associative memory² for a group of agents. A tuple is an ordered list of typed fields and retrieval is governed by matching tuples against a template which is a tuple containing both literals and typed variables. A match occurs when the template and the tuple are of the same length and the field types and the value of constant fields are identical. For example, if a tuplespace contains the tuple (*"Bobby Bear", GBP, 25.18*) then it will match a template such as (*"Bobby Bear", GBP, ?amount*) with the value 25.18 being bound to the variable *amount*³.

The coordination primitives are a small yet elegant set of operations that permit agents to emit a tuple into the tuplespace (operation *out*) or associatively retrieve tuples from the tuplespace either removing those tuples from the space (operation *in*) or not (operation *rd*). Both retrieval operations are blocking, i.e. they return only when a matching tuple is found. In this way Linda combines synchronization and communication in an extremely simple model with a high level of abstraction.

The following features of Linda have been mentioned as attractive for programming open distributed applications [20]:

- It uncouples interacting processes both in space and in time. In other words, the producer of a tuple and the consumer of that tuple do not need to know one another's location nor exist concurrently.
- It permits associative addressing, which means that data is accessed in terms of what kind of data is requested, rather than which specific data is referenced.
- It supports asynchrony and concurrency as an intrinsic part of the tuplespace abstraction.
- It separates the coordination implementation from characteristics of the host platform or programming language.

Several Linda implementations as well as extensions of the core concept have emerged in the last decades. We mention in particular XMLSpaces [22] which

² Associative retrieval implies that tuples are not addressed by ID or address, but by their content.

³ But there will be no match with (*"Polly Panda", GBP, ?amount*), (*"Bobby Bear", EUR, ?amount*) or (*"Bobby Bear", GBP, ?amount, "DiscountStock"*).

is our extension of the basic Linda model to support the manipulation of XML documents within tuple fields.

6 Semantic Web Spaces

Semantic Web Spaces is a middleware platform intended to fulfil the requirements of reliability, scalability, self-organization, coordination w.r.t. the open distributed system of the Web and of the emerging Semantic Web.

Semantic Web-based systems make use of access to knowledge stores distributed on the Web to acquire and infer knowledge required for specific tasks. These knowledge stores must handle parallel access from multiple, heterogeneous systems, coordinating responses with other systems (e.g. that resolve ontological mismatches). Applying tuplespaces to the open global environment of the Web raises new requirements, some of which have already been mentioned in other work [6, 16]:

- The naming of spaces, semantics and structure in describing the information content of the tuples. Otherwise tuples can not be distinguished from one another in terms of their contents when they have the same number of fields and field order.
- The nesting of tuples. Web data models such as XML permit the nesting of elements within a single document. Likewise Web-based information should be able to explicitly show where one unit is contained within another.
- A reference mechanism. The Web uses URIs as a global mechanism to uniquely address resources.
- A separation mechanism. Distributed applications which have independent naming schemes may use the same names for their tuplespaces, semantics or structure. On the Web, vocabularies can be kept separate – even when using the same terms – using the namespaces mechanism.
- Richer typing. Tuple values are typed according to the core data types. However this is not precise enough in a large scale environment with dynamically changing information. Richer typing can support validation and correct interaction with tuplespaces.

In a Semantic Web Space we represent RDF statements as tuples and the RDF graph as a tuplespace. Each tuple in the tuplespace has three fields typed `rdfs:Resource`, `rdf:Property` and `rdfs:Resource`, modelling the RDF triple. All RDF resources are represented in tuple fields by URIs. In order to support rich typing, tuple fields are also typed using URIs identifying classes constrained by an (RDFS/OWL) ontology. In other words, each field value in the tuplespace is associated to a RDF type. We consider three RDF modelling primitives in particular in terms of their representation within a Semantic Web Space. (See also [23]).

Blank nodes are nodes in the RDF graph which do not have any form of URI identification. However associations tied to the same blank nodes must be both maintained in the tuplespace as well as represented by clients writing tuples or

to clients reading tuples. We propose an extended RDF type called BlankNode, which can be instantiated in a tuple and given an internal unique URI value for representing that blank node in the local tuplespace, playing the role of "blank node identifier". **Containers and collections** are special RDF objects which represent sets of resources. We propose that the members of `rdfs:Container` typed resources (`rdf:Bag`, `rdf:Alt`, `rdf:Seq`) are represented by a resizable array datatype, and that the members of a collection (`rdf:List`) are represented by a closed array datatype. In the tuplespace the container or collection can be referenced as a blank node or by an URI, and is related to its members through a statement with the `rdfs:member` property. **Reification** is the means by which a statement can be made about another statement in RDF. For this, RDF uses its own vocabulary (`rdf:Statement`, `rdf:subject`, `rdf:predicate`, `rdf:object`) to identify a statement with an URI so that it can be used as a subject or object of another statement. We propose the use of nested tuples to represent reification in the tuplespace. In this way a statement (as a tuple) can be the value of a field in another tuple. Nested tuples are considered as instances of the RDF type `rdf:Statement` with a global URI as identifier.

An initial prototype of Semantic Web Spaces, called `RDFSpaces`, was already implemented. Taking advantage of the XML-oriented syntax of Semantic Web representation languages, `RDFSpaces` relies on `XMLSpaces`, which is a Java and .NET implementation of the classical Linda model extended to support the management of XML documents as types of tuples and tuple fields (We refer to [22] for details about the implementation of `XMLSpaces`, which are out of the scope of this paper). However we consider that there are two views on the tuplespace in

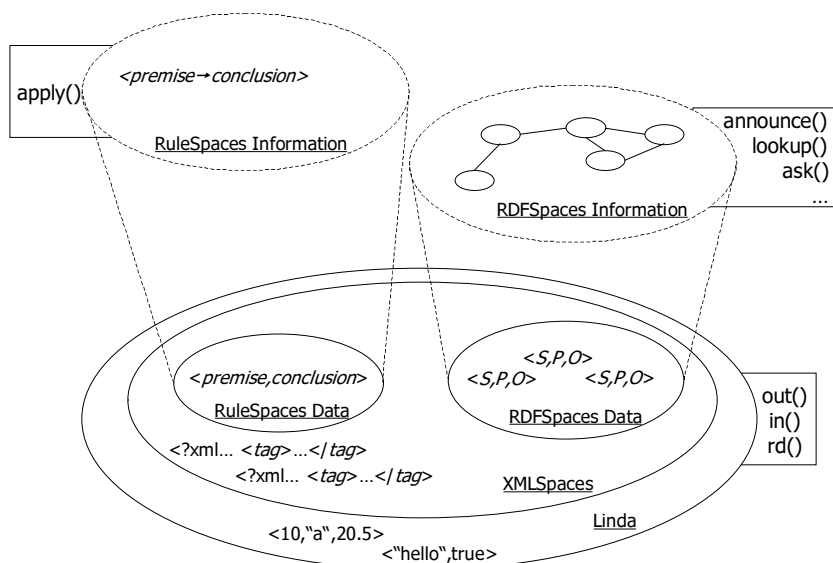


Fig. 2. The different views on the different spaces in Semantic Web Spaces

Semantic Web Spaces. The *information view* interprets the data from the space according to the semantics of the information it encodes. In this view, Semantic Web Spaces define additional primitives with their own semantics. The *data view* of Semantic Web Spaces defines the operations *out*, *in* and *rd* as they are defined in Linda. Semantic Web Spaces also provide extended matching relations that work on RDF typing and are able to take into account defined RDF(S) semantics, for example to match a sub-relation in a tuple for a relation in a template. Figure 2 shows the structure of a Semantic Web Space. The traditional Linda primitives operate upon the data view, encompassing simple datatype tuples, XMLSpaces tuples (containing XML documents) and Semantic Web tuples such as RDF and Rules. The latter also have an information view, where additional primitives are defined to operate upon the data according to the semantics of the information that it contains.

7 Design of the Traffic Management System

The requirements analysis of the traffic management scenario (see Section 2) revealed that Semantic Web technologies do not currently cover some significant aspects related to coordination and scalability. For the realization of real world Semantic Web-based systems, one needs powerful middleware technologies to cope with these typical characteristics of open distributed infrastructures as the Web. In this section we propose an extension of the Semantic Web enhanced traffic management scenario into tuplespace computing. The Linda model for coordination is suited to this scenario, as it provides the basic requirements of the system: a common data store, support for multiple agents and their interaction, coordination of that interaction and decoupling from time and space. A new architecture

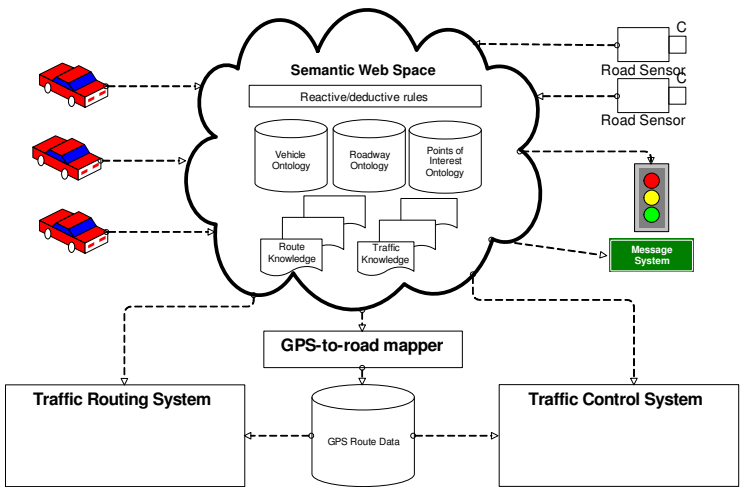


Fig. 3. Tuplespace-enhanced architecture of a traffic management system

is proposed (Figure 3) in which the agents interact directly with the data store using the simple Linda coordination primitives⁴. The functionality of the system is also abstracted into external agents who interact with the data store. This not only is a basis for modularizing the traffic management system and hence supporting reusability and updatability but also makes system knowledge directly available to any interested (and access-enabled) agents. Simple agent operations (reading some knowledge from the system) are then standardized (through Linda) and supported from the tuplespace platform without requiring any specific functionality to be executed from the traffic management system.

To illustrate the use of ontologies and tuple-space functionality we consider three use cases: (1) A vehicle requests a route not to a specific location but to a point of interest (2) Routing takes into account the characteristics of the vehicle (3) Routing takes into account characteristics of the route being proposed.

7.1 Point of Interest Use Case: Getting to the Nearest Esso Station

In this use case, low level routing data is not sufficient to meet the routing request. The system needs to be able to understand what the user is asking for (i.e. what is an Esso petrol station) and have access to the information about it (i.e. where Esso petrol stations are located). Hence we must extend the system with an ontology for points of interest and a knowledge base which defines instances of the location types in the ontology and tie them to physical locations (coordinates).

For example, there might not be an Esso petrol station anywhere nearby. The user is requesting Esso because he has a loyalty card, but this card is valid at Total petrol stations too (for the purposes of this scenario we suppose that they are brands of the same company). So with that knowledge the system could infer that the user won't mind being directed to a Total station if it is nearby. When none of these brands are close, the system should also be able to infer that the user will then accept being directed to some other station (he loses out on loyalty points but at least he can still buy petrol).

For example, a small ontology of petrol stations can be formalized in Description Logics as follows:

PetrolStation $\sqsubseteq \exists \text{belongsTo.Company}$
Company $\sqsubseteq \exists \text{hasCustomerProgram.CustomerProgram}$
Company(*Esso*)
Company(*Total*)
CustomerProgram(*PayBack*)
hasCustomerProgram(*Esso*, *PayBack*)
hasCustomerProgram(*Total*, *PayBack*)

⁴ Extensions to this primitive set will be considered later. At this stage we consider that *in* and *rd* operations retrieve all matching tuples without addressing further in this paper the multiple read problem.

An agent requesting this route sends a message with its location and a statement "find me the route to an Esso petrol station" like this⁵:

- (1) Out(**#agent876**[tms:Agent],loc:isAt,
"long=04657459345&lat=47856486475"[geo:GPS])
- (2) Out(**#agent876**[tms:Agent],loc:routeRequest,
(?X[poi:PetrolStation],poi:belongsTo,poi:Esso))

The variable ?X in (2) is constrained not only by its type but also as a subject in a nested template to the matches made to that template. In other words, ?X is matched to instances which are of type poi:PetrolStation AND which belong to the company Esso. As a result a set of tuples are inserted in which the object of the route request is a Esso petrol station. In other words the nested template sent by the agent acts as a representation of a set of matching instances.

It would be the role of the system functionality in the Traffic Routing System to monitor for a tuple with a loc:routeRequest property (i.e. a routing request) (3) and when a match is returned to query in the tuple space for the GPS location of the requesting agent (4) and the desired point of interest (5) through these templates:

- (3)Rd(?A[tms:Agent],loc:routeRequest,?P[poi:PointOfInterest])
- (4)Rd(?A[tms:Agent],loc:isAt,?START[geo:GPS])
- (5)Rd(?P[poi:PointOfInterest],loc:isAt,?END[geo:GPS])

Within the Traffic Routing System the possible routes between the GPS coordinate values tied to the variable ?START and the variable ?END can be calculated. The selected route (e.g. based on distance) would be inserted into the tuplespace (6) and read by the agent who is now monitoring for a tuple with the loc:RouteResponse property and the agent ID as subject. It is proposed that the route is expressed as a RDF sequence of GPS coordinates, i.e. the agent can retrieve the reference to the Sequence instance and can then read over time the coordinates to guide it to the point of interest.

- (6) Out(**#agent876**[tms:Agent],loc:routeResponse,**#route876**[rdf:Seq])

Through the additional ontological information it is possible to reason over alternative possibilities for the route which still fulfil the user's request. First, it could be expressed that in the matching rule that a match on an Esso petrol station, Total petrol station or an IFP petrol stations should also match on instances belonging to the other two companies. This could be done by stating that any petrol station whose company has the Payback customer program is to be matched equally. Second, it is proposed to enable in the matching procedure an optional support for supersumption, i.e. permit a match on a superset in the

⁵ A QName represents a class or instance in an ontology, a term in speech marks is a literal, a value beginning with a hash is an internal ID, a value beginning with a question mark is a variable and a value in square brackets is the field type.

event of there being no matches on a subset of the class. In this case, given that the Esso petrol station is considered a subset of all petrol stations (the property of belonging to Esso being considered a class restriction) we could support that in the event of no suitable petrol station being available a route is proposed to some other petrol station.

7.2 Vehicle Use Case: Routing a Slow-Moving Truck

Again in this use case the low level data is insufficient. The current routing calculation is based on a simple determination of possible routes from A to B, and selection based on internal calculation (e.g. the shortest distance). However road and vehicle metadata is a relevant input to the route deduction process, as e.g. a slow-moving vehicle should not travel on a motorway or a high load on a road with a low bridge crossing over it.

In this case, two ontologies are required: one for vehicle types and characteristics, and another for road types and characteristics. These ontologies then also are able to define what is logically consistent or inconsistent. A system processing possible routes with this information can then reject anything which contradicts its ontological knowledge.

For example, we could model the following ontological knowledge for a vehicle and a road:

Truck $\sqsubseteq$ *Vehicle*
Vehicle $\doteq \exists \text{hasCharacteristic.VehicleProperty}$
SlowMoving $\sqsubseteq$ *VehicleProperty*
HighLoad $\sqsubseteq$ *VehicleProperty*
 (a) *Truck* $\sqsubseteq \exists \text{hasCharacteristic.SlowMoving}$
Truck $\sqsubseteq \exists \text{hasCharacteristic.HighLoad}$
Motorway $\sqsubseteq$ *Roadway*
Roadway $\doteq \exists \text{hasCharacteristic.RoadProperty}$
Vehicle $\sqsubseteq \exists \text{travels.Roadway}$
LowBridge $\sqsubseteq$ *RoadProperty*
HighSpeed $\sqsubseteq$ *RoadProperty*
 (b) *Motorway* $\sqsubseteq \exists \text{hasCharacteristic.HighSpeed}$
 (c) $\exists \text{hasCharacteristic.SlowMoving} \sqcap \exists \text{hasCharacteristic.HighSpeed} \sqsubseteq \perp$
 $\exists \text{hasCharacteristic.HighLoad} \sqcap \exists \text{hasCharacteristic.LowBridge} \sqsubseteq \perp$

When generating a route for an agent the Traffic Routing System can check if the roadways travelled along in the route is consistent with the agent in terms of their characteristics. Note the requirement here for a GPS-to-road mapping component which is able to provide the necessary translation between GPS coordinates (which can be interpreted by the traffic management system) and roadway instances (which are understood by the Semantic Web Space). For a given route then the agent provides its characteristics (1) and an individual roadway in the route (2).

```
(1) Out(#agent876[tms:Agent], owl:sameAs, #truck876[veh:Truck])
(2) Out(#truck876[veh:Truck], loc:travelTo,
      "long=04657459367&lat=47856486511" [geo:GPS])
```

Note that the Traffic Routing System is only able to send tuples with GPS coordinates (as that is all that it understands). The GPS-to-road mapping is triggered by the GOS-to-road mapping agent monitoring for tuples stating that vehicles travel in some GPS coordinates (3), removing this tuple (2) and inserting into the tuple space a new tuple (4) with a Roadway instance value.

```
(3) In(?WHO[veh:Vehicle], loc:travelTo, ?GEO[geo:GPS])
(4) Out(#truck876[veh:Truck], loc:travelTo, #road378645[road:Motorway])
```

This statement is now inconsistent, referring to the ontological statements above: #agent876 is a particular instance of a truck, thus it can be inferred that it is slow moving (a), and is travelling onto a motorway. Since a slow moving vehicle can not travel on a motorway (b,c), #agent876 can not travel on a motorway.

The Traffic Routing System is informed that the statement it provided to the tuplespace (2) is logically false. A possible means to achieve this is that the Linda coordination primitives are extended to include the idea of checking the truth of a tuple. Then a *rdiftrue* or *iniftrue* can be used to retrieve a tuple if and only if the statement made in the tuple can be held to be logically true i.e. ontologically correct. Here the Traffic Routing System has inserted into the tuple space a set of possible roadways and now ins-if-true those tuples with a Roadway instance. A logically false tuple like the one above will not be read by the system, hence that possible route will not be considered any further.

```
InIfTrue(#truck876[veh:Truck], loc:travelTo, ?RD[road:Roadway])
```

7.3 Route Use Case: Checking the Traffic on the Whole Route

Finally in this use case we integrate the traffic conditions data being fed into the tuple space from the road sensors. In a traditional traffic management system, this data is processed by the application at a low level and results communicated to traffic control agents such as message boards or traffic lights. For example, where traffic is registered as being at a standstill diversions are placed into effect or where congestion is identified traffic may be held back.

The important aspect of traffic control data is that it is dynamically changing in real time. In a routing situation where vehicles are to be routed to their desired destination while taking into account real time changes in traffic conditions, the coordination functionality of the tuplespace is necessary. We consider a scenario where vehicle agents are constantly updating their position, are being routed to a particular destination, and are seeking to always take the fastest-moving route.

In this case, the Traffic Routing System carries out a further selection phase after checking all possible roadways for logical consistency (e.g. that no tractor is sent onto the motorway). From the set of logically consistent roadways, traffic

condition data is retrieved and the system selects the roadway with the fastest-moving traffic. This is done by retrieving the instance of a traffic sensor at the given roadway and matching on the tuple which states the current traffic speed reported from that sensor e.g.:

```
Rd((?R[tr:trafficSensor],loc:isAt,#road0857[road:Street]),
tr:hasTrafficSpeed,?I[xsd:integer])
```

The agent could then perform the remaining functionality of checking which street has the fastest moving traffic. Hence the traffic management system can perform intelligent routing in that:

- 1. The mobile agent updates the tuple space with its current location
- 2. The traffic router updates the tuples containing the potential routes to the agents desired destination
- 3. Mobile agent and route characteristics are used to identify logical inconsistencies in routes, which are ignored by agents by using a "truth" test
- 4. From the remaining route possibilities, the agent selects the quickest route using traffic conditions also being expressed in the tuple space.

As all the knowledge for the routing and traffic management is being stored in the tuple space, agents can act upon the "overall" view of the data even when some data is spatially or temporally disjoint (i.e. the originating agent is no longer available or the inserted tuple was placed into the space at an earlier timepoint). For example, in the Linda model the *rd* operation is blocking, meaning the template only returns when a match is found. This can be used by an agent to wait for a notification when the entire route is good to take:

```
Rd((?R[tr:trafficSensor],loc:isAt,(#route876[rdf:Seq],
rdfs:member,?ST[road:Roadway])),tr:hasTrafficCondition,
"GOOD"[tr:trafficCondition])
```

Note that the agent can access all roadways in a route in that a route is modelled as a RDF sequence and hence all its members can be accessed through the *rdfs:member* property. Additionally, the traffic condition with the controlled vocabulary value "GOOD" exists to simplify reasoning over the traffic statistics being generated from the traffic sensors. Statements with these conditions could be inserted from the Traffic Control System inferred from the traffic statistics that it reads from the tuple space. A possible extension would be to consider points of interest in the vicinity of the route and events associated with them that lead to changes in traffic conditions. Then the inference by the Traffic Control System could be extended to include predicted conditions based on where and when the agent will be travelling. For example, the major routes to and from a football stadium are likely to be busier at times shortly before and after a football game. Events could be integrated from other sources on the Web like a football league schedule. A similar case would be to permit the insertion of traffic announcements from other sources such as accidents (from the emergency

services) or building works (from the public roads department), and to be able to reason on the consequences for traffic on nearby roadways (e.g. if a given road is being closed off, traffic on a parallel route will shortly increase) and include this reasoning in the routing calculations. Importantly this use case raises the need for extensions with spatial and temporal ontologies and rules as well as probabilistic logic.

8 Conclusions and Future Work

In this paper we presented the usage of the tuplespace paradigm as Semantic Web middleware for a traffic management system. Tuplespaces are a good alternative to common information management and interaction models on the Web, since they allow agents to publish and retrieve information in an uncoupled manner in terms of space and time. By extending tuplespaces to represent Semantic Web knowledge we allow Semantic Web applications to store and exchange information in a decentralized and distributed manner, while taking advantage of the powerful coordination mechanism of Linda. However the realization of Semantic Web enhanced tuplespaces means not only enabling RDFS(S) and OWL data to be represented in terms of tuples, but also the revision of the classic Linda model w.r.t. the meaning of its primitives and w.r.t. scalability issues. The redefinition of Linda primitives in the context of Semantic Web Spaces with a focus on the scalability of the system is subject of future work.

References

- [1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 5 2001.
- [2] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *The Semantic Web - ISWC2002*, 2002.
- [3] P. Buhler and J. M. Vidal. Semantic Web Services as Agent Behaviors. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, editors, *Agentcities: Challenges in Open Agent Environments*, pages 25–31. Springer-Verlag, 2003.
- [4] M. Cannataro and D. Talia. The Knowledge Grid. *CACM*, 46(1):89–93, 2003.
- [5] D. De Roure and J.A. Hendler. E-Science: the Grid and the Semantic Web. *IEEE Intelligent Systems*, 19(1):65–71, 2004.
- [6] D. Fensel. Triple-based computing. <http://www.wsmo.org/2004/tp-computing/>, June 2004.
- [7] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.
- [8] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.
- [9] N. Gibbins, S. Harris, and N. Shadbolt. Agent-based Semantic Web Services. In *Proceedings of the twelfth international conference on World Wide Web*, pages 710–717. ACM Press, 2003.

- [10] C. Goble and D. De Roure. The Semantic Grid: Myth Busting and Bridge Building. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, Valencia, Spain, 2004.
- [11] S. Harris and N. Gibbins. 3store: Efficient Bulk RDF Storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, 2003.
- [12] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.
- [13] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Proceedings of the 2004 Description Logic Workshop (DL 2004)*, 2004.
- [14] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
- [15] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available at <http://www.w3.org/Submission/SWRL/>, 2004.
- [16] B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. *Journal of Systems and Software*, 69(3):243–266, 2004.
- [17] R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic Web Services: description requirements and current technologies. In *International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003)*, Pittsburgh, PA, 2003.
- [18] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. <http://www.cs.unibo.it/gaspari/www/iswc03.pdf>, 2003.
- [19] OWL Services Coalition. OWL-S: Semantic Markup for Web Services. <http://www.daml.org/services/owl-s/1.0/owl-s.pdf>, November 2003.
- [20] D. Rossi, G. Cabri, and E. Denti. Tuple-based Technologies for Coordination. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 4, pages 83–109. Springer Verlag, 2001. ISBN 3540416137.
- [21] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Proceedings of the International Conference on Web Services (ICWS'03)*, 2003, June 2003.
- [22] R. Tolksdorf and D. Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFCS International Conference on Cooperative Information Systems (CoopIS 2001)*, number LNCS 2172, pages 356–370. Springer Verlag, 2001.
- [23] R. Tolksdorf, L. Nixon, F. Liebsch, N. Duc Minh, and E. Paslaru Bontas. Semantic Web Spaces (Technical Report TR-B-04-11). Technical report, Free University of Berlin, 2004.
- [24] Y. Zou, T. Finin, L. Ding, H. Chen, and R. Pan. Using Semantic Web technology in Multi-Agent systems: a case study in the TAGA Trading Agent Environment. *Proceeding of the 5th International Conference on Electronic Commerce*, September 2003.

A Semantic Service Environment: A Case Study in Bioinformatics

Stephen Potter and Stuart Aitken

School of Informatics, The University of Edinburgh, Appleton Tower,
Crichton Street, Edinburgh EH8 9LE, UK
{stephenp, stuart}@inf.ed.ac.uk

Abstract. In recent years, web services have become increasingly important components of the scientific methodology of certain domains. Currently, however, the description and use of most these is purely ‘syntactic’; that is, the semantics of the services are left to the human user to infer or acquire by other means before deciding whether and how to use a service. Consequently, there are opportunities to bridge this semantic gap through the application of emerging semantic web and semantic web service technologies in these domains, thereby enriching and expanding a user’s service interactions. This paper presents its authors’ experiences of the application and use of these emerging technologies in a discipline in which web services already play a key role: bioinformatics.

1 Introduction

As a discipline, bioinformatics is notable for its diversity of aims and methods, and the heterogeneous nature of the computing resources applied to achieve them [13]. Bioinformaticians are accustomed to creating analysis pipelines [2] [16] [19] for, say, the assembly of a complete gene sequence from a set of subsequences derived from a lab experiment or the alignment of potentially homologous gene sequences. The dynamic nature of the databases accessed in these steps — the rate of data production is such that databases may be updated daily — creates a demand for automation in the analysis process, with the explicit representation and storage of the workflow, its invocation, and potentially, its exchange and reuse. Due to uncertainty of service availability, dynamism in the selection of an active service must be taken into account. Finally, the bioinformatician may wish to inspect the results as the workflow progresses to obtain feedback and determine whether the processing parameters are correct.

Web services is one of the paradigms that has been adopted within bioinformatics for exposing computational resources; these offer the advantages of providing an open architecture using relatively standardised transport and communications layers. However, these transactions occur at a syntactic level; there is, as yet, little semantic description of the available services. Consequently, bioinformatics presents an opportunity to apply emerging semantic web services technologies and standards to an existing set of services, and, in so doing, to learn more about the engineering aspects of such an enterprise. This paper presents our approach and experiences of an application of this sort; this should be of interest those who are planning similar applications, or who are involved in the design of these technologies and standards.

Section 2 of this paper provides a discussion of the nature of current bioinformatics web services, and presents one of a number of scenarios we have used to guide our approach. Section 3 itemises some of the requirements we identify for a practical semantic layer in this domain. This is followed by a description of how we have gone about the task of introducing these semantics. Finally, following a brief discussion of related work (section 5), some of the implications of this work are discussed and some conclusions drawn in section 6.

2 Bioinformatics Web Services

Before proceeding, it is first necessary to say a little about the nature of extant web services in bioinformatics. These resources are often provided by large bioinformatics ‘data centres’ such as the European Bioinformatics Institute (EBI)¹, the Virginia Bioinformatics Institute (VBI)², and the DNA Data Bank of Japan (DDBJ).³ Many commonly used bioinformatics computational components are available over the web, through either (or both) ‘manual’ web browser interfaces or conventional ‘programmatic’ web services interfaces. ‘Manual bioinformaticians’ will create an analysis pipeline by cutting and pasting data from one browser interface to another, often accompanied by an arbitrary amount of editing and reformatting of the data. Here, though, we are primarily concerned with the programmatic interfaces; for our purposes we assume that, in general, the offered web services share the following characteristics:

- a web service can be considered an ‘information transformer’, converting one particular input into an output; values of other parameters may qualify this behaviour.
- a service will appear as a ‘black box’ to its user; that is, the transformation it effects appears as a single atomic process.
- a web service will be ‘idempotent’; that is, no notion of state persists from one call to the service to the next (exceptions to these last two points include certain EBI services which have a notion of state that allows clients to query the current status of a running service).
- a service will be independent, in that its operation does not depend on the existence or availability of other services.
- the interface to a service will be through a SOAP [9] ‘remote procedure call’ over HTTP; this interface will be described using a WSDL [4] document.
- the WSDL document will not introduce any complex (XML Schema) typing of inputs or outputs; instead, it will rely on the use of ‘simple’ types such as “string”.

This last point raises the question of data formats for bioinformaticians. As has been noted (by, for example, Stein [18]) there are no agreed formats within bioinformatics for the formats used by services; typically they will return results represented in some *ad hoc* format, containing, say, a mixture of search results, hand-crafted natural language annotations and references to third-party publications and other data sources. As

¹ <http://www.ebi.ac.uk/xembl/XEMBL.wsdl>

² <http://staff.vbi.vt.edu/pathport/services>

³ <http://xml.nig.ac.jp>

a result, much time is expended by the users of on-line services in writing code for ‘screen-scraping’ (for extracting data from standard web browser pages) or for converting data from the format produced by one service into the format expected as the input for the next. (Indeed, the realisation of this latter fact has led to the establishment of the Open Bioinformatics Foundation⁴, whose goal of supporting bioinformatics programming has involved the creation of code libraries for a number of different programming languages for this sort of data manipulation.) This characteristic of services makes workflows in the domain particularly brittle: if a provider alters a result format, these translators must be re-engineered.

There are few universal standards for data representation. However, there are efforts to create standard representations in specific areas, for example, the MIAME (Minimum Information About a Microarray Experiment) initiative [3] aims to standardise both the recording and the reporting of microarray-based gene expression data. In the ontology arena, the Gene Ontology consortium has established a flat-file format for simple ontologies, and is moving to a more flexible format (the OBO format⁵). Currently, however, each web service (provider) will generally use its own manner of representing data, and, in the absence of standards, an alternative approach to automating service workflows is to make conversion programs — termed *shims*, following [11] — available as web services like any other. (In general, a shim service might be thought to perform a purely syntactical manipulation of its input. However, its effects might be more subtle; for instance, a shim service which selects a subset of some data might better be thought of as performing some semantic winnowing of this data.)

2.1 Scenario

To motivate this work, we have considered a number of real scenarios in which bioinformaticians achieve their aims by interacting with existing web services. These scenarios include: the assembly of a complete gene sequence from a set of subsequences; the alignment of homologous gene sequences; and the search for tissue homologies by identifying homologous genes. (These are comparable to the ‘use cases’ that are being developed under the BioMOBY initiative.⁶)

By way of a concrete example of the use of bioinformatics web services, we here present a *sequence alignment scenario*. This scenario is illustrated in Figure 1 where part a) represents the three major steps in the workflow from the user’s perspective, and part b) shows the mapping into web services. The steps are:

1. given the ID of the gene find the protein sequence that corresponds to the gene;
2. find those sequences that are potentially homologous with this sequence, i.e. find those with a high degree of similarity, and;
3. place those sequences into a relative alignment. The degree of alignment reflects the closeness of molecular function.

This corresponds to the following sequence of web service invocations:

⁴ <http://www.open-bio.org>

⁵ <http://www.geneontology.org/GO.format.html>

⁶ <http://www.biomoby.org>

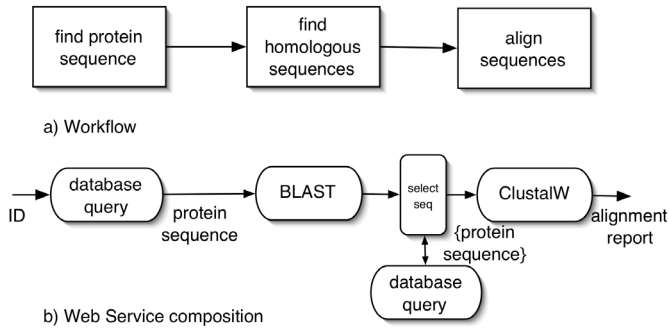


Fig. 1. Workflow and web services for the sequence alignment scenario

1. a protein sequence is obtained through a database lookup service using the supplied gene identifier;
2. a BLAST (Basic Local Alignment Search Tool) search for similar sequences in the sequence database is performed. (Since this is a search of a database of protein sequences, specifically it is the “blastp” variant program that is required here.) This is a pairwise comparison.
3. The most similar sequences are then input to the multiple sequence alignment program *ClustalW*. Multiple sequence alignment considers the entire set of matching sequences, and identifies regions of common structure.

A significant transformation of the output of the BLAST search was required to create the input to *ClustalW*. There are alternative ways of performing this transformation; here it was done by writing code to extract the identifiers of matching protein sequences from the BLAST output report, and then querying the database using these identifiers to retrieve the actual sequences.

In general, bioinformatics web services can be invoked by any client able to parse their WSDL descriptions, and handle SOAP messages over HTTP. Hence, using convenient libraries, sequences of service interactions and data transformations such as that outlined above can be embodied in a conventional computer program. Currently, however, this can be done only if the locations and data formats of the services are known to the programmer. Alternatively, a higher level interface and more flexibility is offered by a number of tools (such as the Taverna Workbench [15], which is aimed specifically (but not exclusively) at bioinformaticians) that allow their users to construct workflows and then, by handling the interactions with services, invoke them.

3 Desiderata for a Semantic Service Environment

By considering scenarios such as that above, we can identify the principal steps that the bioinformatician undertakes when constructing a workflow of services:

1. the identification and expression of the goal of the workflow. This will be determined by the immediate and long-term goals of the scientist’s research. Ideally, this externalisation of scientific goals would be made directly at a semantic level;

2. the identification of the major processing tasks, that is, developing (at the semantic level) a practicable sequence of tasks for arriving at the goal given the data currently available. This will necessarily involve some awareness of the types of service (and data) that are available so as to construct a description of this desired sequence;
3. the identification of actual (and currently active) services able to enact this sequence, including any necessary shim services. This will involve discovery of services, with perhaps a selection from among competing matching services. The discovery will typically involve searching for instances of a specified type of service and/or which produce some specified output;
4. the invocation of the workflow. At this point, the semantic description must be bound to the underlying 'syntactic' computational description of the services involved (this need not be exposed to the scientist);
5. the storage of the workflow for later reuse. For this purpose, some language rich enough to capture the semantics of the developed process would be required.

Steps 1–3 are unlikely to be wholly independent and to conform strictly to this ordering; identification of the major tasks in step 2, for instance, is likely to be influenced by and determined to some extent by knowledge of the services that are available. However, if our aim is to facilitate and enrich workflow construction of this sort, the above steps allow us to outline the semantic properties we would like in this domain, namely *service description*, *discovery*, *selection* and *invocation*, along with the *capture of workflow process*.

Currently, however, the WSDL descriptions of services are primarily syntactic in nature, inasmuch as they describe the types of the service inputs and outputs but not the semantics of these parameters or of the task that the service performs. This means that prospective clients usually need to gain an understanding of the behaviour of and interface to the service from some other source (for example, by emailing its authors or reading their web pages) in order to use the service. Furthermore, it restricts the possibilities for dynamic discovery of appropriate services to meet a client's needs (Taverna, for example, provides its users with lists of 'known' web services from which to choose; this list can be augmented if the user knows of the URIs of other services, but no automatic discovery is attempted).

Given these requirements, and the desire to apply emerging semantic web technologies rather than invent new ones for this domain, in the following sections we outline our approach to introducing semantics to bioinformatics web services.

4 A Semantic Bioinformatics Service Environment

In order to introduce semantics into the current practice of service-based bioinformatics and satisfy the above desiderata, we have introduced the following:

- The semantic description of services using OWL-S, plus a dedicated domain ontology described in OWL (described below in sections 4.1–4.3);
- An automated discovery service using a description logic reasoner (section 4.4);
- A semantic workflow tool, which acts as a user interface to the discovery service, and allows the invocation of services (section 4.5).

4.1 Introducing Semantics: OWL and OWL-S

In an attempt to move towards a more ‘semantic’ environment, we have chosen to introduce OWL-S descriptions of the existing services. OWL-S [5] is a generic upper ontology for specifying web services; it is intended to allow providers to describe (using an XML document) their services in such a manner as to allow their discovery, selection, composition and invocation; and, where appropriate, allow for monitoring, mediation and failure recovery. OWL-S is specified in OWL, the Web Ontology Language [7], which provides a language (built on the RDF data model) for specifying Description Logic (DL) constructs in the syntax of XML. DLs form a subset of first-order logics that are particularly suited to the description of hierarchical ontologies of entities, and possess appealing tractability characteristics.

The OWL-S ontology is divided into three principal areas: the *Profile*, *Model* and *Grounding*. The Profile is used to describe the purpose of the service, and so primarily has a role in the initial discovery of candidate services for a particular task. The Model describes how the service is performed, and is intended to allow simulation of and mediation with the service, to enable the execution of the service to be monitored, etc. Finally, the Grounding specifies in concrete terms how the service is actually invoked.

The role of the Profile, then, is to describe the essential capability of the service by characterizing it in functional terms (in addition, non-functional aspects of the service can be specified through ‘service parameters’). This functional characterisation is expressed by specifying the class of the service and by detailing the inputs it expects, the outputs it produces, the preconditions that are placed on the service and the effects that the service has. The description of preconditions and effects presents something of a problem for OWL-S; their expression requires, in effect, variables and rule-like constructs, which are outside the expressive capabilities of DLs (and hence, outside the capabilities of DL reasoners). In this domain, however, as noted in section 2 above, we assume that web services are essentially stateless; accordingly we do not need to model preconditions and effects, but this will not be the case for all domains. As well as characterising services, the Profile has an additional use: to allow potential clients to specify their desired services (the descriptions of which may be partial or more general in nature where certain specifics are irrelevant to the client).

Since the OWL-S ontology is essentially domain independent, in order to provide this sort of ‘semantic typing’ of both services and queries, we need to extend the ontology with ontological concepts from, in this case, the domain of bioinformatics.

4.2 A Bioinformatics Ontology Extension

The approach taken is to extend the basic generic description of an OWL-S Profile, hierarchically subclassing it with the various types of bioinformatics service that can be identified (figure 2). In addition, we also generated a hierarchy of the various conceptual data types that describe the inputs and outputs.

Being essentially simple taxonomies of services and data, these ontology extensions are not as rich as one might expect, due in part to the practicalities of expressing these concepts in DLs. For instance, when coming to define the class *ProteinSearchService*,

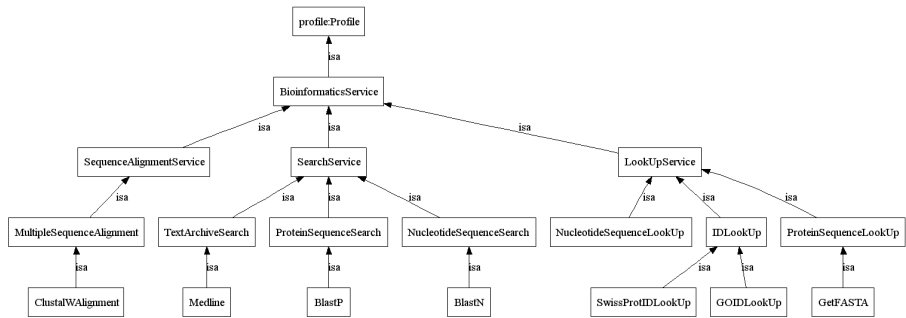


Fig. 2. A fragment of the bioinformatics ontology showing the hierarchical arrangement of some types of service. Note that the root of this subtree is the (OWL-S) Profile concept

one might reasonably attempt to express as restrictions on this class the features of its inputs and outputs that are common to all service instances of the class:

$$ProteinSearchService \equiv SearchService \sqcap \exists hasInput.ProteinSequence \sqcap \exists hasOutput.Report$$

In other words, that instances of *ProteinSearchService* have some input of class *ProteinSequence* and give some output of class *Report*. Now, one might want use this definition to define the more specialised service concept *BlastP*:

$$BlastP \equiv ProteinSearchService \sqcap \exists hasInput.DatabaseName \sqcap \exists hasInput.ProgramName \\ \sqcap \exists hasOutput.BlastSearchReport$$

Now, unless we introduce appropriate disjointness axioms and cardinality constraints, from this definition it is impossible to infer how many inputs and how many outputs a *BlastP* service should have. Introduction of these appropriate axioms and constraints can have implications for the tractability of automated reasoning (e.g., the assertion of cardinalities other than 0 or 1 takes us from the OWL-Lite subset of the language to OWL-DL). A more practical concern, though, is the engineering implications of these sort of definitions; for a bioinformatician wishing to describe his service (who cannot be assumed to be familiar with DLs), complex definitions of this sort are unwieldy and difficult to use, and can result in inappropriate conceptualisations having unintended implications. The client who wishes to state her requirements will face similar difficulties. To a certain extent, these problems are due to the unsuitability of DLs, with their lack of arbitrary variables and rule expression, for representing processes.⁷ As a consequence, the ontology extensions we have created are relatively sparse, being little more than definitions of taxonomies of concepts. The intention is that the service provider and client are able to use these ‘naively’ to express the service and data types they provide or require.

⁷ Note that the most recent version of OWL-S (version 1.1) includes the concept of a variable, and suggests ways by which to introduce rule-like expressions; however, it is not entirely clear how these should be used or reasoned with.

There are now several tools available which enable the user to create and extend OWL ontologies; here we use the Protégé ontology editor.⁸ The extensions were created by an informatics researcher who also has experience of bioinformatics, and since we restricted ourselves to modelling only certain — but hopefully representative — scenarios, these extensions represent partial views of the domain for this purpose. The creation of the ‘right’ ontologies is an issue of obvious importance, not only here but throughout the semantic web community and beyond. A good ontology in this case would both allow service providers an appropriate degree of expression to capture accurately and completely the behaviour of their services and enable clients to express their needs in as specific or as general a manner as is appropriate. In this sense, assessing the value of an ontology is a pragmatic question. Moreover, the use of domain-specific ontologies in this manner places certain practical obligations on agents in this domain: for service discovery to be possible, there must be a certain degree of consensus in the content and use of ontologies by both the service providers and potential clients.

4.3 Describing Bioinformatics Services

Now we can use the ontology extensions outlined above to describe the Profiles of bioinformatics web services in the manner suggested. Each particular service is an instance of the appropriate subclass of *BioinformaticsService*, and each of its inputs and outputs are typed⁹ with the appropriate data class expressions. We do not, however, make use of OWL-S service parameters to try to capture the non-functional qualities of these services; factors such as trust, efficiency and availability of services will undoubtedly play a major role in service selection, but remains an area of future research.

Now we need to consider how to express the other constituents of an OWL-S description, namely the model and the grounding. In each case, this is relatively straightforward. Since, as discussed in section 2, these services are generally modelled as ‘black box’ atomic processes, this naturally leads us to describe their models using the OWL-S concept of *AtomicProcess*. However, note that, while appropriate for the existing web services, this choice means that the services have rather inexpressive models, and as a result the possibilities for simulation, mediation, monitoring, etc. of services is limited. As also stated in section 2, the interfaces to these services are usually described using WSDL documents; hence this becomes the obvious choice for their OWL-S grounding.

The construction of an OWL-S document describing a particular service is a semi-automatic process. From its WSDL description, the OWL-S API [6] allows the automatic construction of a basic OWL-S outline document, having a grounding that refers to its WSDL, an atomic process model, and a profile which has the appropriate number of inputs and outputs. Using the bioinformatics ontology extension, these inputs and outputs, along with the class of profile itself must then be manually annotated with the appropriate semantic terms from the extended ontology.

⁸ <http://protege.stanford.edu/>

⁹ Through the use of the OWL-S *parameterType* relationship.

Although in this case we have provided the OWL-S descriptions of others' web services,¹⁰ ideally it would be the service providers who would generate these. This would require appropriate tools to be available and, since the task is always likely to have a manual component, the ontological descriptions to be 'usable' (a subject touched upon in the previous section).

4.4 Semantic Discovery

Among the fundamental capabilities of DL reasoning engines are the subsumption of class terms and the classification of individuals into their appropriate categories or classes. The use of OWL-S and OWL allows us to exploit their underpinnings in DLs to construct discovery services for this domain. Here, we have constructed a simple generic discovery service based on the RACER DL engine [10], which loads and maintains the current Profile ontology (including its bioinformatics extensions) in memory. On receipt from a service provider of a service advertisement in the form of (the URL of) an OWL-S document, the Profile of the service is used to classify this instance into its appropriate location in the ontology.

Subsequent queries (also in the form of OWL-S descriptions) can be interpreted as defining a class of services; the instances of those classes that are equivalent to or subsumed by this class are considered to satisfy this query. Note that queries can be as specific or as general as required, and there may be any number of services that meet a particular query, details of all of which are returned to the client. (Others have proposed similar reasoning mechanisms for discovering services — for example, see [12].) It is easy to imagine applying more elaborate reasoning here, perhaps involving aspects of automated composition to formulate and return sequences of services. In addition, the discovery service described above will miss potential service solutions that are more general than (i.e., that subsume) the current query. While this functionality could easily be provided, it raises a problem with the interpretation of the intended semantics of services: should the claim of a service to take input of some class *I* be interpreted as meaning it can handle *every* instance of (every subclass of) *I*, or merely some of these instances, of which *I* represents the 'least general generalisation'? (While the former interpretation would allow for more definitive reasoning about services and is probably the more 'correct' approach, at a pragmatic level the latter use would appear more natural.) Another difficulty surrounds queries which stipulate, for example, a relatively general input class and a relatively specific output class (as would be used when 'probing' the available services for methods that produce a particular desired output). In this case, any particular service (with, say, more specific input and more general output typings) is unlikely to either subsume or be subsumed by the query: a more sophisticated matching algorithm would be required, one which considers these different constituents of the profile description separately. Problems such as these suggest that service discovery based on the simple subsumption of Profile descriptions is unlikely to be adequate for many domains (and undermines some of the rationale for expressing OWL-S in a DL-based language).

¹⁰ A task which presented some difficulties, since — of course — these services lacked any semantic description!

Notwithstanding these shortcomings, we have chosen to construct and deploy the relatively simple algorithm described above to provide a basic functioning discovery service for our environment. This discovery service is itself implemented as a web service, with a WSDL description specifying the appropriate SOAP messages for publishing services and posting queries (these functions can also be performed through a web browser form). Hence, a further assumption about this environment is that the location of this service is known *a priori* to clients and providers alike.

4.5 Semantic Workflow Tool

The architecture outlined above can be used by clients that are able to parse WSDL and generate, send, receive and parse SOAP messages over HTTP, and, for the purpose of semantic discovery and invocation, generate and parse OWL-S descriptions. However, provision of these abilities currently places quite a burden on any prospective user. Consequently, we decided that, in order to ease this burden and provide a means to construct the workflows of services in the manner described in section 3, we would also provide a client-side interface to this architecture, in the form of a domain-independent semantic workflow tool.

Through a graphical interface, this tool allows its users to specify, using the ontology extensions of the domain in question, their (perhaps partial or general) service needs at a semantic level; these are used to generate the appropriate OWL-S queries, which are then sent to the discovery service. If matching services are returned, their descriptions are then used to fill in details (such as additional inputs and their characterisations). By specifying that the output document of one service is to form the input document of the next, ‘pipelined’ sequences of services can be constructed dynamically. Finally, values can be provided for inputs to the systems, and outputs channelled into specified outputs and the created workflow can be invoked. (This tool is domain-independent; among its parameters are details of the domain ontology extensions to use.)

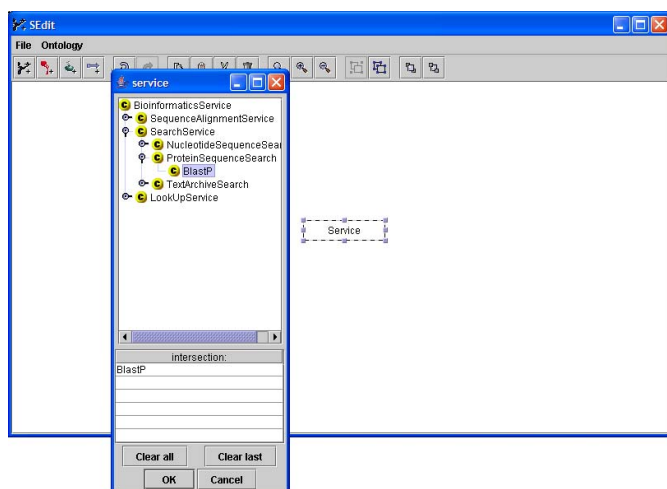


Fig. 3. Using the semantic workflow tool to define a *BlastP* service step

As an example of the use of this tool — and of the use of the semantic environment — we now describe the construction of a sequence alignment workflow as in the scenario of section 2.1. For reasons of brevity, however, we restrict this example to the definition of only the first two service steps in this sequence. The aim of this workflow, then, is to perform a BLAST search over a protein sequence database. Accordingly, the user first introduces a generic service node, and then, using the bioinformatics ontology extensions, browses the hierarchy to specify the class of desired service (figure 3). Having done this, the user then places a call to the discovery service to find if there are any services available which conform to this (partial) service definition.

The response indicates that there is a single available instance of a *BlastP* service (called “SEARCHSIMPLE”), and the user selects this to instantiate this step. This has the effect of elaborating the workflow with the three inputs and single output of this particular service, all named and typed appropriately (figure 4).

The user’s next task is to acquire the desired protein sequence that forms one of these inputs; the user does not have this data directly, and so would like to search for a service that will supply it; hence, the user replaces the input with a new generic service concept, and places a call to the discovery service. This corresponds to a request for any service that produces an output of type *ProteinSequence*. This time, the response indicates that two alternatives are available, namely “GETFASTA_SWISSENTRY” and “GETFASTA_PIRENTRY”, look-up services which access the PIR and the SwissProt protein databases respectively. Since, in this case, the user wishes to use the SwissProt database, the latter of these is selected, and its input added to the model.¹¹

Now, the user can associate appropriate values with each of these inputs (or else read the values from files) and invoke the workflow; the results are displayed to the screen and written to a file (figure 5).

To encourage interoperability and reusability, the workflow is also saved as a file conforming to the SCUFL XML workflow language used by the Taverna Workbench, into which it may then be loaded, executed, modified, etc.

4.6 Summary

It will be useful at this point to reiterate the steps that we performed in order to create this bioinformatics semantic web service-oriented environment from existing computational resources:

¹¹ Inevitably, the description of services is more complex than is suggested by this example. In particular, the ‘pipelining’ of services, as in this example, is complicated by the variety of formats used to describe what are conceptually the same input and output data, and frequently results in the need to resolve these mismatches using shim services. (In this example, the pipelined data is — conveniently enough — in the same format.) Ideally, since we are trying to operate at a ‘semantic’ level, we would like to defer consideration of such ‘syntactic’ questions until invocation-time. However, since the absence of even a single necessary shim service will prevent successful execution the entire workflow, such matters cannot be so easily ignored. Consequently, we currently model them as first-class semantic services, but the appropriate manner of describing and reasoning with data formats remains an open question.

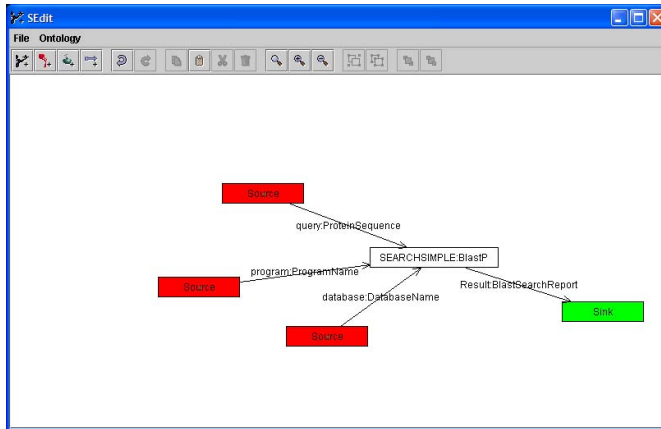


Fig. 4. A service instantiation of the *BlastP* step

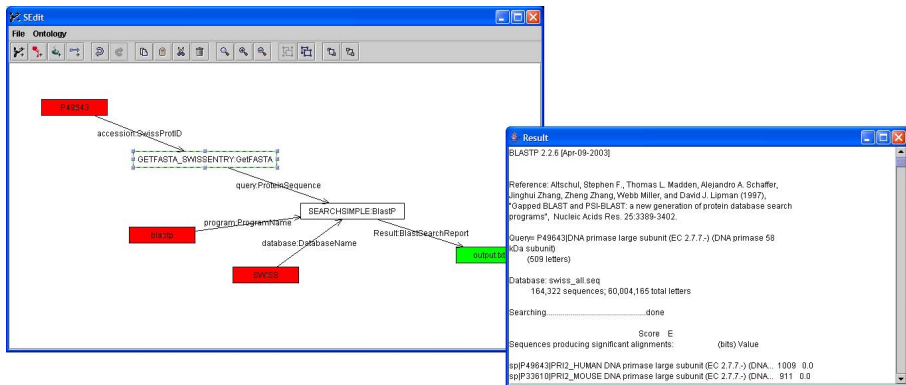


Fig. 5. Finally, the workflow is invoked and the results can be inspected

1. Analysis of existing web services and of their use by bioinformaticians has allowed us to define the desirable properties of any such environment to act as drivers for the engineering effort;
2. An OWL bioinformatics ontology extension to the OWL-S ontology was engineered; this step involves deciding how best to characterise the domain and differentiate services, from the perspectives of both service providers and potential clients.
3. Using this ontology and the WSDL description of an available service, an OWL-S description of the service is generated (in part manually).
4. Using an existing DL reasoner, a semantic discovery (web) service was constructed. This uses the ontology to classify available services, whose OWL-S descriptions can now be published to this service.
5. In this case, we provided a client-side workflow tool interface that would allow its users to interact with this environment.

By way of an aside, one might expect many of the characteristics noted above to be found in other domains in which existing computational resources are invoked from the command-line using UNIX-pipelines. In its embrace of web services, bioinformatics is a relatively advanced domain; when considering how to ‘servicify’ similar but less evolved domains, there are a number of issues that arise. These include deciding what should constitute a service in the domain (a convenient rule of thumb might be to consider what constitutes a ‘minimal unit of reusability’), how best to expose its inputs and outputs (since command-line programs will often refer to local files and directories) and how to provide the appropriate computational (HTTP, SOAP, WSDL) wrapping around the service. (With others we have addressed some of these issues when creating a semantic environment similar to that described in this paper for the domain of natural language processing; see [8] for details.)

5 Related Work

In this section we highlight some current work that is closely related to that presented in this paper. Within the bioinformatics community, the myGrid project¹² has investigated the use of DAML-S (the precursor to OWL-S) for service description, and one thread of the BioMOBY project mentioned above concerns the semantic description of services, with, as here, the use of OWL ontologies of service and data types. Rather than using the kind of service-oriented architecture adopted here, though, this work is experimenting with an alternative model in the form of the joint development between agents of a ‘negotiated’ service description. This approach is intended to help counter some of the problems that would occur whenever ontologies or service descriptions alter. However, since this work is still under development, it is not yet possible to judge the merits of this approach.

The work presented here has some overlap with the Task Computing project [14], at least in terms of the underlying semantic web technologies that are adopted, and that project’s STEER interface is somewhat similar to the workflow tool developed here. However, there are differences: Task Computing is an ambitious project concerned with pervasive computing, and adopts an appropriate non-centralised architecture for service discovery; moreover, it is aimed at a wider range of potential users than the work here, which is focused on the needs of a particular community.

The WSMO working group¹³ is directly concerned with providing semantic service environments, and in some respects represents an alternative to the OWL-S approach and the architecture that it suggests; however, its work is still at a relatively early stage, and does not yet allow a full appraisal of its applicability to this domain. The METEOR-S/WSDL-S [17] project is an attempt to integrate semantic ‘type’ descriptions (expressed using OWL constructs) more directly into WSDL documents. This sort of approach might be appropriate in this particular case, since the assumptions we make about bioinformatics services mean that, in effect, OWL-S is used for little more than typing of this sort.

¹² <http://www.mygrid.org.uk>

¹³ <http://www.wsmo.org>

6 Conclusions

The environment described above supports the interactive construction and execution of workflows, i.e., their realisation in an orchestrated sequence of web services. From the user's perspective, the creation of a workflow can take place at the 'knowledge level' of service types, with calls to the discovery service used to try to ground the workflow in actual computational resources. The choice of a particular candidate service for a given workflow step also has the effect of introducing any additional input and output parameters associated with it into the model. Data flow is achieved by 'pipelining' an output of one service into the input of another. When actual services instantiate all the steps, the workflow input values can be supplied, either as literals or from files, and the workflow can be invoked. Hence, the discovery of services plays an important role in providing access to active services. In comparison with conventional look-up approaches such as UDDI [1], which rely on generic service taxonomies, our discovery mechanism can perform more detailed matching using subsumption over service requests and advertisements. This mechanism, the workflow tool and the underlying architectural aspects of the environment are essentially domain-independent; the specifics of the domain are expressed through the ontology extensions and their use in OWL-S service descriptions.

We conclude with some remarks about the engineering aspects. The need to wrap services with a SOAP messaging layer, and generate the corresponding WSDL and (in particular) OWL-S documents remains an obstacle to those trying to 're-purpose' existing resources as web services.¹⁴ The end-user tools for doing this are not yet readily available, and until such time as they are, take-up of these technologies will necessarily be limited. At a more general level, the suitability of the OWL-S ontology and OWL itself, and, more specifically, their underlying grounding in DLs, for the purpose of describing services remains questionable. As discussed in section 4.2, DLs do not readily lend themselves to the description of processes. Furthermore, as seen in section 4.4, service discovery based simply on the subsumption of Profiles is not always going to be adequate. The workflow tool that we have developed currently has a rather limited vocabulary for specifying workflows; for instance, iterations over resources cannot be specified in the language, but must be encapsulated in a composite process. In part, this is a result of the desire to provide a simple graphical interface, which does not lend itself to subtleties of this sort (and, of course, users with more advanced needs could always revert to a conventional programming language to specify their workflows). However, another factor is that the contents of an appropriate workflow language(s) for bioinformatics (and for e-Science more generally) are, as yet, not entirely clear. Indeed, this — along with other aspects of this environment — is something that one might expect to evolve as semantic web services are assimilated into scientific research methodologies.

¹⁴ To address part of this problem, the myGrid project has developed Soaplab, to help provide a SOAP wrapper for programs; see <http://industry.ebi.ac.uk/soaplab/>.

Acknowledgements

This work is supported by BBSRC grant BBSRC 15/BEP 17046 (XSPAN), and by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

1. T. Bellwood *et al.* UDDI technical white paper. <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>.
2. E. Birney *et al.* An overview of ensembl. *Genome Research*, 14:925–928, 2004.
3. A. Brazma *et al.* Minimum information about a microarray experiment (miami): toward standards for microarray data. *Nature Genetics*, 29(4):365–371, 2001.
4. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL). <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>, 2001.
5. The OWL Services Coalition. OWL-S: Semantic markup for web services (v1.1). <http://www.daml.org/services/owl-s/1.1/>.
6. The MINDSWAP Group. OWL-S API. <http://www.mindswap.org/2004/owl-s/api/>.
7. The Web Ontology Working Group. OWL web ontology language reference. <http://www.w3.org/TR/owl-ref/>.
8. C. Grover, H. Halpin, E. Klein, J.L. Leidner, S. Potter, S. Riedel, S. Scrutchin, and R. Tobin. A framework for text mining services. In Simon J. Cox, editor, *Proceedings of the UK e-Science Programme All Hands Meeting 2004 (AHM 2004)*, pages 878–885, Nottingham, UK, 2004. 31st August–3rd September.
9. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H.F. Nielsen. Simple object access protocol (SOAP). <http://www.w3.org/TR/soap12-part1/>.
10. V. Haarslev and R. Möller. RACER system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706. Springer-Verlag, London UK, 2001.
11. D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. In *First AKT workshop on Semantic Web Services (AKT-SWS04)*, KMI, The Open University, Milton Keynes, 2004.
12. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331–339. ACM, 2003.
13. P. Lord, S. Bechhofer, M.D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *Proceedings of Third International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004, pages 350–364. Springer-Verlag LNCS 3298, 2004.
14. R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, 2003.
15. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal*, 20(17):3045–3054, 2004.
16. S.C. Potter *et al.* The ensembl analysis pipeline. *Genome Research*, 14:934–941, 2004.

17. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada (June 2003)*, pages 395–401, 2003.
18. L. Stein. Creating a bioinformatics nation. *Nature*, 417, 2002.
19. R. Stevens, R. McEntire, C.A. Goble, M. Greenwood, J. Zhao, A. Wipat, and P. Li. myGrid and the drug discovery process. *Drug Discovery Today: BIOSILICO*, 2(4):140–148, July 2004.

Towards B2B Integration in Telecommunications with Semantic Web Services

Alistair Duke¹, Marc Richardson¹, Sam Watkins¹, and Martin Roberts²

¹ BT, Next Generation Web Research, Adastral Park, Martlesham Heath,
Ipswich IP5 3RE, United Kingdom

² BT Exact, OSS Solution Design, Adastral Park, Martlesham Heath,
Ipswich IP5 3RE, United Kingdom
{alistair.duke, marc.richardson, sam.watkins,
martin.me.roberts}@bt.com

Abstract. This paper describes BT Wholesale's B2B Gateway as an approach to provide Business-to-Business integration within the Telecommunications sector. Although the Gateway provides increased efficiency over separate systems, the process to allow business partners to integrate is lengthy and costly. The application of Semantic Web Services will ease the integration process. The Web Services Modelling Ontology is described and applied to the Gateway. The paper presents the initial requirements of the Gateway upon WSMO and proposes how WSMO could provide further benefit in the future.

1 Introduction

Today's telecommunications industry is facing many challenges. After many years of high growth and profit the last few years has seen rapidly falling prices and increasingly intense competition. Operators (and in particular, the large incumbents) have realised that they must radically transform the way they do business in order to reduce costs and remain competitive.

Currently the industry suffers from high manpower costs due to a lack of automation, poor time-to-market due to inflexible business processes and customer service which has suffered due to a lack of integrated support systems. On the other hand, customers are demanding integrated services, tailored to their specific needs. The market is becoming increasingly federated due both to regulatory pressures and to companies' attempts to catch market opportunities with tailored, bundled services. In this market, the number of Business-to-Business (B2B) relationships between telecommunications companies and specialist content and service providers has dramatically increased.

The current recession, changing market and new technology has led many companies to radically rethink the way they operate. They have realised that the new environment requires tighter management of processes and the eradication of bureaucracy and duplication of effort and systems. These requirements can be met by B2B integration where companies expose interfaces to their processes and systems, thus allowing their partners to integrate. This process, however, can be time-consuming and costly and can even result in proposed services being commercially

unviable. These problems are partly due to the legal and commercial aspects of forming a partnership and partly due to a lack of automation from the technical perspective.

The EU DIP project¹ and its technical basis, the Web Services Modelling Ontology aims to address the integration problem through the adoption of Semantic Web Services. In this paper we present initial work in applying WSMO to a telecommunications integration platform as part of a DIP case study.

2 BT Wholesale's B2B Gateway

Traditionally, vertically integrated telecommunications companies such as BT have provided end-to-end services to customers using their own retail operations and their own hardware. Over recent years, these companies have worked hard to improve customer service and reduce costs through greater process efficiency and effectiveness. These efforts have been enhanced with the introduction of integrated Operational Support Systems (OSS). These can provide customers with end-to-end visibility of service delivery and assurance. The challenge in the new environment is to maintain these levels of efficiency and customer service even though there are multiple parties and organisations acting to deliver the service who inevitably have their own systems that cannot be directly integrated with those of others [1]. BT Wholesale's B2B Gateway is provided to Service Providers² to allow them to integrate their Operational Support Systems with those of BT. Without such a system the service provider would either need to manually coordinate with BT via a BT contact centre or operate a system separate to its own OSS that communicated with BT's – thus requiring information to be duplicated.

The B2B Gateway exposes an interface whose behaviour is a combination of transport technologies such as SOAP, security protocols such as SSL, messaging middleware such as ebXML and the process behaviour of back end systems.

Over the last 10 or so years BT Wholesale has been involved in a number of B2B solutions. Earliest examples were based on Electronic Data Interchange (EDI) solutions for delivering billing information. The take up of these solutions was very low and it was not until the need to offer interfaces to other licensed operators for regulated products that their use increased.

About 5 years ago the use of XML based content was demonstrated. The content for this demonstration was based on a XML version of an American EDI library. This was publicly available and utilised an early version of a schema language known as SOX (Schema for Object-oriented XML). The strength of the library and the schema language it used was the ability to build simple extensions to support industry specific solutions.

Over that last four years a number of real solutions have been deployed by BT Wholesale using XML based on this early work. The most successful has been the DSL broadband interface that has been used to capture more than 1 million orders.

¹ <http://dip.semanticweb.org>

² A service provider in this context is the organisation who has the relationship with the end customer.

With the publication of the XML Schema³ (XSD) standard there have been attempts to move the current XML interfaces away from SOX to XSD. The move has not been achieved for any live solution. For new solutions an XSD translation of the SOX library has been used or new XML schemas created where no library solution was available. The advantage of moving to the XSD format has been the availability of tools and the increased possibility of integrating with newer transport standards such as Web Services.

Currently the process involved in granting access to the Gateway for a new service provider is lengthy and complex. It commences with a commutation phase where partners assess their technical suitability, receive documentation and consider the level of fit with their existing OSS. A development phase follows this during which support is provided by BT. During the testing phase, the partner is given access to a test environment provided by BT where they can test the validity of their messages and their transport and security mechanisms. Firewalls, proxies, etc. must be configured by both parties to ensure that communication can occur. Once the testing phase is complete and documented the partner can move to a pilot phase where terms must first be agreed regarding volumes, frequency and support arrangements before access is given to the live system. Transactions are monitored during the pilot phase to ensure validity.

The process can take several months from start to finish. Any approach that can reduce development time, improve the quality of development through enhanced understanding and as a result avoid significant problems during the testing and pilot phases will naturally save BT and its partners significant time and money.

The Gateway currently exposes a number of interfaces concerned with service fulfilment and assurance. These are generally concerned with regulated services such as broadband access. The interfaces allow Service Providers to order and cease broadband lines on behalf of their customers, manage faults (i.e. raise and manage faults, request, confirm and cancel repair appointments and receive status fault status notifications) and carry out diagnostics (i.e. request tests and handle the response to these). In this paper, the application of Semantic Web Services to the Broadband Diagnostics interface is examined.

2.1 Broadband Diagnostics

As part of its OSS process, a Service Provider may wish to raise a test on the BT network. This is typically due to a problem that has been reported by one of its customers. The Service Providers OSS should collect the necessary information from the customer and assuming that the problem cannot be resolved internally issue a request via the B2B Gateway.

Interactions are implemented through the exchange of business documents, sent as messages. These interactions are known as transactions. The Gateway currently uses ebXML Business Process Specification Schema⁴ to model the sequencing of these transactions into a collaboration. The Broadband Diagnostics interface has only two transactions. These are 'RequestTest' and 'NotifyOfTestCompleted'. 'RequestTest'

³ <http://www.w3.org/XML/Schema>

⁴ <http://www.ebxml.eu.org/process.htm>

is a ‘RequestResponse’ transaction which means that a response to the test request is expected. This indicates whether the test has been accepted or rejected. It may be rejected if, for example, the Service Provider is requesting a test on a circuit that does not belong to them. The ‘NotifyOfTestCompleted’ is a ‘Notification’ transaction. This is a single message that is sent following the completion of an accepted test. It describes the results of the test. The sequence diagram for the collaboration in the case of an accepted test is shown in Figure 1.

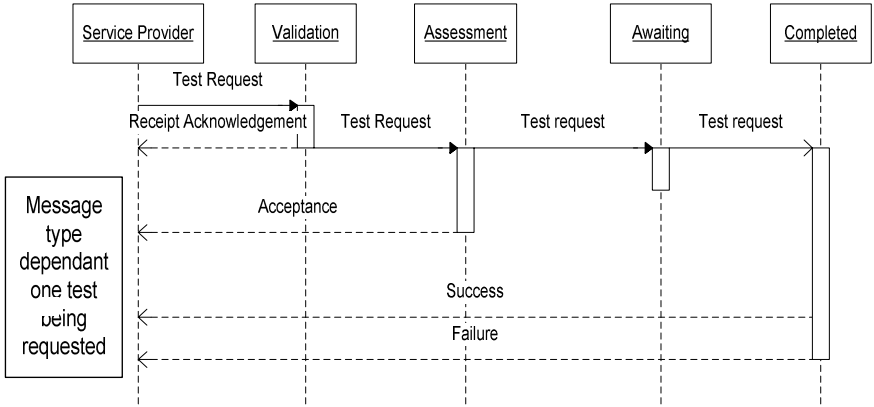


Fig. 1. Sequence Diagram for Broadband Diagnostics Collaboration.

2.2 The B2B Gateway Architecture

The B2B Gateway, in common with most B2B interfaces has three separate elements. The two internal systems of the respective organisations that need to communicate and the interface that they will use to do this. This usually involves both systems translating their internal application view of data and process into the interface view of the problem. Depending upon who produces the interface definition the amount of translation involved can be either very small or almost impossible to achieve without development effort.

The Gateway architecture can be represented as shown in Figure 2. The Service Provider’s OSS is able to generate a call to request a test. In order to pass this on to the B2B Gateway, it must first be adapted to enable it to be understood. The adaptation process has two key elements. Firstly the test call must be represented as a business message that will be understood by the gateway as a valid message given the current state of the transaction i.e. it must be represented as a TestRequest message which is the initial interaction of the ‘RequestTest’ transaction. Secondly, the business message must be wrapped within the protocol envelope i.e. ebXML Messaging.

A message received by the B2B Gateway must also be adapted before it can be processed by the BT Wholesale OSS. This adaptation is effectively the reverse of the previous one. A message handler checks an incoming message for validity according to the message protocol and assuming it is valid unwraps the business message from

the protocol. The business message is then checked for validity according to the current state of the transaction. Assuming this it valid, the message can be adapted into a call to the BT Wholesale OSS Test and Diagnostics system.

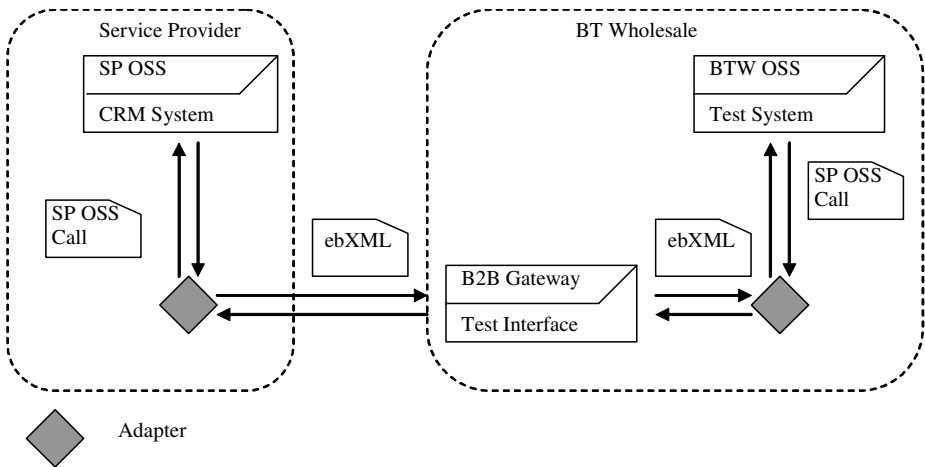


Fig. 2. B2B Gateway Architecture

The system is asynchronous so ‘Time To Perform’ values are associated with the transactions. After processing the test request, the Test and Diagnostics system responds to the Service Provider via the B2B Gateway. The process is the exact reverse of the original request.

Generating the adapter between OSS calls and valid B2B Gateway messages is one of the key challenges of the integration process. The Web Services Modelling Ontology aims to significantly simplify this integration process. The next section briefly introduces WSMO and describes how the Broadband Diagnostics interface could be modelled using it.

3 The Web Services Modelling Ontology

The Web Service Modeling Ontology⁵ along with its related language (WSML) and execution environment (WSMX) presents a complete framework for Semantic Web Services, combining Semantic Web and Web Service technologies. WSMO is an ontology and a conceptual model for the description Semantic Web Services. It is derived from and based on the Web Service Modeling Framework WSMF [2]. WSMO has a number of features (either defined or under development) that makes it appropriate (versus related approaches such as OWL-S) for application to the case study described in this paper. A detailed comparison of WSMO and OWL-S is given in [3]. WSMO features such as scalable mediation between loosely coupled elements and support for multiple groundings for processes are not well supported in OWL-S.

⁵ <http://www.wsmo.org>

Despite WSMO's advantages, it is the opinion of the authors that, in order not to inhibit its adoption, a complete departure from the W3C recommendation, OWL, should be avoided. Although WSMO claims to be able to handle other ontology languages by providing a meta-level ontology, tool support for import and export has yet to be provided.

WSMO is based on two fundamental design principles:

- Strong *de-coupling* of the various components that realize an application
- Strong *mediation* to enable anybody to speak to everybody

The Web Service Modeling Framework consists of four elements. These are Ontologies which describe the domain in a machine processable way and provide the formal semantics to the information used by other WSMO elements; Goals, which are used to define a certain aim which will be achieved by using a Web Service, or the specific objectives that a client may have when consulting a Web Service; Web Services, which provide the functionality to process data or changing states in the physical world and Mediators, which are used to map between WSMO elements if they are based on different technologies, e.g. input and output data structures, business logics, message exchange protocols, etc.

The explicit relationship between services and ontologies is the key element for Semantic Web Services and WSMO. It is envisaged that this will enable:

- **Improved service discovery**

Semantic Web search technology allows users to search on ontological concepts rather than by keyword. This would allow users (and indeed computers) to find the most appropriate services more quickly and then narrow down their search via more expressive queries if required.

- **Re-use of service interfaces in different products / settings**

Services that are described semantically can more easily be discovered, understood and applied thus reducing the need to create new services that serve the same purpose. This could also be used in a strategy to reduce complexity i.e. remove services / interfaces that exactly repeat the function of other services but are described slightly differently.

- **Simpler change management**

Changes to models and services are inevitable over time. The key thing is to reduce the knock-on effect of change or at least manage it. A Semantic approach will significantly reduce the overhead and simplify the process. For example, when a proposed change is made to a data element, those services or interfaces that employ that data in some way can be dynamically discovered and appropriate action taken e.g. to contact the owner of the service with details of the proposed change.

- **A browseable, searchable knowledge base for developers (and others)**

In tandem with the example given above for simpler change management, semantically described services and ontologies would enable a knowledge base to be constructed. This would allow developers and solution providers to perform queries relating to the data and processes they were concerned with, for example to determine the origin piece of data or its destination.

- **Semi-automatic service composition**

Given a high level goal which we wish a service or set of services to achieve, expressed in terms of an ontology, it should be possible to carry out decomposition into components parts and then match these components with appropriate services. The level of automation possible is a matter for ongoing research. Initial practical results are likely to provide users with a set of candidate services that might satisfy their needs. They are then left to decide between these services and oversee the composition required in order to satisfy the goal.

- **Mediation between the data and process requirements of component services**

Often there is need for two or more services to interact even though their communication requirements are semantically the same but syntactically different (they may require different message exchange patterns or different data formats). In this case it should be possible to automatically construct a translation between message data elements that allows the services to communicate. This process is known as mediation. It relies upon the mappings of messages and data elements to an ontology allowing semantic equivalence to be inferred.

- **Enterprise Information Integration**

As the name suggests, the Semantic Web is based upon web technology. This can afford universal (or at least enterprise wide) access to semantic descriptions of services (or information). One advantage of this is the ability to answer complex queries without having to consider how to access the various systems where the data required for the answer is held. For example, suppose there is a requirement to determine the number of customers within a particular postcode who spend more than £100 per quarter. If that information is held within one database and the person asking has access to it and knows how to query it then an answer could readily be obtained. Of course the situation is more complex if multiple databases hold the answer and access and a query interface have to be determined. The humans involved have some work to do in locating the data and processing it in the required way. A semantic approach, however, allows a single query to be made via a unifying ontology. [4]

3.1 Describing The Diagnostics Interface Using WSMO

This section describes a WSMO service description for the *testRequest* message.

WSMO service descriptions are separated into two parts – a functional part, describing what the service can actually achieve (service capability) and a non-functional part, providing additional information about the service (non-functional and quality of service properties).

Non-functional Properties

The non-functional properties for *testRequest* use the Dublin Core elements. These can be extended to include, for example, quality of service information. The property of most interest here is `dc:subject`, which allows the service to be attributed to a class. Here we use an existing telecoms industry-wide process framework: eTOM (enhanced Telecoms Operations Map)[5] to provide a domain ontology. The value of the property is: `eTOM:EvaluateAndQualifyProblem`. This allows a direct link to be made between the service and a specific class in the eTOM ontology. The eTOM prefix is an XML namespace which is shorthand for the location where the

eTOM ontology is defined. 'Evaluate & Qualify Problem' is an eTOM category in the 'Service Problem Management' process grouping. If appropriate, several eTOM categories can be attributed to a service using the subject property.

```

nonFunctionalProperties
  dc:title hasValue "Test Request"
  dc:subject hasValue "
    eTOM:EvaluateAndQualifyProblem"
  dc:description hasValue "Initiate a request for a
    Test"
  dc:contributor hasValue {
    <"http://www.btwholesale.com "> }
  dc:date hasValue "2004-12-10"
  dc:type hasValue
    <"http://www.wsmo.org/2004/d2/v1.0/#services">
  dc:format hasValue "text/plain"
  dc:language hasValue "en-US"
endNonFunctionalProperties

```

Functional Properties

The *testRequest* service could be described as having the capability that it initiates requests for diagnostic tests to be carried out over BT's network. In WSMO, modelling preconditions and postconditions provide the conditions over the information space that must hold before and after the execution of the service. Capability assumptions and effects also define conditions but describe the state of the world instead. As part of the modelling of WSMO services it is therefore necessary to identify what are the expectations and the influences of this service on the information space and the state of the world.

The inputs to *testRequest* are detailed in Table 1.

Table 1. Inputs to TestResquest transaction

Input	Description
partyID	The identifier for the Service Provider who is placing the Request for a diagnostic test
refNumber	The reference that identifies a particular test for the Service Provider
testRequestDateTime	The date and time the test request was sent.
testCategory	The type of test requested

Preconditions of this service would be is that the *partyID*, *refNumber* and *testCategory* are all valid. In WSMO this is modelled with Axioms:

```
?refNum memberOf refNumber and validRef(?refNum) .
?testCat memberOf testCategory and
validTestCat(?testCat) .
?parID memberOf partyID and validpartyID(?parID)
```

where *refNumber*, *testCategory* and *partyID* are predefined concepts and *validRef()*, *validTestCat()* and *validPartyID* are predefined axioms in the same ontology. *refNum*, *testCat* and *parID* are variables representing the input to the *testRequest* service.

The output of the service will be the results of the test, so the postcondition would be a valid test result:

```
?testRes memberOf testResult and validTestRes(testRes)
```

where *testResult* is a pre-defined concept and *validTestRes()* is a pre-defined axiom in the same ontology. *testRes* is a variable representing the output of the *testRequest* service.

To enable definition of assumptions and effects of the *testRequest* service the state of the world should be modelled. In this scenario a *testRequest* could have one of four states:

Ready – The test request is ready for submission

Awaiting – The test request has been accepted but the test has not been completed yet.

Completed – The test has been completed

Failed – The test request was not accepted and no test was carried out.

The states are modelled through variables, and the values of these indicate the state of the world at a given time. The state of the world in which *testRequest* can operate (assumption) is one where the variable *TRstate* has value “*Ready*”. The execution of *testRequest* changes the world state (effect) so that *TRState* value changes to “*Completed*”. The WSMO axiom defining this assumption can be defined with the following expression:

```
? WState memberOf WorldState and
? WState [TRState hasValue “Ready”]
```

where *WorldState* is a predefined concept having *TRState* as an attribute. The effect is similarly defined:

```
? WState memberOf WorldState and
? WState [TRState hasValue “Completed”].
```

Thus WSMO can be used to describe a rich set of properties that the service has including its capability, its data requirements for input and output and its effect of the information space and state of the world.

3.2 A WSMO-Based B2B Architecture

Describing the Broadband Diagnostics interface using WSMO allows the B2B architecture picture to be redrawn as shown in Figure 3.

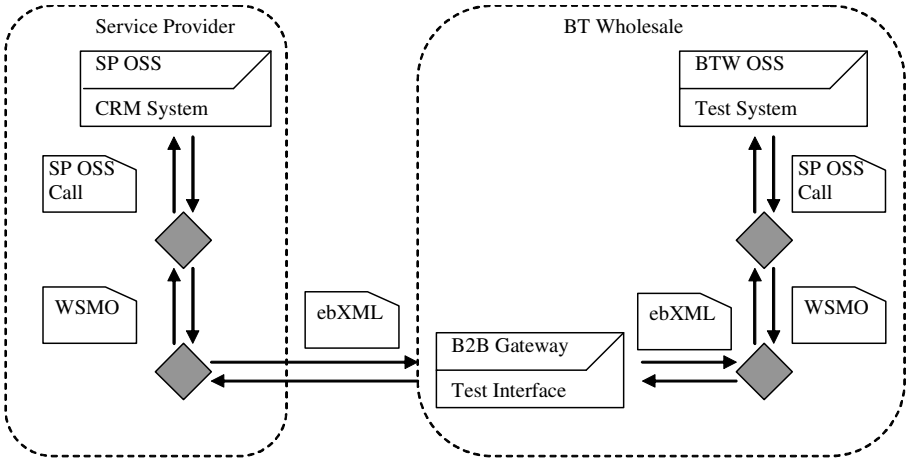


Fig. 3. A WSMO based B2B Architecture

Initially it may appear that the WSMO architecture is more complex since one extra adapter has been added to each party's side of the architecture. OSS calls are mapped to WSMO before being mapped to ebXML messages. However, from the Service Provider's perspective, the WSMO representation of the Gateway already exists – BT Wholesale has provided it. The Service Provider is required to map its calls to WSMO and then transform this representation into the WSMO representation of the Diagnostics interface. This still sounds like two major adaptation steps, however, the point is that the first of these could be automated and the second is at the ontological level and should be easier much easier to perform than the equivalent 'syntactic' transformation that is required with the current B2B gateway. The reasons behind this and approaches for automation will be identified in section 4.

From BT Wholesale's perspective, initially, more work *is* required than with the current Gateway. Firstly, it must represent its interface using WSMO. This requires developing ontologies (either from scratch or by building upon any industry ontology work such as that in the TeleManagement Forum [5]. The ontologies must represent the complete data requirements of the interface as well as the process of collaboration. However, following this effort it is to be expected that ongoing effort will be reduced since Service Providers will require less support and should be able to integrate more quickly with fewer errors.

Whether the WSMO adaptation within BT Wholesale's part of the architecture is required is an interesting point. Initially, this seems unnecessary since BT should fully understand how to integrate here and should only need to do it once. However, in practice both of these statements are rarely true: often the people responsible for

internal systems are different from those for external integration. Also, there is no reason why the internal systems should be static. Updates are always required to existing systems and software vendors may change resulting in complete platform changes. In this environment a semantically described internal interface makes more sense.

4 Mapping Messages

Designing interface content appears at first glance to be very simple. Simply gather up the dataset that you know your internal systems require and package them into suitable format using straightforward and unambiguous field names. XML brings only one significant advantage over the original fixed length records – any examples produced during design will look exactly like the final solution. This means that early examples can be ‘war gamed’ to verify a solution before any code is written. However, beyond this XML still requires about the same amount of explanation and documentation as the earlier cryptic file formats. The reason for this is purely one of semantics. XML is only an ‘improved syntax’. When humans read XML they bring their own semantic understanding to bear and this process initially makes it appear that XML is an improvement as it is supposed to be ‘self-describing’.

This apparent self-describing quality of XML has given rise to a huge number of misunderstandings when interfaces are implemented. For example is the contact for a Service Provider the person who is managing the transaction or the customer that the Service Provider is supporting? There have been examples where the same set of fields have been used to convey both interpretations. This means that BT Wholesale have difficulty knowing if the contact information provided is suitable for them to use. Another inherent problem with XML is the apparent ease with which anyone can produce it. This inevitably means consistency is threatened.

4.1 XML ‘Semantics’

There are a number of issues with representing and determining the semantics of XML including:

Structure: The hierarchical design of XML means that the placement within the structure can hold the key to the semantics of a particular data item. For example the following are all semantically equivalent but syntactically different:

1. `<ServiceProviderReference>123wqrh10</ServiceProviderReference>`
2. `<ServiceProvider>
 <Reference>123wqrh10</Reference>
</ServiceProvider>`
3. `<ServiceProvider Reference="123wqrh10"/>`

Meaning: The meaning of a field may change dependant upon who is looking at the message. Examples of this are the typical ‘Our Reference’- ‘Your Reference’ fields. The problem of using these ‘relative’ terms is that the semantics can only be

interpreted if the context, i.e. who sent the message, is understood at the time of interpretation.

Inter-field links: The traditional schema languages that are used to describe XML have no mechanism to describe inter-field links. These are where a value in one field indicates that a particular set of fields should exist or adopt certain values in a different part of the message.

This means that it is possible to create ‘valid’ messages that parse successfully through a schema validator, yet they will fail when applied to the receiving system. This can force the XML designer to adopt extra layers of hierarchy to show these links and this can mean obscuring the essence of the messages meaning.

In order to enable successful understanding of what the B2B Gateway interfaces actually mean a great deal of documentation has been required. It is ironic that in order to get computer systems to communicate successfully that you first need to communicate successfully with fellow humans. This process usually gives rise to ambiguities as to the meaning of fields. The actual process of producing documentation and associated tools such as example generators and validators actually means that the designer has to explain the meaning of each field. To help this a technique of adding metadata to the underlying schemas has been used. This metadata was defined by the ITU in a standard called tML (Telecommunications Markup Language)⁶. Adding the metadata does help document the content but it slows the process of designing interfaces down and can sometimes introduce incorrect definitions as the XML designer is not usually the person who really understands the underlying data and its associated meaning.

4.2 Achieving Successful Integration

To achieve a successful integration of two systems, they must both understand the process and data related to the problem space. Increasing the understanding of particular messages and data items through the use of XML syntax helps, but it does not remove the need for humans to do the mapping from the internal systems to the external interfaces.

This mapping involves the developers/designers in a semantic exercise that is often underestimated in its complexity and the time it can take to code a solution. This time and complexity is reduced significantly if the domain of the problem is understood by both parties and the systems using the interface have been built with the interface in mind or with at least an understanding of the domain involved. For example, if the Order system being used to produce orders has the ability to capture the required data in a manner that is similar to the requirements of the interface the chances of a successful integration are increased.

4.3 How Could Having a Semantic Layer Help?

If you take two systems and imagine that they both have a clear semantic definition of their capabilities, namely the processes and data they support, being able to access the

⁶ <http://www.atis.org/>

ability of the systems to integrate should be fairly straight forward. To date the issue has been that this Semantic layer has been in the form of document (usually Word and Excel) and is aimed purely at a human readership. If these semantics could be expressed in a computer readable form, it would be possible to do a gap analysis of the data, but also it should be possible to automatically build the mediation layer between the systems. This could mean that the interface between the systems is actually bespoke to them rather than being published as a generic interface.

This semantic layer would have to encompass not only a one dimensional understanding of the data at face value – i.e. this is a date, but also be able to understand that this date is the ‘Customer Required by Date’. Above this it would have to understand that this is the date when the product associated with the Order is to be provided. To develop this level of semantics the Semantic Layer needs to be able to express understanding not just about the messages themselves but also the underlying concepts associated with them, such as products, services, bills, invoices, etc.

With this deep understanding it becomes increasingly possible to change the way in which B2B solution actually work. From transactional model it is possible to move to a model where the exchange of relevant information and state when appropriate is the norm. The computer systems would be able to derive which bits of information they require from each other. This could mean that the systems ordering a product could be asked for extra information which is not known at the start. The fact that this new information is ‘understood’ by the Semantic layer would allow the system being asked for this information to assess if it has it or if it needs to ask a human to provide it.

Although the above utopian dream is goal worth expressing it is also worth looking at what can be achieved today with the Semantic tools available. Assuming a given process with a given set of data it should be possible to express a set of semantics that describe this finite world. By expressing an ontology of these semantics it is possible to map a given systems data into the ontology. This means that if the interface needs were expressed in terms of the initial ontology then the mapping from the system to the interface could be automatically generated.

4.4 Mapping to an Ontology

It is possible to map between XML and an ontological representation such as RDF using XSLT. However, this is problematic since two unrelated transformations – one from XML to the ontological layer and another in the opposite direction are required. In addition, because of the graph structure of RDF, there is no canonical serialisation that can ensure sequence information is maintained. A promising recent approach [6] which overcomes these issues is perhaps the most appropriate in this scenario. A bi-directional mapping is enabled through the naming of schema components which preserves the scope of elements and attributes in XML. In additions, the DIP project has considered the state-of-the-art in this area [7] and will build a WSMO mediation component that can be applied to the B2B Gateway.

5 Looking Ahead

As described in section 2, the common scenario for B2B integration in Telecommunications involves long-standing partnerships and agreements. Technical integration takes place following a legal and commercial process where service levels, price and other agreements are reached. In this scenario the need for dynamic, run-time discovery and composition is not obvious. Even at design-time, discovery is not currently required since a relatively small number of interfaces are available and these are well-known and documented – this explains the large focus on mediation requirements in this paper which most definitely is required at design-time.

This scenario is expected to change dramatically in the near future. The long-standing partnerships are unlikely to disappear overnight but shorter-term, ad-hoc collaborations are expected to emerge. For both of these forms of relationship there will be a greater reliance on dynamic integration. Organisations will expose more-and-more of their business interfaces to both their long-term partners and their customers but also to anyone else who wants to do business (of course with the appropriate restrictions). This trend is due to both regulatory pressure (e.g. as seen in the UK with ‘local-loop unbundling’) and the need to make supply chains more efficient allowing cost reductions and greater agility in service delivery. In this more open environment the level of integration, of course, increases; as do set-up and management costs. There are severe doubts as to whether a static integration approach is sustainable under these circumstances. Even if design-time integration is still heavily relied upon, the ability to discover alternative functionality, perhaps for resilience or where factors such as price and availability are changing quickly, will be required.

How does the B2B Gateway fit into this changed environment? Many more service, wholesale and content providers are expected to expose their interfaces for integration. It is here that technologies such as WSMO have real value since the initial effort required in creating ontologies, describing interfaces semantically and relating the two together is now much less than the total integration effort. The ontology created by BT Wholesale to describe their interfaces will perhaps be adopted by other smaller suppliers since they realise that service providers who are already using it can more easily integrate with them if they do so. Other ontologies are created by other dominant players in other markets such as global content providers. Here mediation between the ontologies is required for integration. Discovery in this environment leads to a dramatic change in business relationships. Today, companies ask ‘Who are my partners?’ then ‘How do I integrate with them?’. Tomorrow, companies may ask ‘What services can I discover?’ then ‘How do I partner with their owners?’.

6 Conclusion

This paper has presented an existing approach for integration within the Telecommunication sector – the BT Wholesale B2B Gateway. Although providing increased efficiency when compared to separate OSS systems the process to enable BTW’s partners to use the Gateway is long and costly. It is proposed that Semantic Web Services could ease this integration process. The Web Services Modelling

Ontology is presented together with a description of how it could be applied to the Gateway today and how it might provide further benefit in the future. Initial benefits are expected in the mediation space with discovery and composition aspects undergoing consideration following this. The paper describes initial work on a case study within the DIP project. The application of WSMO and associated tools that emerge to this case study will continue to be explored within the project.

References

1. Evans, D., Milham, D., O'Sullivan, E., Roberts, M.: Electronic Gateways—Forging the Links in Communications Services Value Chains. *The Journal of The Communications Network* Volume 1 Part 1 (2002)
2. Fensel, D., Bussler, C.: *The Web Service Modeling Framework WSMF*. *Electronic Commerce Research and Applications*, Vol. 1, Issue 2, Elsevier Science B.V. (2002)
3. Lara, R., Roman, D., Polleres, A., Fensel, D. A Conceptual Comparison of WSMO and OWL-S. In *Proceedings of the European Conference on Web Services (ECOWS 2004)*, 2004.
4. Duke, A. Davies, J., Richardson, M. Kings, N.: A Semantic Service Orientated Architecture for the Telecommunications Industry. *INTELLCOMM* (2004) 236-245
5. TeleManagement Forum, Enhanced Telecom Operations Map (eTOM) data sheet. Available on the web at: <http://www.tnforum.org/>
6. Battle, S.: Round-tripping between XML and RDF. *ISWC04 Poster Presentation* (2004)
7. Grimm, S. et al.: *Service Mediation: State-of-the-art and Requirements Analysis*, DIP Deliverable 5.1 (2004)

SWebB: Semantic Web Browsing

Fausto Giunchiglia

Dept. of Information and Communication Technology
University of Trento, 38050 Povo, Trento, Italy
fausto@dit.unitn.it

In this talk I will present a browser, called *SWebB* (Semantic Web Browser), which explicitly uses, whenever available, the semantic information codified in Contextual Ontologies. Contextual Ontologies are ontologies enriched with Context Links. Ontologies define the conceptual model of a peer and describe its content. Context mappings are sets of *bridge rules*, namely pairs which allow to relate, via appropriate semantic relations (e.g., equivalence), concepts, roles and individuals in different ontologies. Using *SWebB*, context mappings, similarly to standard Web links, can be discovered, navigated, copied, and so on. Contexts mappings are discovered by Semantic Matching. Contextual Ontologies are formalized using the C-OWL Language. The talk will also provide an overview of the key aspects of C-OWL.

References

1. F. Giunchiglia, P. Shvaiko. Semantic Matching. In *The Knowledge Engineering Review Journal*, 18(3) 2003.
2. L. Serafini, P. Bouquet, B. Magnini, S. Zanobini: Semantic Coordination: A new approach and an application. In proceedings of *ISWC'03* (2003)
3. P. Bouquet, F. Giunchiglia, F. Van Harmelen, L. Serafini, H. Stuckenschmidt . C-OWL: contextualizing ontologies. "*2nd international semantic web conference (ISWC 2003)*", edited by Dieter Fensel and Katia p. Sycara and John Mylopoulos, Sanibel Island (Fla.), 20-23 October 2003, pp. 164-179
4. F. Giunchiglia, P. Shvaiko, M. Yatskevich. S-Match: An algorithm and an implementation of semantic matching. In *Proceedings of ESWS'04*, pages 61–75, 2004.
5. F. Giunchiglia, M. Yatskevich, E. Giunchiglia. Efficient Semantic Matching. In *Proceedings of ESWC'05*, 2005.

The Semantic Grid: Past, Present and Future

David De Roure

Intelligence, Agents, Multimedia Group,
University of Southampton, Highfield,
Southampton, SO17 1BJ United Kingdom
dder@ecs.soton.ac.uk

Grid computing offers significant enhancements to our capabilities for computation, information processing and collaboration, and has exciting ambitions in many fields of endeavour. This talk will explain why the full richness of the Grid vision, with its application in e-Science, e-Research or e-Business, requires the combination of Semantic Web and Grid - giving us the "Semantic Grid", an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation. The history and state of the art in Semantic Grid will be presented, and future trends discussed.

Author Index

- Aitken, Stuart 694
Alani, Harith 664
Ali, Rana Kashif 333
Alper, Pinar 17
Angles, Renzo 346
Avrithis, Yannis 592
- Baumgartner, Robert 515
Behrendt, Wernher 257
Benerecetti, M. 211
Bindrer, Walter 32
Bloehdorn, Stephan 592
Bojars, Uldis 500
Bontcheva, Kalina 531
Bouquet, P. 211
Breslin, John G. 500
Brunzel, Marko 608
Burek, Patryk 377
- Carroll, Jeremy J. 108
Castells, Pablo 455
Castro, Rui 471
Cayzer, Steve 333
Celino, Irene 423
Chapman, Sam 649
Chirita, Paul Alexandru 439
Choi, Dae Woo 408
Ciravegna, Fabio 623, 649
Colucci, Simona 633
Constantinescu, Ion 32
- Dalamagas, Theodore 123
Decker, Stefan 500
Della Valle, Emanuele 423
Denker, Grit 78
De Roure, David 726
Di Noia, Tommaso 633
Di Sciascio, Eugenio 633
Donini, Francesco M. 633
Duke, Alistair 710
- Elenius, Daniel 78
Esposito, Floriana 138
- Faltings, Boi 32
Fensel, Dieter 1
Fernández, Miriam 455
- Gangemi, Aldo 257
Garwood, Kevin 578
Gavriloaie, Rita 439
Ghita, Stefania 439
Gibbins, Nicholas 664
Gilham, Fred 78
Giunchiglia, Enrico 272
Giunchiglia, Fausto 272, 303, 725
Glaser, Hugh 664
Goble, Carole 17, 578
Graboś, Rafał 377
Groot, Perry 318
Gutierrez, Claudio 93, 346
- Haase, Peter 182, 486
Handschuh, Siegfried 592
Harris, Stephen 664
Harth, Andreas 500
Henze, Nicola 515
Herzog, Marcus 515
Heymans, Stijn 392
Horrocks, Ian 153
Hotho, Andreas 486
Hurtado, Carlos 93
- Iannone, Luigi 138
- Jimenez, Ivan 471
- Kaoudi, Zoi 123
Keller, Uwe 1
Khoury, John 78
Kim, Wooju 408
Kompatsiaris, Yiannis 592
Kotis, Konstantinos 198
- Lanfranchi, Vitaveska 623
Lara, Rubén 1
Lausen, Holger 1
Liebsch, Franziska 679
Lopez, Vanessa 546
Lord, Phillip 17, 578

- Maass, Wolfgang 257
 Marcos, Gorka 471
 Martin, David 78
 Mauri, Marco 471
 McGuinness, Deborah L. 303
 Motta, Enrico 546
 Müller, Roland M. 608

 Nejdll, Wolfgang 290, 439
 Nguyen, Duc Minh 679
 Nixon, Lyndon J.B. 679
 Norton, Barry 649

 Olmedilla, Daniel 290

 Paiu, Raluca 439
 Palmisano, Ignazio 138
 Pan, Jeff Z. 153
 Park, Sangun 408
 Parkinson, Helen 578
 Pasin, Michele 546
 Paslaru Bontas, Elena 679
 Paton, Norman W. 578
 Peer, Joachim 47
 Petrelli, Daniela 623
 Petridis, Kosmas 592
 Phillips, Addison 108
 Pinciamore, Massimiliano 471
 Pinheiro da Silva, Paulo 303
 Pinto, H. Sofia 241
 Pistore, Marco 62
 Polleres, Axel 1
 Posada, Jorge 471
 Potter, Stephen 694

 Ragone, Azzurra 633
 Redavid, Domenico 138
 Reinberger, Marie-Laure 563
 Richardson, Marc 710
 Roberti, Pierluigi 62
 Roberts, Martin 710

 Saathoff, Carsten 592
 Sadaati, Shahin 78
 Schaal, Markus 608
 Schlobach, Stefan 226

 Schmidt-Thieme, Lars 486
 Sellis, Timos 123
 Selvini, Paolo 471
 Semeraro, Giovanni 138
 Senanayake, Rukman 78
 Serafini, Luciano 361
 Sevilmis, Neyir 471
 Shadbolt, Nigel 664
 Shvaiko, Pavel 303
 Simou, Nikos 592
 Smithers, Tim 471
 Spiliopoulou, Myra 608
 Spyns, Peter 563
 Staab, Steffen 241, 592
 Stojanovic, Ljiljana 182
 Stork, André 471
 Straccia, Umberto 167
 Strintzis, Michael G. 592
 Stuckenschmidt, Heiner 318
 Sure, York 241, 486

 Tamin, Andrei 361
 Tempich, Christoph 241
 Thelen, Bruno 471
 Tolksdorf, Robert 679
 Traverso, Paolo 62
 Tzouvaras, Vassilis 592

 Vaisman, Alejandro 93
 Vallet, David 455
 Van Nieuwenborgh, Davy 392
 Vermeir, Dirk 392
 Vouros, George A. 198

 Wache, Holger 318
 Watkins, Sam 710
 Westenthaler, Rupert 257
 Winslett, Marianne 290
 Wroe, Chris 17

 Yatskevich, Mikalai 272

 Zanobini, S. 211
 Zecchino, Vincenzo 471
 Zhang, Charles C. 290