Lecture Notes in Computer Science 3870

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Stefano Spaccapietra Paolo Atzeni
Wesley W. Chu Tiziana Catarci
Katia P. Sycara (Eds.)

Journal on Data Semantics V

Springer

Volume Editors

Stefano Spaccapietra
École Polytechnique Fédérale de Lausanne, EPFL
Laboratoire de Bases de Données
Station 14, 1015 Lausanne, Switzerland
E-mail: stefano.spaccapietra@epfl.ch

Paolo Atzeni
Università Roma Tre, Dipartimento di Informatica e Automazione
Via della Vasca Navale 79, 00146 Roma, Italy
E-mail: atzeni@dia.uniroma3.it

Wesley W. Chu
University of California, Computer Science Department
Los Angeles, CA 90095-1596, USA
E-mail: wwc@cs.ucla.edu

Tiziana Catarci
Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica
Via Salaria 113, 00198 Roma, Italy
E-mail: catarci@dis.uniroma1.it

Katia P. Sycara
Carnegie Mellon University, Robotics Institute
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
E-mail: katia@cs.cmu.edu

Library of Congress Control Number: 2005938929

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

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-31426-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-31426-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 11617808 06/3142 5 4 3 2 1 0

The LNCS Journal on Data Semantics

Computerized information handling has changed its focus from centralized data management systems to decentralized data exchange facilities. Modern distribution channels such as high-speed Internet networks and wireless communication infrastructure, provide reliable technical support for data distribution and data access, materializing the new, popular idea that data may be available to anybody, anywhere, anytime. However, providing huge amounts of data on request often turns into a counterproductive service, making the data useless because of poor relevance or inappropriate level of detail. Semantic knowledge is the essential missing piece that allows the delivery of information that matches user requirements. Semantic agreement, in particular, is essential to meaningful data exchange.

Semantic issues have long been open issues in data and knowledge management. However, the boom in semantically poor technologies, such as the Web and XML, has boosted renewed interest in semantics. Conferences on the Semantic Web, for instance, attract crowds of participants, while ontologies on their own have become a hot and popular topic in the database and artificial intelligence communities.

Springer's *LNCS Journal on Data Semantics* aims at providing a highly visible dissemination channel for most remarkable work that in one way or another addresses research and development on issues related to the semantics of data. The target domain ranges from theories supporting the formal definition of semantic content to innovative domain-specific application of semantic knowledge. This publication channel should be of highest interest to researchers and advanced practitioners working on the Semantic Web, interoperability, mobile information services, data warehousing, knowledge representation and reasoning, conceptual database modeling, ontologies, and artificial intelligence.

Topics of relevance to this journal include:

- Semantic interoperability, semantic mediators
- Ontologies
- Ontology, schema and data integration, reconciliation and alignment
- Multiple representations, alternative representations
- Knowledge representation and reasoning
- Conceptualization and representation
- Multi-model and multi-paradigm approaches
- Mappings, transformations, reverse engineering
- Metadata
- Conceptual data modeling
- Integrity description and handling
- Evolution and change
- Web semantics and semi-structured data
- Semantic caching

- Data warehousing and semantic data mining
- Spatial, temporal, multimedia and multimodal semantics
- Semantics in data visualization
- Semantic services for mobile users
- Supporting tools
- Applications of semantic-driven approaches

These topics are to be understood as specifically related to semantic issues. Contributions submitted to the journal and dealing with semantics of data will be considered even if they are not within the topics in the list.

While the physical appearance of the journal issues looks like the books from the well-known Springer LNCS series, the mode of operation is that of a journal. Contributions can be freely submitted by authors and are reviewed by the Editorial Board. Contributions may also be invited, and nevertheless carefully reviewed, as in the case for issues that contain extended versions of best papers from major conferences addressing data semantics issues. Special issues, focusing on a specific topic, are coordinated by guest editors once the proposal for a special issue is accepted by the Editorial Board. Finally, it is also possible that a journal issue be devoted to a single text.

The journal published its first volume in 2003 (LNCS 2800), its second volume at the beginning of 2005 (LNCS 3360), and its third volume in summer 2005 (LNCS 3534). Volumes I and II, as this volume V, are special issues composed of selected extended versions of best conference papers. Volume III is a special issue on Semantic-based Geographical Information Systems, coordinated by guest editor Esteban Zimányi. The fourth volume is the first "normal" volume, comprising spontaneous submissions on any of the topics of interest to the journal. Currently planned volumes include a special issue on Emergent Semantics.

The Editorial Board comprises one Editor-in-Chief (with overall responsibility) and several members. The Editor-in-Chief has a 4-year mandate to run the journal. Members of the board have a 3-year mandate. Mandates are renewable. More members may be added to the board as appropriate.

We are happy to welcome you to our readership and authorship, and hope we will share this privileged contact for a long time.

Stefano Spaccapietra
Editor-in-Chief
http://lbdwww.epfl.ch/e/Springer/

JoDS Volume V – Guest Editorial

To foster the dissemination of the best ideas and results, the *Journal on Data Semantics* (JoDS) pursues a policy that includes annually publishing extended versions of the best papers from selected conferences whose scope encompasses or intersects the scope of the journal.

This initiative is motivated by the difference in goals we have between conferences and journals. Conferences usually have a faster turnaround and focused audience, but they have to enforce space limitation and a fixed time frame, with no chances for improving a paper by producing multiple versions. In contrast, journals offer more space, room for debate and refinement, and are usually considered the real archival venue.

Therefore, the publication of an extended version of a conference paper is a much appreciated opportunity for researchers to widely disseminate a significantly improved presentation of their work, where they can develop the appropriate motivations, reasoning, results and comparative analysis.

This issue includes selections from three international conferences: ER 2004, the 23rd International Conference on Conceptual Modeling, which took place in November 2004 in Shanghai, China, ODBASE 2004, the Third International Conference on Ontologies, Databases, and Applications of Semantics, which took place in October 2004 in Ayia Napa, Cyprus, and ICSNW 2004, the First International Conference on Semantics of a Networked World, organized by IFIP WG 2.6 in Paris, France, June 2004.

Papers from these conferences were selected based on their quality, relevance and significance, and the viability of extending their results. All extended papers were subject to a scholarly review process, and the authors were required to respond to all concerns expressed by the reviewers before papers were accepted.

Four papers, showing consistently high reviews from the Program Committee, were selected among those presented at ER 2004.

When reusing ontologies, many superfluous concepts are often included in the final conceptual schema. The first paper, entitled “A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems” by Jordi Conesa and Antoni Olivé presents a formal method of pruning ontologies to remove these superfluous concepts automatically.

The second paper, “XSLTGen: A System for Automatically Generating XML Transformation Via Semantic Mappings” by Stella Waworuntu and James Bailey, presents a method to automatically generate XSLT transformations based on the semantic mappings between input and output documents. Their experimental results show that the XSLTGEN works well with varieties of XML and HTML documents.

Based on the premise that semantically related data are highly likely to be changed as a result of the effort by the same or even different information sources for maintaining freshness and consistency, the third paper, “An Ontology-Guided Approach to Change Detection of the Semantic Web Data” by Li Qin and

Vijayalakshmi Atluri presents an approach that explores the relationship among concepts in guiding the change detection to their data instances for the Semantic Web.

The fourth paper, "Conceptual Modelling Patterns for Roles" by Jordi Cabot, and Ruth Raventós, studies role semantics in conceptual modeling and proposes a pattern based approach.

The selection from the ODBASE conference resulted in two extended papers being accepted for JoDS.

Relationships among concepts, namely inclusion dependencies, are also analyzed in the paper by Andreas Koeller and Elke A. Rundensteiner. The authors present heuristics to scale hypergraph-based inclusion dependencies discovery algorithms. Heuristics are based on the notion of inclusion dependencies between different relations of a database (or different databases) that are discovered by hypergraph-based algorithms, but that do not correspond to a real semantic relationship between such relations.

The paper by de Souza et al. describes a complete solution for the alignment of subdomain ontologies using an upper domain ontology that is built based on a thesaurus of terms. Mappings from the concepts of the individual ontologies to sets of thesaurus terms are established. A novel measure of similarity among concepts is also introduced together with suitable visualization techniques.

Finally, two of the selected papers from the ICSNW conference were accepted after rigorous review. The first one, by Bagüés et al., addresses "Semantic Interoperation Among Data Systems at a Communication Level." The authors propose to achieve semantic interoperability in a framework of agent-based data systems that exchange messages at a semantic level without requiring pre-established communication patterns. An ontology of communication acts is a key resource for this kind of interoperability. Semantic description of Web services and two case studies are also discussed.

The second paper, "Matching Ontologies in Open Networked Systems: Techniques and Applications," by S. Castano, A. Ferrara, S. Montanelli, and G. Racca, describes an algorithm and related techniques for performing matching of independent ontologies in open networked systems. A key feature is the capability of dynamically configuring the algorithm taking into account the complexity of the ontologies at hand. Implementation and experimental results are also presented.

The Guest Editors

ER 2004
Paolo Atzeni, Università Roma Tre, Italy
Wesley W. Chu, University of California Los Angeles, Los Angeles, USA

ODBASE 2004
Tiziana Catarci, Università di Roma "La Sapienza", Italy
Katia P. Sycara, Carnegie Mellon University, USA

ICSNW 2004
Stefano Spaccapietra, EPFL, Switzerland

Paolo Atzeni and Wesley Chu would like to dedicate this issue to the memory of Hongjun Lu, ER04 Program Co-chair, who passed away a few months after the conference, to which he had dedicated a lot of effort, especially in coordinating the overall program and the relationships between the Chinese and the international research community.

Reviewers

We are please to mention the reviewers who contributed their voluntary effort to the timely completion of this volume:

Karl Aberer, EPFL, Switzerland
James Baley, University of Melbourne, Australia
Roberta Benassi, Università di Modena e Reggio Emilia, Italy
Athman Bouguettaya, Virginia Tech, USA
Diego Calvanese, Free University of Bozen-Bolzano, Italy
Jordi Conesa, Universtat Politènica Catalunya, Barcelona
Jerome Euzenat, INRIA, France
Csilla Farkas, University of South Carolina,USA
Enrico Franconi, Free University of Bozen-Bolzano, Italy
Antonio L. Furtado, Pontificia Universidade Católica do Rio de Janeiro, Brazil
Jianming He, UCLA, USA
Lalana Kagal, MIT, USA
Hannu Kangassalo, University of Tampere, Finland
Andreas Koeller, Montclair State University, USA
Fabrice Jouanot, IMAG Grenoble, France
Domenico Lembo, Università di Roma "La Sapienza", Italy
Diego Milano, Università di Roma "La Sapienza", Italy
John Mylopoulos, University of Toronto, Canada
Antoni Olivé, Universitat Politècnica de Catalunya, Spain
Li Qin, Western New England College, USA
Sudha Ram, University of Arizona, USA
Pavel Shvaiko, University of Trento, Italy
Sergio Tessaris, Free University of Bozen-Bolzano, Italy
David Toman, University of Waterloo, Canada
Petko Valtchev, University of Montreal, Canada

The Guest Editors

JoDS Editorial Board

Table of Contents

Semantic Interoperation Among Data Systems at a Communication Level

Miren I. Bagüés*, Jesús Bermúdez, Arantza Illarramendi, Alberto Tablado, and Alfredo Goñi **

University of the Basque Country, Donostia, Spain

Abstract. The traditional perception of isolated data systems is changing to a new one where the interest of a real and efficient interoperation among those data systems is recognized. However, many problems must be solved yet before a real interoperation becomes true. In order to overcome the existing problems, there is a considerable number of proposals that can be found in the specialized literature that promote the idea of semantic interoperability.

A new framework to achieve semantic interoperability among agent based data systems at a communication level is proposed in this paper. This framework permits agents belonging to different data systems 1) to send each other suitable messages without requiring the establishment of a common communication pattern in advance; 2) to understand, completely or partially, those messages that are interchanged among themselves; and 3) to invoke web services offered by the data systems at a high level without needing to go into technical details. An ontology that captures the semantics of different communication acts is the key element for supporting the functionalities provided by the framework. Furthermore, the framework has been extended to support semantic descriptions of web services, which favor their automatic discovery. The usefulness of the presented framework is evaluated using two case study of interoperation among heterogeneous data systems; on one side through the agents of those systems, and on the other side, through the combination of an agent and a web service.

1 Introduction

The traditional perception of isolated data systems is changing to a new one where the interest of a real and efficient interoperation among those data systems is recognized. New advances in the areas of Internet and network communications are favoring somehow this change. However, many problems must be solved before a real interoperation becomes true. To that end, many proposals appear

* This work is supported by a grant of the Basque Government.

** All authors are members of the Interoperable DataBases Group, online at http://siul02.si.ehu.es. This work is mainly supported by the University of the Basque Country, Diputación Foral de Gipuzkoa (cosupported by the European Social Fund) and CICYT [TIN2004-07999-C02-00].

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 1–24, 2006.

in the specialized literature aiming at overcoming those existing problems. A great number of those proposals promote the idea of semantic interoperability.

Establishing semantic interoperability among heterogeneous and distributed data systems has been, and still is, a critical issue attracting significant attention from research and practical reality. That issue can be tackled from two main perspectives. From one side, the focus is on the data sources and how to provide a unified view of their underlying representational and reasoning formalisms for a semantic mediation process (sometimes the notion of *data integration* is used to refer to this perspective). Several proposals consider this perspective and can be distinguished according to: first, the type of mappings among the unified views and the schemas of the sources (Global as View *versus* Local as View); and second, the languages used for modelling the views and the sources. Among those proposals, we can mention [1, 2, 3, 4, 5, 6, 7]. From the other side, the focus is on the communication aspects and how to achieve communication at a semantic level in presence of different execution platforms when the data systems agree on the semantics of the interchanged data. So far, this second perspective has yielded less proposals (e.g. [8]). However, the relevance of the semantic communication problem is also widely accepted and this matter goes far beyond the use of XML for the interchange of data. The framework presented in this paper considers the latter perspective.

Moreover, nowadays there is a tendency to use agents in data systems because agent technology is broadly recognized as an appropriate technology for approaching problems which show a highly distributed nature and need flexible and adaptable solutions [9]. In this paper we focus on these kinds of agent based data systems.

There are two ways in which agent based data systems can interoperate among themselves. One, through messages that are interchanged among the agents of both systems, and two, using the web services that are provided by each data system. In this paper we present a proposal that considers both cases.

Currently the communication among agents is, in general, based on the interchange of messages. Agents must be aware in advance of the structure, language and the meaning of the messages in order to deal with them. Although this kind of communication is useful, it is also true that it is somehow limited because it forces agents to share the same communication pattern. Therefore, the interoperation of agents from different systems, independently developed, is extremely unlikely in this scenario. Moreover, nowadays interoperation is unconsidered when the agents follow different standards of communication languages. The framework proposed in this paper is used as a basis for automating the detection and resolution of conflicts that arise when dealing with messages interchanged by agents of different systems.

Furthermore, concerning the use of web services, the most used approach is to invoke them. In this case it is necessary to know in advance the number and type of the parameters of the invocation and the service capabilities. In addition, the Semantic Web Services framework is trying to soften those requirements. Semantic Web Services [10], as a new paradigm, is generally defined as the

augmentation of Web Service descriptions through Semantic Web annotations in order to facilitate the higher automation of service discovery, composition, invocation, and monitoring in an open and unregulated environment that is the Web. Several research activities in Semantic Web Services are emerging [11, 12, 13]. Our contribution in this paper with regard to web services concerns the framework we propose for a semantic communication among agents, in the sense that it can be easily adapted to support Semantic Web Services descriptions. This issue can help in the process of discovering dynamically and on the fly web services, and therefore facilitating the data systems interoperation using web services.

Taking into account the aforementioned difficulties when communicating different data systems through agents or web services, and being aware of the interest of the web services discovering process, we present in this paper a new framework that promotes communications among different data systems at a semantic level (the basics of this framework can be found in [14]) with the goal of overcoming those difficulties. The proposed framework is mainly based on the use of ontologies: a communication acts ontology, domain specific ontologies and an actions/operations ontology.

We have dedicated a considerable effort to develop the communication acts ontology (called CommOnt), so we present it more in detail. The desideratum for the other ontologies is to use the ones developed by specialists in the corresponding areas. We distinguish three categories in the CommOnt ontology: the *Actors* category that represents those entities that send or receive messages; the *Communication Acts* category that represents the messages which have different purposes and deal with different kinds of contents; and the *Contents* category that represents the kind of sentences included in the message. In the ontology there are axioms which describe the interrelationships among these categories. Moreover, for the sake of presentation we explain the *Communication Acts* category divided into three layers. The *upper layer* of the ontology, which include classes that describe general communication acts with the aim of being shared by any communication framework. The *standards layer* of the ontology which is devoted to general purpose communication languages, is specified as subclasses of the upper layer classes. Furthermore in the communication acts category appear classes of messages that different standards such as, FIPA-ACL [15] and KQML [16], have defined to use as communication acts (e.g. `FIPA-Inform` term for FIPA-ACL and `KQML-Tell` term for KQML). Finally in the *applications layer* all the classes described are directly related to a concrete data system that it is considered. Therefore, the first two layers may be shared by all data systems. Only the applications layer must be defined for each data system. Thus, for each data system, the classes of messages that it is capable of dealing with must be specified. If a data system can deal with a particular class `M` of messages then it can also deal with any message of a subclass of `M` in the CommOnt ontology. We claim that the whole communication acts ontology provides interoperability support due to the recognition of communication acts from one language as instances of communication acts in another language. Sometimes the "translation"

will not be complete, but partial comprehension of the communication may be useful and preferable to the "not understood" answer given nowadays. Reasoning support provided by the chosen formalism (OWL) [17] will help during the interaction process.

Domain specific ontologies, that is to say, ontologies that contain terms related to the domains considered by the data systems, are necessary in our framework to establish the semantics of the vocabulary used in the interoperation. Moreover, we advocate the importing of those ontologies that are built by experts in each domain. Finally, an operations/actions ontology which contains terms such as: getting, giving, transaction, etc. is used jointly with the domain specific ontologies, among other things, to describe web services semantically. Our proposal for web service descriptions follows a set-based modelling approach supported by a description logic (OWL-DL) formalism. This proposal is compatible with those approaches proposed in [18, 19].

In summary, the specific contributions of the work presented in this paper are:

- The design of one ontology, CommOnt, that contains terms associated to communication acts. Those terms of the ontology are divided into three layers. The first layer contains terms valid in any communication framework. The second layer contains terms related to the so called standard communications languages (e.g. FIPA). Finally, the third layer contains terms related to particular messages that handle the agents of each data system.
- The development of a framework for allowing interoperation among heterogeneous and distributed agent based systems, based on the CommOnt ontology. It is worth mentioning that this framework detects and resolves conflicts that appear in the communication process among agents using different agent communication languages (following the same standard or following different ones, FIPA and KQML for example). Our framework provides agents of the data systems with the following additional feature: *Understanding*. An agent can *understand*, completely or partially the message sent by an agent of another data system by following a reasoning process with the ontologies.
- Furthermore, the framework has been adapted to support on the fly, the automatic discovering of web services offered by the data systems. Taking the framework developed for the interoperation among agents of different data systems as the starting point, we have adapted it to provide semantic web service description and discovering capabilities.

The feasibility of our framework is demonstrated in two scenarios of data systems interoperation. In the first case, one agent of the Aingeru system (http://www.aingeru.com) interoperates with one agent of the HeCaSe system (http://grusma.etse.urv.es/hecase/index.htm). In the second case, one agent of the HeCaSe system deals with a web service provided by the Aingeru system.

In the rest of this paper we present first, in section 2, the main components which constitute the proposed framework and the steps that are followed when the communication among agents takes place. In section 3 we describe the main

features of the CommOnt ontology and we show how the other kinds of ontologies are used within our framework. In section 4 and 5, we respectively explain two cases of study that illustrate the usefulness of the proposed approach when agents of different systems communicate, and when agents and web services communicate. The applicability of the framework to describe and to discover dynamically web services are presented in section 6. Lastly, we finish with some related works and conclusions in sections 7 and 8.

2 Framework Components for Communicating Agent Based Data Systems

Following, we present a simple set theoretic model of the proposed framework which provides an abstract understanding of the relationships among the components, independently of implementation details.

In order to deal with our proposal, the following 7-tuple $\Gamma = (\Omega, \Delta, O, \tau(E), \tau(\Lambda), \pi, \lambda)$ of components should be incorporated at each node where a data system runs.

Let C, P, I, V be the sets of ontology concepts, properties, instances and literal values respectively. We call T to the set of OWL triples in the cartesian product $(C \cup I) \times P \times (C \cup I \cup V)$. Then,

- $\Omega \subseteq T$ is the CommOnt ontology where the classes of messages, contents and actors are designed. CommOnt has three layers; the two first layers can be imported from one repository and the application layer must be designed by the data system administrator;
- $\Delta \subseteq T$ is a *domain ontology* where the terms for referring to objects and properties of the application domain are specified. This ontology can be profitably imported from shared repositories;
- $O \subseteq T$ is an *ontology* describing suitable *operations* or *actions* to be performed by agents. This ontology can also be imported.

Messages are communicative actions expressing the sender's attitude toward some possible complex proposition. Therefore, a message has two main components. First, the attitude of the sender which expresses intentions such as informing, requesting or promising, for example. And second, the propositional content which is the subject of what the attitude is about.

Let $M = E \cup \Lambda$ be the vocabulary for writing messages; where E represents the vocabulary subset for writing the "envelope" of messages and Λ represents the vocabulary for the content language of messages. Then,

- $\tau(E) \subseteq E \times (C \cup P)$ is an *attribute mapping* that relates attributes from the envelope of messages with ontology terms;
- $\tau(\Lambda) \subseteq \Lambda \times (C \cup P)$ is an *attribute mapping* that relates content language features with ontology terms.

A key point of our approach is that every message has an abstract representation as an individual of a shared universal class of messages. Such representation

is founded on OWL [20] triples[1]. The Web Ontology Language OWL is a W3C recommendation that facilitates greater machine interpretability.

Let $\Phi \subseteq T \cap (I \times P \times (C \cup I \cup V))$ be the collection of OWL triples which express statements about instances; and let M^* and Λ^* be the sentences for messages and contents, respectively. Then,

- π: $M^* \xrightarrow{\tau(E),\Omega} \Phi \times \Lambda^*$ is a *platform mapping* that decompose messages into a collection of triples and in one content language expression.
- λ: $\Lambda^* \xrightarrow{\tau(\Lambda),\Omega,\Delta,O} \Phi$ is a *content language mapping* that applies content language expressions on the corresponding collection of triples;

Following we describe the steps followed during a communication process among agents from different data systems. There is a particular agent, called *CommOnt Manager* (CM) in charge of coordinating the process of dealing with the ontologies (Ω, Δ, O), and it plays the role of intermediary among agents from different data systems. There will be a *CommOnt Manager* associated within each data system. One main reason to include this *CommOnt Manager* agent in the framework is that only one agent needs to be a specialist in the process of dealing with the ontologies. However, this agent could be eliminated and the proposed framework would also be valid in order to allow direct communication among agents of different data systems (in this case each agent should deal with the ontologies). Let us suppose two systems Σ and Θ, and an agent A_Σ from the first system sending a message to an agent A_Θ from the second system. The process consists of the following steps (see figure 1):

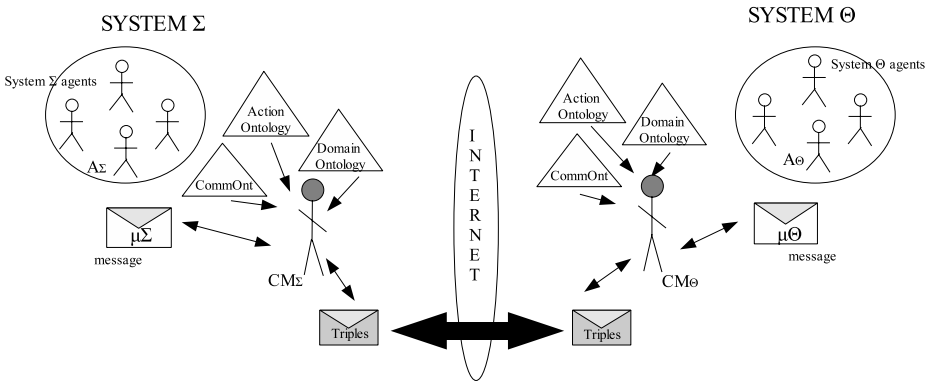


Fig. 1. Architecture of the proposal.

[1] More precisely, they are RDF [21] triples. But, since OWL is an extension of RDF, we highlight them as OWL triples.

1. The agent A_Σ creates the appropriate message $\mu\Sigma$ in the format used in the Σ system. Also creates and sends an $\epsilon\Sigma$ message requiring the CM_Σ agent to send the $\mu\Sigma$ message.
2. The CM_Σ agent transforms the message $\mu\Sigma$ into an equivalent collection of triples using $\pi_\Sigma(\mu\Sigma) = (\mu\phi, \mu\kappa)$ and $\lambda_\Sigma(\mu\kappa)$.
3. The CM_Σ agent sends the collection of triples to the CM_Θ agent.
4. The CM_Θ agent, with the help of ontology reasoners derives more triples and calculates the most specific class of the $\mu\Sigma$ message within the CommOnt ontology. Let us call it $msc(\mu\Sigma)$.
5. If CM_Θ recognizes $msc(\mu\Sigma)$ as an understandable message class for A_Θ, then it constructs a $\mu\Theta$ message out of the enriched collection of triples using[2] π_Θ^{-1} and λ_Θ^{-1}. A complete understanding is achieved in this case if the systems Σ and Θ share the domain and operation ontologies Δ and O.
6. In case of CM_Θ does not recognize $msc(\mu\Sigma)$ as an understandable class for A_Θ, it sends a message to CM_Σ advertising which are the subclasses of $msc(\mu\Sigma)$ that A_Θ is able to deal with. In this case, for the moment, only a partial understanding is achieved. However, CM_Σ can search in CommOnt for ontological relationships among those subclasses of $msc(\mu\Sigma)$ and classes of messages that A_Σ understands. Then CM_Σ has the opportunity to inform A_Σ of the capabilities of A_Θ just in terms of Σ messages. In the best case, A_Σ is able to reconstruct its original $\mu\Sigma$ message (or something similar) and renew the communication process.

3 Ontologies

As already mentioned, three different kinds of ontologies take part in the framework. Our main contribution is concerned with one of them; namely, the communication acts ontology called CommOnt. Therefore in this section we will explain it more in detail.

3.1 The CommOnt Ontology

Every ontology is designed with a specific purpose. The goal of our ontology is to achieve the interoperability of agent based systems (a more complete presentation of this CommOnt ontology that includes semantic issues related to the ontology terms can be found in [22]). For the sake of clarity the CommOnt ontology is divided into three interrelated layers: Upper layer, Standards layer and Applications layer.

Upper Layer Currently the communication among agents is, in general, based on the interchange of messages. The term `CommunicationAct`[3] represents the universal class of messages. Every concrete message is an individual of this class.

[2] π_Θ^{-1} and λ_Θ^{-1} are the inverse mappings of π_Θ and λ_Θ, respectively.
[3] We will use `this type` style to refer to terms specified in the ontology.

Moreover, we want to point out some parameters of messages that in this setting are represented by properties. The most immediate are the propositional content and the actors who send and receive the message.

Let us specify the class `CommunicationAct` using an abstract syntax in the following manner:

$$\begin{aligned}\texttt{CommunicationAct} \sqsubseteq\ & \forall \texttt{has-sender}.\texttt{Actor} \sqcap =1.\texttt{has-sender} \sqcap \\ & \forall \texttt{has-receiver}.\texttt{Actor} \sqcap \\ & \forall \texttt{has-content}.\texttt{Content}\end{aligned}$$

For the presentation we prefer this logic notation[4] instead of the more verbose OWL/XML notation. The sentence means that `has-sender`, `has-receiver` and `has-content` are properties that may be applied to communication acts. Every sender and receiver of an instance of the `CommunicationAct` class must be an individual of the class `Actor` and there is exactly one value filling the property `has-sender` for each instance of `CommunicationAct`. Moreover, every content must be an individual of the class `Content`.

The term `Content` in this upper layer represents the class of all possible contents of messages. So far, we have considered three direct subclasses of `Content`. First, propositions that can be viewed as statements and which are individuals of the class `Proposition`. Second, actions that agents are able to perform and which are individuals of the class `Action`. And third, referential expressions describing values from some domain which satisfies certain constraints and that we represent as individuals of the class `RefExpression`.

In the context of CommOnt, by `Actor` we refer to those entities sending or receiving messages. We have divided the category of actors into three subclasses: `WebService`, `Agent` and `Human`. Those subclasses respectively represent the universal class for the web services defined in the system, the class of agents that take part in the system and the human users that are going to interact with the system.

Furthermore, we have designed different specializations of the `CommunicationAct` class, taking into account the different primitive attitudes considered in the speech acts theory [24]. That theory has been developed in the linguistic area and such attitudes are called illocutionary acts. Among the different classifications of illocutionary acts that have appeared in the specialized linguistic literature, we have opted for the classification presented by K. Bach and R. M. Harnish in [25] because we found it more amenable to a formalization of the semantics in our context. In their proposal, Bach and Harnish divide illocutionary acts into six general categories. The communicative illocutionary acts: *constatives, directives, commissives* and *acknowledgements*; the conventional illocutionary acts: *effectives* and *verdictives*. For example, `Directive` is the class of those communication acts (representing *directives*) that expresses the sender's attitude towards getting the receiver to do something and his intention that his attitude be taken as a reason for the receiver's action. That is to say, the sender

[4] This notation is common in the description logics field. See [23] for a full explanation.

desires that the receiver do something influenced by the sender's desire. For example, "Get the values of the monitored vital signs". Moreover, some constraints are required for communication acts in those classes. For example, the content of every instance of the `Directive` class must be an action.

$$\texttt{Directive} \sqsubseteq \texttt{CommunicationAct} \sqcap \exists \texttt{has-content.Action}$$
$$\texttt{Constative} \sqsubseteq \texttt{CommunicationAct} \sqcap \exists \texttt{has-content.Proposition}$$

It is also reasonable to design particularizations of these classes. Keep in mind that software agents in our context are not prepared to interpret arbitrary communication acts (as is the case in natural language communication). They only recognize messages on the basis of the values of their properties. For instance, different classes of contents must be applied to some directives (from the class `Directive`) depending on their intention. A directive asking for information (`Inquiry`) should contain an action for reporting (`ReportAct`) about a referential expression (`RefExpression`) representing a query or a proposition (`Proposition`) to be checked as true or false. However a directive ordering the performance of another kind of action (`Request`) should contain a command (`Command`). More specialized classes should be included if necessary, and notice that disjointness of classes is not assumed unless stated explicitly or logically deduced from statements.

$$\texttt{Inquiry} \sqsubseteq \texttt{Directive} \sqcap \exists \texttt{has-content.ReportAct}$$
$$\texttt{Request} \sqsubseteq \texttt{Directive} \sqcap \exists \texttt{has-content.Command}$$
$$\texttt{Responsive} \sqsubseteq \texttt{Constative} \sqcap \exists \texttt{in-reply-to.Inquiry}$$

Finally, we cannot forget that any singular data system may use specific classes of messages, which will be particularizations of those upper classes with their specialized interpretations. Then, interoperation among agents of two systems using different kinds of messages will proceed through this upper classes of messages, functioning as a by default semantics of any message.

Standards Layer A standards layer extends the upper layer of the ontology with specific terms that represent classes of messages of general purpose agent communication languages, like those from KQML or FIPA-ACL.

Concentrating on FIPA-ACL we can observe that it proposes four primitive communicative acts [15]: *Confirm*, *Disconfirm*, *Inform* and *Request*. Analysis carried out on the FIPA primitive communicative acts permit us to specify *Inform* and *Disconfirm* as two different subclasses of the CommOnt `Constative`, and *Request* as a subclass of the CommOnt `Directive`, and the *Confirm* communicative act as a particular subclass of `Inform` message (which is a subclass of `Constative`). Then, we choose the term `FIPA-Confirm` to represent in CommOnt the class of every FIPA *Confirm* communicative act, and respectively the terms `FIPA-Disconfirm`, `FIPA-Inform` and `FIPA-Request`.

$$\begin{aligned}
\texttt{FIPA-Confirm} &\sqsubseteq \texttt{Inform} \\
\texttt{FIPA-Disconfirm} &\sqsubseteq \texttt{Constative} \\
\texttt{FIPA-Inform} &\sqsubseteq \texttt{Constative} \\
\texttt{FIPA-Request} &\sqsubseteq \texttt{Directive}
\end{aligned}$$

Furthermore, the rest of the FIPA communicative acts are derived from those primitive four, and so a classification of terms representing them in CommOnt can be established. For example, the communicative act *Query-if* is the act of asking another agent whether (it believes that) a given proposition is true. The sending agent is requesting the receiver to *Inform* it of the truthness of the proposition. The formal definition of the *Query-if* communicative act [15] specifies it as a *Request* of an action of the kind *Inform-if*. The agent which enacts an *Inform-if* will actually perform a standard *Inform* of whether or not a proposition is true. Then, we can specify them in CommOnt as follows:

$$\begin{aligned}
\texttt{FIPA-Query-If} \equiv\ & \texttt{Inquiry} \sqcap \texttt{FIPA-Request} \sqcap \\
& \texttt{=1.has-content.FIPA-Inform-If} \\
\texttt{FIPA-Inform-If} \equiv\ & \texttt{ReportAct} \sqcap \\
& \exists \texttt{has-content}^{-1}.\texttt{FIPA-Request} \sqcap \\
& \forall \texttt{has-query.Proposition}
\end{aligned}$$

Analogously, communication acts from KQML are analyzed and the corresponding terms in CommOnt are specified. For example,

$$\begin{aligned}
\texttt{KQML-Tell} &\sqsubseteq \texttt{Constative} \\
\texttt{KQML-Ask-If} &\sqsubseteq \texttt{Inquiry} \sqcap \\
& \forall \texttt{has-content}.(\forall \texttt{has-query.Proposition})
\end{aligned}$$

It is of vital relevance for the interoperability aim to be able to specify ontological relationships among classes of different standards. For instance, when a FIPA agent wants to request another agent to achieve one goal G, the agent will send a message of class `FIPA-Request`, with a content expression referring to an action `achieve` (from an ontology of actions) and with the goal G as a parameter. Instead, a KQML agent trying to communicate the same thing will send a message of class `KQML-Achieve` with the goal G in the content expression. Fortunately, that relationship among FIPA and KQML communications acts can be expressed in the CommOnt ontology by the following axiom:

$$\texttt{KQML-Achieve} \equiv \texttt{FIPA-Request} \sqcap \exists \texttt{has-content.Achieve}$$

This is a simple example but shows the flexibility of the proposed framework.

Applications Layer It is often the case that every single agent based data system uses a limited collection of communication acts that constitute its particular agent communication language. The CommOnt ontology provides a framework

for the description of the nuances of such communication acts. Some of those communication acts can be defined as particularizations of existing terms in the standards layer and maybe some others as particularizations of upper layer terms. Nevertheless their specification in terms of the CommOnt ontology will favor the interoperability among agent based data systems.

We are going to present terms for this layer using the concrete system Aingeru[5]: an agent based data system for a new way of tele-assistance for elderly people [26]. The Aingeru system, apart from supporting the functionalities provided by current tele-assistance services, also offers: an active assistance by using agents that behave in the face of anomalous situations without a direct intervention of the user; an anywhere and at any time assistance by using wireless communications and PDAs (Personal Digital Assistant); and the monitoring of personal vital signs by using sensors that capture the values of those signs and feed a decision support system that analyzes them and generates an alarm when necessary.

After completing the requirements analysis of the system, three major groups of messages were identified. One group includes the messages demanding the receiver to perform an action. Another group includes the messages asking the receiver for some information. And a third group, includes the messages sending results in reply to some request. Consequently, the classes of messages in each group were defined as subclasses of `Request`, `Inquiry` and `Responsive` respectively. Those were previously defined at the upper layer of the ontology. The distinctive features of Aingeru messages appear as constraints in their content.

Furthermore, when explaining the upper layer, we have mentioned the class `Content` and its subclasses `Proposition`, `RefExpression` and `Action` (see figure 2(b) for specializations of the subclass `Action`). An important property that applies to instances of the class `Content` within the CommOnt ontology, is `has-subject`. It relates contents to instances in classes of domain ontologies (e.g. Sanitary, Traveling, E-commerce). In CommOnt, there exists the class `Subject` that represents the top level superclass of any class in a domain ontology (see figure 2(a)). The following axiom expresses that the class `Content` is the domain of the property `has-subject` and the class `Subject` is its range.

$$\texttt{has-subject} \sqsubseteq \texttt{Content} \times \texttt{Subject}$$

Taking into account the former considerations, now we are prepared to grasp the nature of term definitions that correspond to some messages of the Aingeru system. For example, an `A-MedicineModify`[6] message is used to request a change in the medicines prescription: Thus, a particular kind of action (from the class `Overwrite` in this case) must fill the content of the message. An `A-LocationQuery` message asks for the coordinates of the physical location of a user, and an `A-VitalSignInform` message replies to a request of the values of the vital signs of a person. Moreover, all of them have only one content.

[5] Aingeru is the word in the Basque language for expressing the notion of a guardian angel.

[6] The `A-` prefix intends to label the Aingeru messages.

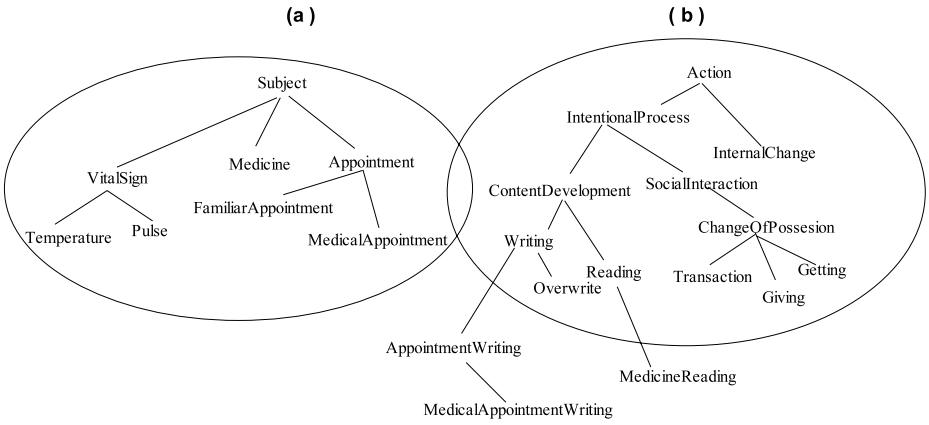


Fig. 2. Sample diagram of the Domain and Action ontologies.

$$\texttt{A-MedicineModify} \equiv \texttt{Request} \sqcap \texttt{=1.has-content} \sqcap \forall \texttt{has-content}.(\texttt{Overwrite} \sqcap \exists \texttt{has-subject.Medicine})$$

$$\texttt{A-LocationQuery} \equiv \texttt{Inquiry} \sqcap \texttt{=1.has-content} \sqcap \forall \texttt{has-content}.(\forall \texttt{has-query}.(\texttt{RefExpression} \sqcap \exists \texttt{has-subject.Location}))$$

$$\texttt{A-VitalSignInform} \equiv \texttt{Responsive} \sqcap \texttt{=1.has-content} \sqcap \forall \texttt{has-content}.(\texttt{Proposition} \sqcap \exists \texttt{has-subject.VitalSignData})$$

This kind of representation describes necessary and sufficient features of the structure of messages. Finally, in figure 3 we show a diagram for classes in the CommOnt ontology. Notice that in the figure appear also some specializations of the class `Actor` for the AINGERU system.

3.2 Domain Specific Ontologies and Actions Ontology

Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them. Consequently, they make that knowledge reusable [27]. Ontologies are used by people, data systems, and applications that need to share domain information (a *domain* is just a specific subject area or area of knowledge, like medicine, imagery, automobile repair, etc.). Our desire is not to describe in this paper the features of domain ontologies that are a result of consensus of domain experts. Here we are only going to show how they are used in our framework. In figure 2(a) we show a small part of a domain ontology. In that ontology we can see terms such as: `VitalSign`, `MedicalAppointment`.

Moreover an action ontology that contains terms that describe different actions and operations is also included in our framework. In figure 2(b)

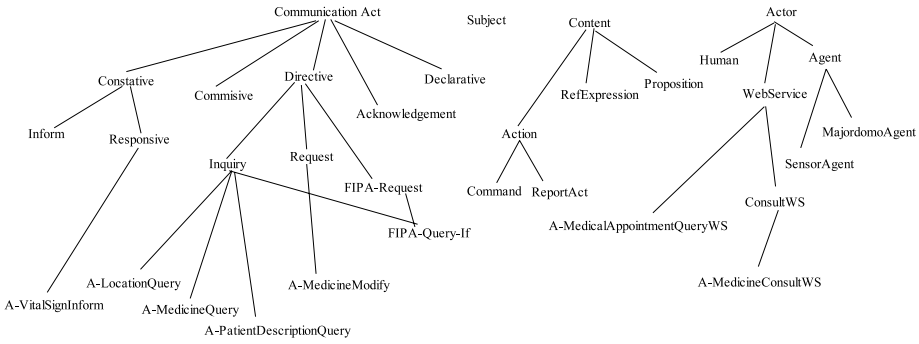


Fig. 3. Diagram of a fragment of CommOnt.

we show a small part of an action ontology (it corresponds to a subset of the Suggested Upper Merged Ontology (SUMO) that can be found in http://ontology.teknowledge.com/). In that ontology we can see terms such as: `Writing`, `Reading`.

Terms from both ontologies will be used to describe semantically web services, and also will be use by the agents of the data systems when they need to build messages.

For example, if an agent would want to obtain the medicines that someone is taking, then it would send a message that would include in its content "the action of reading medicines". That is, it would use the term `MedicineReading` that is composed of the term `Reading` from the Action ontology and the term `Medicine` from the Domain ontology to express its desire (see the following axiom).

$$\texttt{MedicineReading} \equiv \texttt{Reading} \sqcap \exists \texttt{has-subject.Medicine}$$

In the same way the terms `MedicalAppointmentWriting` and `AppointmentWriting` would express the action of writing a medical appointment or writing an appointment respectively (see the axioms).

$$\texttt{MedicalAppointmentWriting} \equiv \texttt{Writing} \sqcap \exists \texttt{has-subject.MedicalAppointment}$$
$$\texttt{AppointmentWriting} \equiv \texttt{Writing} \sqcap \exists \texttt{has-subject.Appointment}$$

4 Case Study: Communications Among Agents Through Messages

In this scenario, an HeCaSe agent asks for information about a user of the Aingeru system (we assume here that it has the adequate permissions to ask for such information). HeCaSe is a system that uses FIPA-ACL for agent communication; it is associated to the Agentcities[7] project and it is an agent-based

[7] http://www.agentcities.org/

application that provides medical services. Specifically, the HeCaSe agent wants to know conditions of the patient identified by 39900002 and therefore it requests the *CommOnt Manager* to send the message in figure 4(a).

```
(QUERY-REF
   :sender
      (agent-identifier
      :name Personal@grusma.agentcities.net)
   :receiver (set
      (agent-identifier
      :name MajordomoAgent@aingeru.com)
   :language message-codified
   :ontology sanitary
   :protocol FIPA-Query
   :content
      (iota
         :iota-left "?x"
         :iota-right
          (is-patient-description
            :patient-var "?x"
            :patient-attributes
               (set)
            :patient-conditions
               (set :id-medical "39900002"))
      )
)
```

(a)

```
<CommOnt:A-PatientDescriptionQuery>
  <CommOnt:has-sender>Personal@grusma.agentcities.net</CommOnt:has-sender>
  <CommOnt:has-receiver>MajordomoAgent</CommOnt:has-receiver>
  <CommOnt:has-content>
     <sanitary:patient-description rdf:ID="pd001">
        <sanitary:patient-var />
        <sanitary:patient-attributes />
        <sanitary:patient-conditions>
           <sanitary:id-medical>39900002</sanitary:id-medical>
        </sanitary:patient-conditions>
     </sanitary:patient-description>
  </CommOnt:has-content>
</CommOnt:A-PatientDescriptionQuery>
```

(b)

Fig. 4. (a) HeCaSe message. (b) Aingeru message

The *CommOnt Manager* associated to the HeCaSe system receives the message and submit it to a module in charge of doing a syntactic analysis helped by axioms in the CommOnt ontology. That module transforms it into a collection of triples. Individual objects are created when necessary and they are given unique identifiers. For example `m1` represents the message to be sent to the Aingeru system. Then the header `QUERY-REF` permits the generation of the triple `m1 <type> FIPA-Query-Ref`. Due to the axiom `FIPA-Query-Ref`$\equiv$ `Inquiry` $\sqcap$ `FIPA-Request` $\sqcap$ `=1.has-content.FIPA-Inform-Ref`, the sentence following the tag `:content` is recognized as a `FIPA-Inform-Ref`, yielding the triples `m1 <has-content> m1C` and `m1C <type> FIPA-Inform-Ref`. Next, the axiom `FIPA-Inform-Ref`$\equiv$ `ReportAct` $\sqcap$ $\exists$ `has-content`$^{-1}$`.FIPA-Request` $\sqcap$ $\forall$ `has-query.RefExpression` helps in the generation of triples like `m1C <has-query> m1CQ`, `m1CQ <type> RefExpresion`, and so on. A detailed description of how the parser module works goes out of the scope of this paper.

Following we summarize some of the corresponding triples:

```
m1 <type> FIPA-Query-Ref
m1 <has-content> m1C
m1C <type> FIPA-Inform-Ref
m1C <has-query> m1CQ
m1CQ <type> RefExpresion
m1CQ <has-subject> m1CQS
m1CQS <type> PatientDescription
m1CQS <patient-conditions> PC
PC <id-medical> 39900002
```

Then, the *CommOnt Manager* of the HeCaSe system sends the collection of triples to the *CommOnt Manager* of the Aingeru system. When the Aingeru *CommOnt Manager* receives the collection, it asserts the triples in the local facts box of the reasoning system and asks the reasoner to proceed. Due to the presence of the following axioms in the Aingeru fragment of the CommOnt ontology:

$$\begin{aligned}
\texttt{FIPA-Query-Ref} &\equiv \texttt{Inquiry} \sqcap \texttt{FIPA-Request} \sqcap \\
&\quad\ \texttt{=1.has-content.FIPA-Inform-Ref} \\
\texttt{FIPA-Inform-Ref} &\equiv \texttt{ReportAct} \sqcap \\
&\quad\ \exists \texttt{has-content}^{-1}\texttt{.FIPA-Request} \sqcap \\
&\quad\ \forall \texttt{has-query.RefExpression} \\
\texttt{A-PatientDescriptionQuery} &\equiv \texttt{Inquiry} \sqcap \texttt{=1.has-content} \sqcap \forall \texttt{has-content.}(\\
&\quad\ \forall \texttt{has-query.}(\texttt{RefExpression} \sqcap \exists \texttt{has-subject.PatientDescription}))
\end{aligned}$$

the reasoner will derive the following triples:

```
m1 <type> Inquiry
m1 <type> =1.has-content.FIPA-Inform-Ref
m1C <type> FIPA-Inform-Ref
m1C <type> ∀has-query.RefExpression
m1 <type> A-PatientDescriptionQuery
```

We want to stress here that axioms stated with the $\equiv$ symbol express necessary constraints and sufficient conditions for the individuals in the named class to the left of the equivalence symbol (i.e. $C \equiv D$ means that $C \sqsubseteq D$ and $D \sqsubseteq C$). Notice that it is also possible to take advantage of the formalism to state axioms that only specify minimal sufficient conditions to recognize that an individual belongs to a certain class. Using the expressiveness power of the owl formalism and the supported reasoning capability, it is possible to discover the most specific class of a message and therefore helping agents to recognize certain messages that were not explicitly created as one of its owner classes.

Now, the message `m1` has been recognized as an instance of an Aingeru message class. In order for `m1` to be completely understood by an Aingeru agent, the domain ontology used for terms in the `m1` subject must be a shared domain ontology by the HeCaSe and the Aingeru agents.

Then, the *CommOnt Manager* of the Aingeru system can create the message shown in figure 4(b), which the MajordomoAgent of the Aingeru system will understand completely.

5 Case Study: Communications Among Agents and Web Services

In this scenario we want to show how our framework can be also used when agents and web services communicate.

Let us suppose that an agent of the HeCaSe system wants to use the `A-MedicalAppointmentQueryWS` web service of the Aingeru system in order

```
(QUERY-REF
   :sender  (agent-identifier  :name Personal@grusma.agentcities.net)
   :receiver  (set   (agent-identifier :name A-MedicalAppointmentQueryWS@aingeru.com))
   :language message-codified
   :ontology sanitary
   :protocol FIPA-Query
   :content
      (iota
         :iota-left "?x"
         :iota-right
          (is-visit-description
            :visit-var "?x"
            :visit-attributes  (set)
            :visit-conditions
               (set :id-medical "39900002")
               (set :visit-date "2005/3/30"))))
```

Fig. 5. HECASE message.

to know which are the medical appointments that the patient identified by 39900002 has on March 30th of 2005. We are assuming here that the agent of the HeCaSe system knows that this concrete web service exists because it has previously made a discovering process. The agent will request the *CommOnt Manager* of the HeCaSe system to send the message shown in figure 5 (it knows how to send messages but it does not know how to invoke web services). The *CommOnt Manager* of the HeCaSe system will transform the message into a collection of triples:

```
m2 <type> FIPA-Query-Ref
m2 <has-sender> Personal@grusma.agentcities.net
m2 <has-receiver> A-MedicalAppointmentQueryWS
m2 <has-content> m2C
m2C <has-query> m2CQ
m2CQ <type> RefExpresion
m2CQ <has-subject> m2CQS
m2CQS <type> VisitDescription
m2CQS <visit-conditions> VC
VC <id-medical> 39900002
VC <visit-date> 2005/3/30
```

Then, the *CommOnt Manager* of the HeCaSe system will send the collection of triples to the *CommOnt Manager* of the Aingeru system. Once the *CommOnt Manager* of the Aingeru system receives the triples, it will try to construct the corresponding invocation of the web service.

Due to the triple `m2 <has-receiver> A-MedicalAppointmentQueryWS` the *CommOnt Manager* of the Aingeru system will recognize the web service to be invoked and it will retrieve its interface description. Then, applying the platform mapping (π^{-1}) and the content language mapping (λ^{-1})(see section 2) to the collection of triples, particularly to `m2CQS <visit-conditions> VC`, `VC <id-medical> 39900002` and `VC <visit-date> 2005/3/30`, the *CommOnt Manager* is able to construct the service invocation shown in figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <A-MedicalAppointmentQueryWS soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <patient-id xsi:type="xsd:string">39900002</patient-id>
   <appointment-date xsi:type="xsd:string">1112281811153</appointment-date>
  </A-MedicalAppointmentQueryWS>
 </soapenv:Body>
</soapenv:Envelope>
```

Fig. 6. Web Service invocation

Notice that the property `id-medical` is in correspondance to the attribute `patient-id` and the property `visit-date` is in correspondance with the attribute `appointment-date`. Furthermore, the date March 30th of 2005 has been properly transformed into the codified expression 1112281811153, as required by the web service interface.

At this point we want to mention an opinion that appears in [28] "Web Services might evolve into software agents" with which our proposed framework fits well.

6 Discovering Web Services

In the area of Web Services, the industry has strongly embraced the use of SOAP (Simple Object Access Protocol) [29] and WSDL (Web Services Description Language) [30]. Both are simple open standards with plenty of available tools, most of which are useful for other purposes and are becoming de facto standards in themselves. Moreover, UDDI (Universal Description, Discovery and Integration) [31] is used to discover Web Services. However, UDDI does not provide service descriptions and the automated search supported is severely restricted. UDDI fails in the semantic description of services necessary for automated search.

OWL-S [32] provides a solution to the problem of providing semantics for distributed search of Web services. However, it has one weakness: it currently only allows service providers to specify pre and postconditions.

In this scenario we propose, as other approaches do (e.g. UBL[33], [34]), to use an ontology of shared terms in order to provide semantics for service operations.

If we add to our framework an ontology for describing Web Services (see figure 7), we can observe that the proposed framework can be used to discover dynamically and on the fly, the web services.

Let us explain this feature in more detail. Assuming that there exists a standard WS Ontology, its terms will constitute the upper layer specialization of the class `WebService` represented in figure 3. Then the WS Ontology administrator of each data system should describe the offered web services as subclasses of those defined at the upper layer. Moreover, terms from the domain ontology and

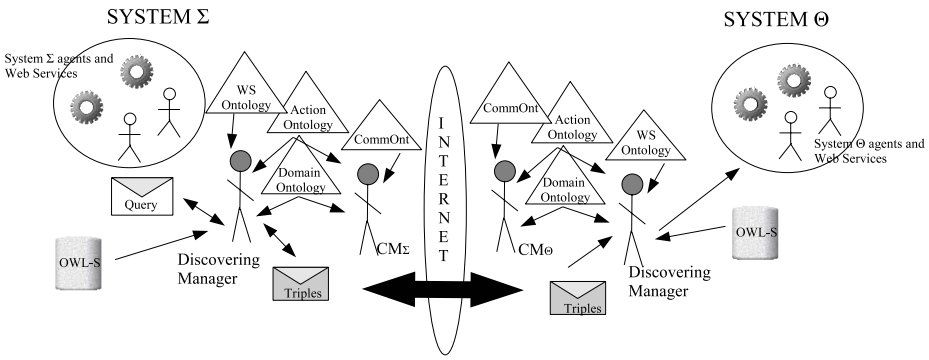


Fig. 7. Architecture of the proposal.

terms of the actions ontology will be used to describe the operations that are performed by the Web Services.

In the case of the AINGERU system, among others, it offers the web service called `A-MedicineConsultWS`. This web service informs about the medicines that a user of the AINGERU system has to take. Let us suppose that in the standard web services ontology exists a class that describes a general web service called `ConsultWS`, and that the class `MedicineReading` from the Action ontology is described as previously: $\texttt{MedicineReading} \equiv \texttt{Reading} \sqcap \exists\ \texttt{has-subject.Medicine}$

The Web Service `A-MedicineConsultWS` offered by AINGERU will be described as:

$$\texttt{A-MedicineConsultWS} \equiv \texttt{ConsultWS} \sqcap \texttt{=1.has-capability} \sqcap \forall \texttt{has-capability.MedicineReading}$$

In this framework we also add a *Discovering Manager*, which is in charge of dealing with Web Services. Moreover, in order to have a full description of the web services of the system, we have included an OWL-S repository. Each web service is described in the OWL-S description document, and its Service Profile part is augmented with a new property called `has-description` which points to the class of the WS Ontology that describes semantically that web service.

Now, we are going to show the use of this framework for discovering Web Services with an example. When an external agent, for example one of the HECASE system, wants to find an AINGERU Web Service which informs him about "the medicines that a user of the AINGERU system has to take", it would ask the *Discovering Manager* of the HECASE system to find a `ConsultWS` whose service is `Reading` related to `Medicine`. The *Discovering Manager* of the HECASE will construct an instance `sw1` with the minimal properties required by the agent. Then it will send this collection of triples to the *Discovering Manager* of the AINGERU system.

```
sw1 <type> ConsultWS
sw1 <has-capability> sw1C
sw1C <type> Reading
sw1C <has-subject> sw1CS
sw1CS <type> Medicine
```

When the *Discovering Manager* of AINGERU receives that collection it asserts the triples in the local facts box of the reasoning system and asks the reasoner to find all the classes to which `sw1` belongs to. Then, the *Discovering Manager* of AINGERU will reply to the *Discovering Manager* of HECASE with the classes found (there can be more than one web service that fulfill the needed functionality). Particularly the `A-MedicineConsultWS` class will appear in the answer. Once the agent of the HECASE system knows the web services that can fulfill its requirements, it selects one and it can use our framework to invoke that web service or the traditional web services techniques in order to discover the parameters and the structure of invocation.

7 Related Works

To achieve semantic interoperability among data systems is a critical issue for enterprises, organizations, etc. and can yield large profits for them. As we have mentioned in the introduction, that issue can be tackled from two different perspectives.

From one perspective, the focus is on the data sources, that is, how to provide a unified view of their underlying representational and reasoning formalisms for a semantic mediation process. In that optic, research in semantic interoperability can be categorized into three broad areas [1]: mapping-based approach (constructing a federated schema and establishing mappings between that schema and the participating local schemas, e.g. [35]); intermediary-based approach (use of mediator mechanisms to achieve interoperability, e.g. [2]); and, query-oriented approach (based on interoperable languages, e.g. [36]). Moreover, the work presented in [1] tries to take advantage of the works in many different areas, and overcoming at the same time their limitations.

From the other perspective, the focus is on the communication aspects, that is, how to achieve a communication at a semantic level in presence of different execution platforms. The work presented in this paper goes in this direction. Although less research works can be found in the specific literature that follows this perspective, different initiatives related to it are worth mentioning in order to frame our contribution. We group the related works into two groups. In the first, we include those related to agent communication aspects and in the second, those related to web services description.

Among the works of the first group, we start with the SUMO ontology (Suggested Upper Merged Ontology) that arises with the idea of promoting data interoperability, information search and retrieval, automated inferring and natural language processing. That ontology includes, among others, terms related to the communication acts. Those terms can be compared with the terms defined

in the upper layer of our CommOnt ontology; however, CommOnt contains a more exhaustive description of terms concerning communication acts and furthermore, a framework explaining how to deal with them is also provided in our case.

Continuing with the first group, our work is complementary to the development of standards for agent communication languages like KQML or FIPA-ACL (well summarized in [37]). These standards look for general homogeneity through compliance with the standard.

The closest work to ours, from the first group, is the approach defined in [8] that describes a Formal Language for Business Communications (FLBC) based on speech acts theory. That paper promotes a classification of messages that is further discussed in [38], and proposes a formal model of the message interpretation process that defines a standard way for systems to process them. Moreover, the same author presents in [39] a translation of KQML performatives to FLBC messages. However, our foundation on a formal extensible ontology and the use of an OWL representation distinguishes our approach clearly, and tackle more satisfactorily with some technical problems. FLBC messages are pure XML documents and therefore inherent difficulties emerge.

Finally in this first group we include, mechanisms that translate communications from one multiagent system to another. For instance, [40] describes an InterOperator that implements a connection between the RETSINA system (a KQML based system) and the OAA system. In our opinion the great effort needed to implement the mechanism for each concrete interoperation undermines the scalability of the approach. Moreover, the Open Agent Architecture (OAA) [41] is a framework for constructing multiagent systems and their designers intend to minimize the effort involved in wrapping legacy applications. Agent communications are represented as events. Each event has a type, a set of parameters, and a content. The allowable content and parameters vary according to the type of the event. Altogether they specify a class of messages in a similar way as descriptions promoted for the applications level in CommOnt.

Concerning the works of the second group, that is those related with web services description we can mention from one perspective the effort that is being dedicated to develop an ontological basis for e-business standards (UBL [33]), they are building the ontology from the SUMO Upper Ontology. In a similar way, in the OBELIX project [34] they have developed a generic component-based ontology for real-world services. That OBELIX service ontology is a formalization of concepts that represent the consensus in the business science literature on service management and marketing. Both works are complementary to ours and could be incorporated in our framework as a collection of upper level terms of the WS Ontology. From another perspective, we can mention the papers [42, 19] that propose to enhance the DAML-S (predecesor of OWL-S) description of web services and examine different degrees of matching between a service request and service advertisements, that could be incorporated in our framework. Both papers adopt, as we do, the set-based modelling approach. A more in depth analysis of web service discovery is being carried out within the WSMO project

[18]. In that project, in addition to the points considered by the previously mentioned works, they deal with richer descriptions that seem to be necessary for certain applications.

We want to finish this section noticing one relationship that we have found between the two mentioned perspectives tackling the semantic interoperability issue in the introduction. In [1] the authors of the CREAM system, that consider the data integration perspective, define 12 different message types for a semantic mediator communication protocol, namely ASK-ALL, ASK-IF, DELIVER, DETECT, GENERATE, LOCATE, RECONCILE, REPLY-ALL, REPLY-IF, REPORT, RESOLVE-IF and TELL. The protocol contains message types for encapsulating all basic communication events they might need. Those message types could be incorporated into our framework (that considers the semantic communication perspective) and so external agents that would ignore that concrete protocol and were interested in contacting the CREAM system, could interact with it using our proposed framework instead of the querying interface offered by CREAM.

8 Conclusions

Information technology has evolved from a focus on local data systems to more global interaction and integration within enterprises and communities. However, currently the interoperation among heterogeneous data systems is very restricted and there is a long way to go until a real and efficient interoperation is reached.

Two ways are being considered to obtain an actual global interaction: data integration and semantic communication. In the first case, the focus is on the data sources; how to provide a unified view of their underlying representational and reasoning formalisms for a semantic mediation process. In the second case, the focus is on the communication aspects; in other words, how to achieve communication at a semantic level in presence of different execution platforms when the data systems agree on the semantics of the data they interchange. Nevertheless, in both cases the need for explicit, machine-interpretable semantics is recognized and ontologies play the role of providing semantic explicitness that data systems need to interoperate with shared semantics and increasing semantic precision.

In this paper we have presented a framework based on the use of ontologies that favors the interoperation among heterogeneous data systems when communicating through agents or web services.

The main features of the proposed framework are summarized in the following:

- It promotes an explicit description of the communication acts that constitute the particular agent communication language of each system. This issue allows the sharing of knowledge related to the communication acts supported by the agents of each system and, in doing so, favors the interoperation among different data systems.
- It permits the communication among agents of different data systems that use different agent communication languages following the same standard

or a different one. Sometimes the understanding will not be complete but it may be preferable to the not understood answer given nowadays.
- It facilitates for agents the task of invoking web services, therefore handling technical details associated with that task. The framework also favors the evolution of web services into software agents.
- It is easily adapted to support semantic descriptions of the web services provided by the data systems. This issue favors the automatic discovering of those web services.

The future work is oriented in two directions. In one side on the implementation of related aspects. The idea is to incorporate to the existing prototype a friendly interface and to build a web site where the different modules involved in the framework will be arranged in order to be downloaded. On the other side, on extending the framework in order to deal with conversations (i.e. well-suited sequences of messages) among agents.

References

[1] Park, J., Ram, S.: Information systems interoperability: what lies benath? ACM Transactions on Information Systems **22** (2004) 595–632
[2] Goble, C.A., Stevens, R., Ng, G., Bechhofer, S., Paton, N.W., P. G. Baker, M.P., Brass, A.: Transparent access to multiple bioinformatics information sources. IBM Systems Journal **40** (2001) 532–551
[3] Mena, E., Illarramendi, A., Kashyap, V., Sheth, A.: 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) **8** (2000) 223–272
[4] Arens, Y., Knoblock, C.A.: Intelligent caching: Selecting, representing and reusing data in an information server. In: Proceedings of the Third International Conference on Information and Knowledge Management CIKM. (1994)
[5] Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: Integrating and accessing heterogeneous information sources in TSIMMIS. In: Proceedings of the AAAI Symposium on Information Gathering. (1995) 61–64
[6] Carey, M., et. al.: Towards heterogeneous multimedia information systems: The garlic approach. In: Proceedings of the 5th International Workshop on Research Issues in Data Engineering- Distributed Object Management (RIDE-DOM 1995). (1995) 124–131
[7] Ziegler, P., Dittrich, K.R.: User-specific semantic integration of heterogeneous data: The SIRUP approach. In: International Conference on Semantics of the Networked World: Semantics for Grid Databases (ICSNW 2004). LNCS 3226. (2004) 44–64
[8] Moore, S.A.: A Foundation for Flexible Automated Electronic Communication. Information Systems Research **12** (2001) 34–62
[9] Jennings, N.R.: An agent-based approach for building complex software systems. Commun. ACM **44** (2001) 35–41
[10] Payne, T., Lassila, O.: Semantic web services. IEEE Intelligent Systems **19** (2004) 14–15

[11] Burstein, M.H.: Dynamic invocation of semantic web services that use unfamiliar ontologies. IEEE Intelligent Systems **19** (2004) 67–73
[12] Sirin, E., Parsia, B., Hendler, J.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intelligent Systems **19** (2004) 42–49
[13] Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K.: Authorization and privacy for semantic web services. IEEE Intelligent Systems **19** (2004) 50–56
[14] Bagüés, M.I., Bermúdez, J., Tablado, A., Illarramendi, A., Goñi, A.: A new mechanism for the interoperability of data systems. In: International Conference on Semantics of a Networked World: Semantics for Grid Databases (ICSNW 2004). LNCS 3226. (2004) 229–247
[15] Foundation For Intelligent Physical Agents: FIPA Communicative Act Library Specification. (2002) http://www.fipa.org/specs/fipa00037/SC00037J.html.
[16] Finin, T., Labrou, Y., Mayfield, J.: KQML as an agent communication language. In Bradshaw, J., ed.: Software Agents. MIT Press (1997)
[17] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuiness, D., Patel-Schneider, P., Stein, L.: OWL Web Ontology Language Reference. World Wide Web Consortium. (2004) http://www.w3.org/TR/owl-ref.
[18] Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., Fensel, D.: WSMO Web Service Discovery. Working draft, WSML Working Group (2004) http://www.wsmo.org/2004/d5/d5.1/v0.1.
[19] Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, ACM Press (2003) 331–339
[20] Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. Recommendation, World Wide Web Consortium (2004) http://www.w3.org/TR/owl-semantics/.
[21] Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. Recommendation, World Wide Web Consortium (2004) http://www.w3.org/TR/rdf-concepts/.
[22] Bermúdez, J., Goñi, A., Illarramendi, A., Bagüés, M.I., Tablado, A.: Interoperation among information systems through a communications acts ontology. Technical report, University of the Basque Country (2005)
[23] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Theory, Implementation and Applications. Cambridge University Press (2003)
[24] Searle, J.R.: Speech acts. Cambridge University Press (1969) New York.
[25] Bach, K., Harnish, R.M.: Linguistic Communication and Speech Acts. MIT Press (1979)
[26] Tablado, A., Illarramendi, A., Bagüés, M.I., Bermúdez, J., Goñi, A.: Aingeru: an innovating system for tele assistance of elderly people. The Journal on Information Technology in Healthcare **2** (2004) 205–214
[27] Obrst, L.: Ontologies for semantically interoperable systems. In: Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, ACM (2003) 366–369
[28] Petrie, C., Bussler, C.: Service agents and virtual enterprises: A survey. IEEE Internet Computing **7** (2003) 68–78
[29] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H.: Simple object access protocol. Recommendation, World Wide Web Consortium (2003) http://www.w3.org/TR/soap/.

[30] Chinnici, R., Gudgin, M., Moreau, J., Schililmmenr, J., Weerawarana, S.: Web Services Description Language (WSDL). Working draft, World Wide Web Consortium (2004) http://www.w3.org/TR/wsdl20/.
[31] OASIS UDDI Specification Technical Comitee: Uddi version 2 specification. Technical report, OASIS (2002) http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.
[32] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., et al.: Owl-s: Semantic markup for web services. Technical report, World Wide Web Consortium (2004) http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
[33] OASIS Universal Business Language (UBL) Technical Committee: OASIS Universal Business Language 1.0. Technical report, OASIS (2004) http://docs.oasis-open.org/ubl/cd-UBL-1.0/.
[34] Akkermans, H., Baida, Z., Gordijn, J., Peña, N., Altuna, A., Laresgoiti, I.: Value webs: Using ontologies to bundle real-world services. IEEE Intelligent Systems **19** (2004) 57–66
[35] Kashyap, V., Sheth, A.P.: Semantic and schematic similarities between database objects: A context based approach. The Very Large Databases Journal **5** (1996) 276–304
[36] Krishnamurthy, R., Litwin, W., Ken, W.: Language features for interoperability of databases with schematic discrepancies. In Clifford, J., King, R., eds.: ACM SIGMOD International Conference on Management of Data, New York, USA, ACM Press (1991) 40–49
[37] Labrou, Y.: Standardizing agent communication. In Marik, V., Stepankova, O., eds.: Multi-Agent Systems & Applications. Advanced Course on Artificial Intelligence (ACAI-01), Springer-Verlag (2001) 74–97
[38] Moore, S.A.: Categorizing automated messages. Decision Support Systems **22** (1998) 213–241
[39] Moore, S.A.: KQML and FLBC: Contrasting agent communication languages. International Journal of Electronic Commerce **5** (2000) 109–124
[40] Giampapa, J.A., Paolucci, M., Sycara, K.: Agent interoperation across multiagent system boundaries. In: Proceedings of the Fourth International Conference on Autonomous Agents, ACM Press (2000) 179–186
[41] Martin, D.L., Cheyer, A.J., Moran, D.B.: The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence **13** (1999) 91–128
[42] Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proceedings of the First International Semantic Web Conference (ISWC2002), Springer Verlag (2002) 333–347

Matching Ontologies in Open Networked Systems: Techniques and Applications

Silvana Castano, Alfio Ferrara, and Stefano Montanelli

Università degli Studi di Milano
DICo - Via Comelico, 39, 20135 Milano - Italy
{castano,ferrara,montanelli}@dico.unimi.it

Abstract. In open networked systems a varying number of nodes interact each other just on the basis of their own independent ontologies and of knowledge discovery requests submitted to the network. Ontology matching techniques are essential to enable knowledge discovery and sharing in order to determine mappings between semantically related concepts of different ontologies. In this paper, we describe the H-Match algorithm and related techniques for performing matching of independent ontologies in open networked systems. A key feature of H-Match is that it can be dynamically configured for adaptation to the semantic complexity of the ontologies to be compared, where the number and type of ontology features that can be exploited during the matching process is not known in advance as it is embedded in the current knowledge request. Furthermore, this number can vary, also for the same ontologies, each time a new matching execution comes into play triggered by a knowledge request. We describe how H-Match enforces this capabilities through a combination of syntactic and semantic techniques as well as through a set of four matching models, namely *surface*, *shallow*, *deep*, and *intensive*. Then, we describe the application of H-Match and its implementation for knowledge discovery in the framework of the Helios peer-based system. Finally, we present experimental results of using H-Match on different test cases, along with a discussion on precision and recall.

1 Introduction

Open networked systems like Peer-to-Peer networks and Grids are becoming more and more semantics-enriched infrastructures enabling to share and create knowledge and to enforce semantic collaboration among the involved parties. For example, basic P2P networks adopting simple filenames for data sharing have been evolving to schema-based P2P networks, capable of supporting the exchange of complex resources like documents and services described by using metadata or thematic ontologies [1, 2]. P2P scientific collaboration networks have recently emerged to take advantage of the inherent properties of P2P networks in order to enforce scientific data sharing and to obtain better performance, flexible and efficient use of resources and system resilience [3, 4].

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 25–63, 2006.

In such systems, it is widely recognized that the use of ontologies plays a crucial role for providing a semantic description of the resources to be shared and for enhancing resource discovery through expressive queries [5]. A key feature of open networked systems is that the network organization can vary at any moment and a unique global ontology committed by all the parties is not a viable solution. Rather, a networked system is characterized by a multitude of independent *peer ontologies* autonomously made available by each node joining the system. Consequently, for knowledge discovery and sharing, a varying number of nodes interact each other just on the basis of their own independent ontologies and of knowledge discovery requests submitted to the network. In this context, appropriate ontology matching techniques are required to determine whether and how concepts of different ontologies are semantically related each other [6, 7]. The problem of schema and ontology matching has been investigated in the literature and a number of approaches and tools have been proposed in the area of data and knowledge management [7, 8, 9, 10, 11, 12]. A reference survey on schema matching is given in [13] while ontology matching is surveyed according to different classification frameworks in [14, 15, 16, 17].

Existing ontology matching approaches address a number of general requirements which remain very important in open networked systems. A first general requirement is the applicability to different ontology specification languages, with special attention to recent standards of the Semantic Web like OWL [18]. A further general requirement is the capability of coping with different levels of detail and design choices in describing the knowledge of interest using a certain language. In addition, the capability of considering different constructs used in ontology languages is required for matching purposes.

In addition, new peculiar requirements must be taken into account in conceiving ontology matching techniques for open networked systems. These requirements are originated by the dynamic behavior of peers in such a scenario. A first peculiar requirement is that the number and type of ontology features that can be exploited during the matching process is not known in advance as it is embedded in the current knowledge request. Furthermore, this number can vary, also for the same ontologies, each time a new matching execution comes into play triggered by a knowledge discovery request. Moreover, design principles of ontology matching techniques must be driven by i) the necessity of satisfying matching requests that are dynamically posed by peers on the basis of unexpected needs that can vary continuously, and ii) by the necessity of addressing all general and peculiar matching requirements as a whole.

In this paper, we present the H-Match algorithm and related techniques for matching independent ontologies in open networked systems. H-Match has been developed in the framework of the Helios peer-based system, where it is used to enable knowledge discovery and sharing [19, 20]. A key feature of H-Match is that it can be dynamically configured for adaptation to the semantic complexity of the ontologies to be compared, using a combination of syntactic and semantic techniques. This feature is achieved by means of four matching models, namely *surface*, *shallow*, *deep*, and *intensive* defined with the goal of providing a wide

spectrum of metrics suited for dealing with many different matching scenarios. Another distinguishing feature of H-MATCH is that the matching configuration is selected in an automated way according to a matching policy embedded in the incoming request.

In developing H-MATCH, we started from the schema matching functionalities of the ARTEMIS integration system [21]. From ARTEMIS we borrowed the thesaurus-based approach for name affinity management, and we made a number of extensions for matching linguistic features of ontology elements to provide a fully-automated approach. Furthermore, we have moved from the notion of structural affinity, typical of schema elements based on attributes, to the notion of contextual affinity, typical of ontology elements, based on semantic relations with explicit semantics, with consequent development of suitable techniques for contextual affinity. Moreover, we have introduced the notion of matching model and of configurability of the matching process through matching models. Finally, we want to remark that H-MATCH implements an automated ontology matching approach, since it has been conceived to enable the knowledge discovery process in open networked systems without any manual intervention. On the contrary, ARTEMIS enforces a semi-automated approach to schema matching being targeted to support the schema unification process in data integration systems with expected interaction with the designer.

Motivating and Running Example. In Figure 1, we show a graphical representation [1] of two simple ontologies, namely Apple-q and Apple-o[2]. They will be used as running example throughout the paper to show how the H-MATCH techniques work. The Apple-q ontology specifies that the concept Apple is a fruit and a kind of food. Apples, in this ontology, origin from Italy. The Apple-o ontology describes two kinds of products, namely Edible_fruits and Computer, where Mobile_Computer is a kind of computers. For fruits (i.e., Banana, Grape, and Pineapple), we have information about the provenance (i.e., Brazil for banana and pineapple, and Italy for grape). In this latter ontology, the concept Apple denotes a brand of computers like IBM, and it is located in USA. One of the challenging goals of ontology matching in this example is to capture the difference between the two Apple concepts, even if they have the same name. We have to evaluate whether the matching techniques are able to capture this difference on the basis of the different context that characterizes the two apple concepts in their respective ontologies. Other relevant matchings that should be found by the matching techniques are the ones between Food and Fruit of Apple-q and the concept of Edible_fruits in Apple-o, as well as between the two Italy concepts which denote the same region in the two ontologies. In the remaining of the paper, we will use this

[1] This graphical representation is based on H-MODEL, the formalism adopted by H-MATCH for internal representation of ontologies for matching (see Section 2.1).

[2] The OWL specification of the two ontologies is provided at http://islab.dico.unimi.it/ontologies/apple-q.owl and http://islab.dico.unimi.it/ontologies/apple-o.owl, respectively.

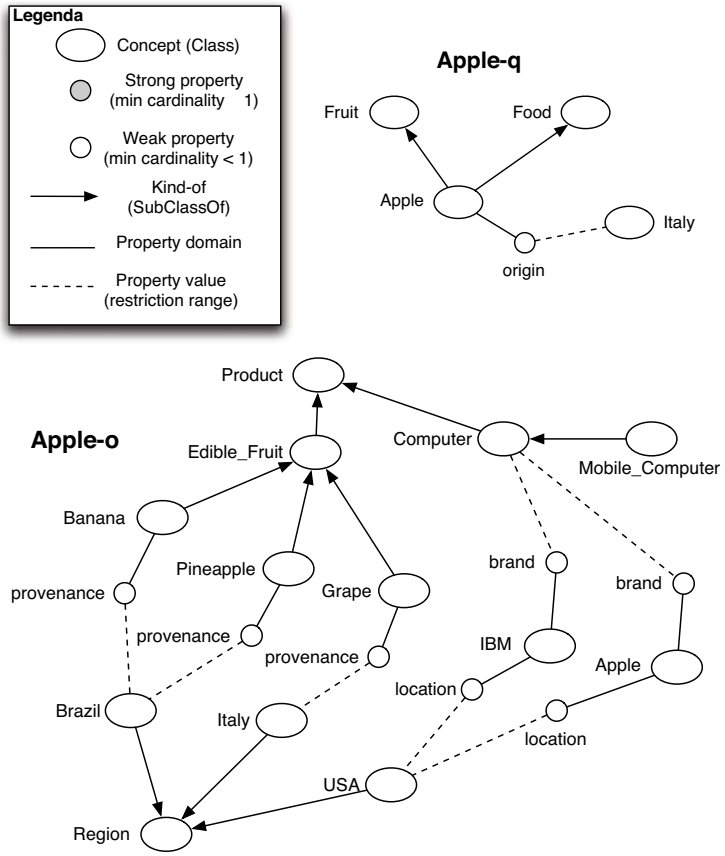


Fig. 1. H-Model graphical representation of Apple-q and Apple-o

example as the running example to show step by step how the matching process works.

Organization of the Paper. The paper is organized as follows. In Section 2, we give the foundations of the proposed ontology matching techniques. In Section 3, we describe the H-Match algorithm and related matching techniques with running examples. In Section 4, we describe the application of H-Match and its implementation for dynamic knowledge discovery in the framework of our open networked system Helios. In Section 5, we provide experimental results of applying H-Match and related matching techniques to Semantic Web ontologies test cases, by discussing the obtained results in terms of precision and accuracy. In Section 6, we make a critical comparison of H-Match with related work in the field of ontology matching. Finally, in Section 7, we give our concluding remarks.

2 Foundations of H-Match

We define *ontology matching* as a process that takes two ontologies as input and returns the mappings that identify corresponding concepts in the two ontologies, namely the concepts with the same or the closest intended meaning. We define a *mapping* as a correspondence between a concept of the first ontology and one or more concepts of the second ontology [3]. Ontology mappings are established after an analysis of the similarity of the concepts in the compared ontologies. In H-Match, we perform similarity analysis through affinity metrics to determine a measure of semantic affinity in the range $[0, 1]$. A threshold-based mechanism is enforced to set the minimum level of semantic affinity required to consider two concepts as matching concepts. With H-Match, it is possible to determine *one-to-one mappings* and *one-to-many mappings*. In a one-to-one mapping, a concept of the first ontology is associated with only one concept of the second ontology, namely the matching concept with the highest value of semantic affinity (also called *best-matching* concept). In a one-to-many mapping, a concept of the first ontology is associated with a set of concepts of the second ontology, namely all the selected matching (also called *best-k matching* concepts).

2.1 Ontology Representation

In H-Match, an ontology is seen as a set of concepts, properties, and semantic relations. For the sake of internal representation of ontology specification languages, and in particular for Semantic Web languages like OWL, we rely on a reference model, called H-Model. H-Model, as many other tools for ontology matching [4], provides a graph-based representation of ontologies in terms of concepts, properties, and semantic relations. In Figure 1, we show an example of H-Model representation of OWL ontologies. In this representation, the graph nodes denote concepts and properties (that in the example represent classes and properties of OWL), while the edges denote the semantic relations between concepts (that in the example represent the equivalence and subclass relations in OWL as well as properties domain and range derived by OWL restrictions). For a more detailed description of H-Model and supported ontology specification languages, the reader can refer to [20].

2.2 Semantic Complexity

The notion of semantic complexity has been introduced in [10] to describe different levels of complexity at which an ontology can be seen for matching purposes.

[3] In this respect, other approaches like [8, 10, 22], consider properties as first-class objects and, therefore, they find mappings also between them. We take an approach similar to [7, 12, 23] where mappings are found for concepts and properties are still matched but for the purpose of evaluating concept similarity.

[4] The state of the art ontology matching tools are analyzed with respect to the supported model for ontology representation in the related work of Section 6.

At each level, the focus is on the different types of constructs of the ontology specification. The four matching models defined in H-MATCH, namely *surface*, *shallow*, *deep*, and *intensive*, allow the matching process to enforce different levels of semantic complexity depending on the ontologies to be matched. In particular, the surface model is suitable for matching ontologies at the entity level, because it considers only names of ontology elements. The shallow model is suitable for ontologies that are semantic nets in that both names and concept properties are taken into account for matching. The deep and the intensive models are adequate for semantic complexity of the Description Logics languages, because they also take into account semantic relations and property values, respectively. H-MATCH performs matching by considering schema-level information of ontology descriptions. In other words, H-MATCH is focused the terminological box level of the description logics languages. This has been a design choice in developing H-MATCH, in order to support knowledge discovery in the HELIOS open networked systems [5].

2.3 Linguistic Features

Linguistic features refer to names of ontology elements and their meaning. To capture the meaning of names for ontology matching, we borrow from ARTEMIS [21] the idea of relying on a thesaurus of terms and weighted terminological relationships among them. In H-MATCH the thesaurus is automatically derived from the lexical system WordNet [24], to provide a common reference basis for all the peers of the system and to achieve a uniform interpretation of linguistic features as much as possible. To this end, we have introduced the following extensions to the ARTEMIS procedure motivated by the use of WordNet:

- Full use of the relations among synsets in WordNet, including not only synonymy and hypernymy/hyponymy, but also other relations provided by WordNet like meronymy and coordinate terms. Thesaurus construction can be configured in order to select the syntactic category to be taken into account. In the case of verbs, adjectives, and adverbs, the corresponding specific relations (e.g., troponymy for verbs) are considered.
- Automated management of compound terms not included in WordNet. In fact, names appearing in real ontologies often are formed by two or more terms originating compound terms that are not retrieved in WordNet.

The thesaurus is structured as a graph, where the nodes represent terms and the edges represent terminological relationships. Terms can be basic or compound. *Basic terms* are all those terms that are included in WordNet, composed by one or multiple tokens. *Compound terms* are all those terms composed by more than one token that are not included in WordNet. Terminological relationships represented in the thesaurus are SYN, BT, NT, and RT. SYN (synonymy)

[5] In Section 6, we discuss how currently available matching techniques of H-MATCH can be taken into account for the purpose of considering also instance-level information of ontology specifications.

denotes that two terms have the same meaning. BT (broader term) (resp., NT (narrower term)) denotes that a term has a more (resp., less) general meaning than another term. Finally, RT (related terms) denotes that two terms have a generic positive relationship. These terminological relationships are derived from the relations defined in WordNet during the thesaurus construction process, as described in Section 3. As in ARTEMIS, a weight W_{tr} is associated with each terminological relationship $tr \in \{\mathsf{SYN}, \mathsf{BT/NT}, \mathsf{RT}\}$ in the thesaurus. Such a weight expresses the implication of the terminological relationship for semantic affinity. Different types of relationships have different implications for semantic affinity, with $W_{\mathsf{SYN}} \geq W_{\mathsf{BT/NT}} \geq W_{\mathsf{RT}}$. In fact, synonymy is generally considered a more precise indicator of affinity than hierarchical relationships, consequently $W_{\mathsf{SYN}} \geq W_{\mathsf{BT/NT}}$. The lowest weight is associated with RT since it denotes a more generic relationship than the hierarchical relationships BT/NT.

2.4 Contextual Features

Contextual features of a concept c refer both to the properties and to the concepts directly related to c through a semantic relation in an ontology. The importance of considering contexts was already pointed out in [25] for matching heterogeneous information. It becomes mandatory for ontology matching especially in distributed contexts, where the meaning of a concept if often determined by the context where it is downhearted [17]. In Section 6, we provide a comparison of the state of the art ontology matching tools with respect to the contextual features that are supported.

Given a concept c, we denote by $P(c)$ the set of properties of c, and by $C(c)$ the set of *adjacents* of c, namely concepts that participate in a semantic relation with c, respectively. The context of a concept in H-MATCH is defined as the union of the properties and of the adjacents of c, that is, $Ctx(c) = P(c) \cup C(c)$. In H-MATCH, we distinguish between strong and weak properties. A strong property sp is a mandatory property with minimal cardinality 1. A weak property wp is an optional property with minimal cardinality 0. In H-MATCH we distinguish four semantic relations sr between two concepts c and c', namely same-as, kind-of, part-of, and associates. The same-as relation denotes that c and c' are equivalent, while the kind-of and part-of relations denote that c and c' are related by a specialization relation and a composition relation, respectively. Finally, the associates relation denotes that c and c' are related by a generic positive semantic association. In Figure 1, examples of how OWL constructs are mapped on such semantic relations are given.

Like linguistic features, also contextual features are weighted in H-MATCH. In particular, we associate a weight W_{sp} to strong properties, and a weight W_{wp} to weak properties, with $W_{\mathsf{sp}} \geq W_{\mathsf{wp}}$ to capture the different importance each kind of property has in characterizing the concept. In fact, strong properties are mandatory properties related to a concept and they are considered more relevant in contributing to concept description. Weak properties are optional for the concept in describing its structure, and, as such, are less important in featuring the concept than strong properties. Each semantic relation has associated a

weight W_{sr} which expresses the strength of the connection expressed by the relation on the involved concepts. The greater the weight associated with a semantic relation, the higher the strength of the semantic connection between concepts. For this reason, we define $W_{\mathsf{same-as}} \geq W_{\mathsf{kind-of}} \geq W_{\mathsf{part-of}} \geq W_{\mathsf{associates}}$.

3 Matching Ontologies with H-Match

H-Match combines a measure of linguistic affinity and a measure of contextual affinity in order to evaluate a comprehensive measure of semantic affinity between ontology concepts. The linguistic affinity provides a measure of similarity between the ontology concepts by considering their linguistic features, while the contextual affinity provides a measure of similarity by taking into account their contextual features. Four matching models, namely, *surface*, *shallow*, *deep*, and *intensive*, are defined for dynamically configuring H-Match for its adaptation to the semantic complexity of the ontologies to be compared. The H-Match matching process is shown in Figure 2. The process starts with the computation of the linguistic affinity among the ontology concepts, which is common to all matching models. Then a matching model is chosen. The context of concepts is

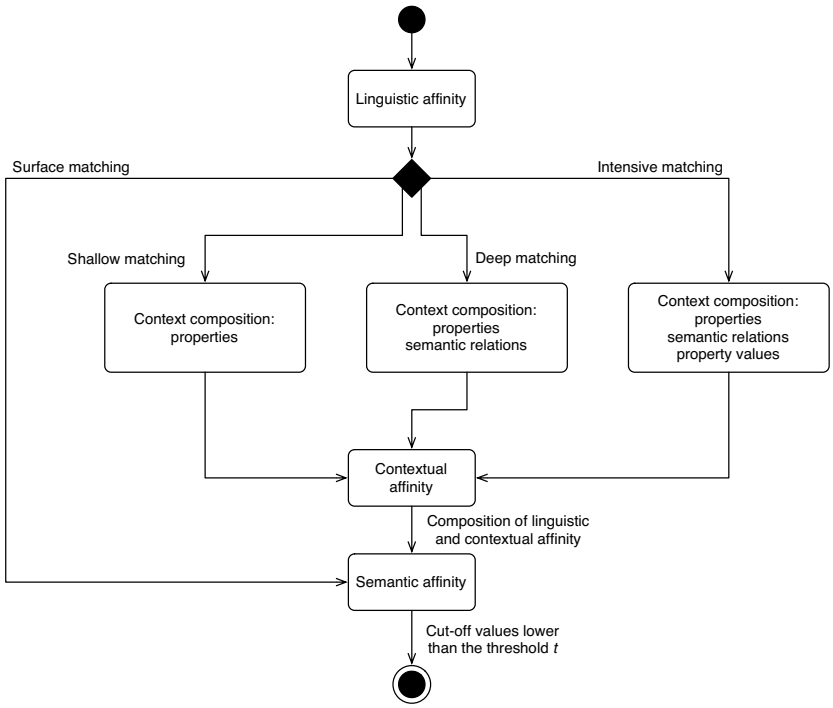


Fig. 2. The matching process of H-Match

then composed according to the selected matching model and the corresponding contextual affinity is computed. Finally, the linguistic and the contextual affinity values are combined to produce the final comprehensive semantic affinity value. Matching concepts are then selected by cutting off the concepts whose semantic affinity is below the threshold.

After describing the H-MATCH WordNet-based procedure for thesaurus construction and the basic functions for matching terms, datatypes, and relations, we describe how each model of H-MATCH works.

3.1 Thesaurus Construction

The H-MATCH thesaurus construction is performed in two steps. First the entries for both basic and compound terms are defined. Then, the terminological relationships holding among the term entries are defined. Given the set T of terms used as names of ontology elements, the construction procedure inserts into the thesaurus an entry for each basic term $bt_i \in T$, for each compound term $ct_i \in T$, and for each constituent token of ct_i. If a token is a compound term itself, the procedure is recursively iterated until all the compound terms are analysed and corresponding basic term entries are defined. Regarding terminological relationships, the first step is devoted to define terminological relationships for compound terms. Our approach relies on the idea that in a typical compound term ct, one of its constituent tokens denotes the central concept represented by ct, while the remaining tokens denote a specification of such a central concept [26]. In particular for English, we follow the heuristics that the last token bt_n appearing on the right side of a compound term ct composed by n tokens denotes the central concept, and that each remaining token bt_i, $i = 1 \ldots n-1$ we encounter going from the left side to the right side of ct denotes a qualification of the meaning of bt_n. On this basis, a NT relationship is defined between ct and bt_n and a RT relationship is defined between ct and each remaining token bt_i, $i = 1 \ldots n-1$. Finally, we define the terminological relationships SYN, BT, NT and RT between basic term entries on the basis of the relations among synsets that are provided by WordNet. In particular: a WordNet synonymy is represented through a SYN terminological relationship; a WordNet hypernymy (resp., hyponymy) relation is represented through a BT (resp., NT) terminological relationship; meronymy and coordinate terms relations of WordNet are represented through a RT relationship in thesaurus [6].

Example. We consider the example of Figure 1. The first step in the thesaurus construction is to extract from Apple-q and Apple-o the names of concepts and properties and to determine the thesaurus entries. Most names are single terms and are already present in WordNet. An entry for them is thus defined in the

[6] The examples and the thesaurus description provided in the paper are referred to nouns for the sake of clarity. In the case of verbs, the corresponding specific relations in WordNet, such as troponymy, are considered and mapped onto the thesaurus terminological relationships following analogous rules.

thesaurus. There are only two compound terms, Edible_Fruit and Mobile_Computer. The first one is retrieved in WordNet and therefore is considered as a basic term and inserted as an entry in the thesaurus. The second one is not retrieved in WordNet. For this reason, we split it into two tokens (i.e., Mobile and Computer); then we insert in the thesaurus two new entries, one for Mobile_Computer and one for Mobile, while an entry for Computer is already present in thesaurus being already a concept name.

The second step is to determine the terminological relationships among the thesaurus entries. First of all, we consider the compound term Mobile_Computer

Apple	SYN	**Apple**
Apple	NT	**Edible_Fruit**
Banana	SYN	**Banana**
Banana	NT	**Edible_Fruit**
Brand	SYN	**Brand**
Brazil	SYN	**Brazil**
Computer	SYN	**Computer**
Computer	BT	**Mobile_Computer**
Edible_Fruit	SYN	**Edible_Fruit**
Edible_Fruit	BT	**Apple**
Edible_Fruit	BT	**Banana**
Edible_Fruit	BT	**Grape**
Edible_Fruit	BT	**Pineapple**
Edible_Fruit	NT	**Fruit**
Food	SYN	**Food**
Fruit	SYN	**Fruit**
Fruit	BT	**Edible_Fruit**
Fruit	NT	**Product**
Grape	SYN	**Grape**
Grape	NT	**Edible_Fruit**

IBM	SYN	**IBM**
Italy	SYN	**Italy**
Location	SYN	**Location**
Location	NT	**Region**
Mobile	SYN	**Mobile**
Mobile	RT	**Mobile_Computer**
Mobile_Computer	SYN	**Mobile_Computer**
Mobile_Computer	NT	**Computer**
Mobile_Computer	RT	**Mobile**
Origin	SYN	**Origin**
Origin	BT	**Provenance**
Pineapple	SYN	**Pineapple**
Pineapple	NT	**Edible_Fruit**
Product	SYN	**Product**
Product	BT	**Fruit**
Provenance	SYN	**Provenance**
Provenance	NT	**Origin**
Region	SYN	**Region**
Region	BT	**Location**
USA	SYN	**USA**

Table 1. Example of thesaurus entries for the running example

and we insert a NT relationship between Mobile_Computer and Computer, in order to denote that mobile computers are a specialization of computers. Moreover, we insert also a RT relationship between Mobile_Computer and Mobile, according to the approach described above. Then, WordNet is exploited for deriving all the other relationships that are reported in Table 1. Finally, at the end of the thesaurus construction phase, a weight is associated with each terminological relationship in the thesaurus. The weights of terminological relationships used for thesaurus construction are 1.0 for SYN, 0.8 for BT/NT, and 0.5 for RT. Such weights have been maintained from ARTEMIS where they have been defined after extensive experimentation on several schema matching and integration cases. We performed experimentations using them also on several ontology matching

cases and we have seen that they work well also for ontology matching. Consequently, we maintain them as default weights also in the H-MATCH thesaurus construction procedure.

3.2 Basic Matching Functions

In this section, we describe the basic matching functions that are used in order to evaluate the similarity/compatibility of terms, datatypes, properties and semantic relations, respectively.

Term Affinity Function. The term affinity function $\mathcal{A}(t, t') \rightarrow [0, 1]$ evaluates the affinity between two terms t and t' based on the thesaurus. The term affinity function is borrowed from ARTEMIS and it is reported here for the sake of clarity. $\mathcal{A}(t, t')$ of two terms t and t' is equal to the value of the highest-strength path of terminological relationships between them in Th if at least one path exists, and is zero otherwise. A path strength is computed by multiplying the weights associated with each terminological relationship involved in the path, that is:

$$\mathcal{A}(t, t') = \begin{cases} \max_{i=1\ldots k} \{W_{t \rightarrow_i^n t'}\} & \text{if } k \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where: k is the number of paths between t and t' in Th; $t \rightarrow_i^n t'$ denotes the ith path of length $n \geq 1$; $W_{t \rightarrow_i^n t'} = W_{1_{tr}} \cdot W_{2_{tr}} \cdot \ \ldots \ \cdot W_{n_{tr}}$ is the weight associated with the ith path, and $W_{j_{tr}}, j = 1, 2, \ldots, n$ denotes the weight associated with the jth terminological relationship in the path.

Datatype Compatibility Function. The datatype compatibility function $\mathcal{T}(dt, dt') \rightarrow [0, 1]$ is defined to evaluate the compatibility of data types of two concept properties according to a pre-defined set CR of compatibility rules. $\mathcal{T}(dt, dt')$ of two data types dt and dt' returns 1 if dt and dt' are compatible according to CR, and 0 otherwise, that is:

$$\mathcal{T}(dt, dt') = \begin{cases} 1 & \text{iff } \exists \text{ a compatibility rule for } dt, dt' \text{ in } CR \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

For instance, with reference to XML Schema datatypes (which are relevant for OWL ontology matching), examples of compatibility rules that have been defined are: xsd:integer $\Leftrightarrow$ xsd:int, xsd:integer $\Leftrightarrow$ xsd:float, xsd:decimal $\Leftrightarrow$ xsd:float, xsd:short $\Leftrightarrow$ xsd:int.

Property and Semantic Relation Closeness Function. The closeness function $\mathcal{C}(e, e') \rightarrow [0, 1]$ is defined to calculate a measure of the distance between two elements e and e' of concept contexts. Depending on the way concept contexts are defined in each respective ontology, e and e' can be either two properties, or two semantic relations, or a semantic relation and a property, respectively.

$\mathcal{C}(e, e')$ exploits the weights associated with context elements and returns a value in the range [0,1] proportional to the absolute value of the complement of the difference between the weights associated with the elements, that is:

$$\mathcal{C}(e, e') = 1 - \mid W_e - W_{e'} \mid \tag{3}$$

where W_e and $W_{e'}$ are the weights associated with e and e', respectively. For any pairs of elements e and e', the highest value (i.e., 1.0) is obtained when weights of e and e' coincide. The higher the difference between W_e and $W_{e'}$ the lower the closeness value of e and e'.

3.3 Matching Models

The matching models have been conceived to span from surface to intensive matching. Each model calculates a semantic affinity value $SA_{c,c'}$ of two concepts c and c' which expresses their level of matching. $SA_{c,c'}$ is produced by considering linguistic and/or contextual features of concept descriptions. In a given matching model, the relevance of the linguistic and the contextual features of c and c' for matching can be established, by properly setting the linguistic affinity weight $W_{la} \in [0, 1]$ in the semantic affinity evaluation process.

Before describing matching models, we make a general consideration for the surface, shallow, deep, and intensive models. In the evaluation of the contextual affinity, a special case occurs when both the concepts have an empty context. To deal with this, three strategies are possible: i) *NULL-value strategy*: the empty contexts can be considered to have a semantics analogous to the one of the NULL value in relational databases. In this strategy, the contextual affinity is set to undetermined to capture this semantics; ii) *worst-case strategy*: since the concepts do not have elements in their contexts, the contextual affinity value is set to 0 for them, to express that no matching elements have been found in their contexts; iii) *best-case strategy*: since the concepts do not have elements in their contexts, the contextual affinity value is set to 1 for them, to express that two empty contexts are considered to fully match. In implementing H-Match, we have decided to adopt the worst-case strategy (ii) in order to avoid to produce semantic affinity values either too much optimistic (iii) or semantic affinity values that are based only on linguistic affinity (i), without accounting for the information about the presence of empty contexts.

Surface Matching. The surface matching is defined to take into account only the linguistic features of concept descriptions. Surface matching addresses the requirement of dealing with high-level, poorly structured ontological descriptions. Given two concepts c and c', surface matching provides a measure $SA_{c,c'}$ of their semantic affinity determined only on the basis of their names using the term affinity function (1), that is:

$$SA_{c,c'} \equiv \mathcal{A}(n_c, n_{c'}) \tag{4}$$

where n_c and $n_{c'}$ are the names of c and c', respectively. When the surface model is selected, the W_{la} weight is automatically considered to be 1 by H-Match.

Example. All the semantic affinity values computed using the surface matching on the running example ontologies Apple-q and Apple-o are shown in Table 3, referring to the thesaurus shown in Table 1. For instance, the semantic affinity

Apple-q/Apple-o	Apple	Banana	Brazil	Computer	Edible_Fruit	Grape	IBM
Apple	1.0	0.64	-	-	0.8	0.64	-
Food	-	-	-	-	-	-	-
Fruit	0.64	0.64	-	-	0.8	0.64	-
Italy	-	-	-	-	-	-	-

Apple-q/Apple-o	Italy	Mobile_Computer	Pineapple	Product	Region	USA
Apple	-	-	0.64	0.512	-	-
Food	-	-	-	-	-	-
Fruit	-	-	0.64	0.8	-	-
Italy	1.0	-	-	-	-	-

Table 2. Surface matching results for the running example

value of Apple and Grape is $0.8 \cdot 0.8 = 0.64$ as NT relationship is defined between Apple and Edible_Fruit and between Grape and Edible_Fruit, whose weight is 0.8. The other semantic values are calculated in an analogous way. According to this model, we note that the semantic affinity of the two Apple concepts is 1 as their names coincide.

Shallow Matching. The shallow matching is defined to take into account concept names and concept properties. With this model, we want a more accurate level of matching, by taking into account not only the linguistic features but also information about the presence of properties and about their cardinality constraints. For property comparison, each property $p_i \in P(c)$ is matched against all properties $p_j \in P(c')$ using (1) and (3), and the best matching value $m(p_i)$ is considered for the evaluation of $SA_{c,c'}$, as follows:

$$m(p_i) = \max\{\mathcal{A}(n_{p_i}, n_{p_j}) \cdot \mathcal{C}(p_i, p_j)\}, \forall p_j \in P(c') \tag{5}$$

where n_{p_i} and n_{p_j} denote the names of p_i and p_j, respectively. $SA_{c,c'}$ is evaluated by the shallow matching as the weighted sum of the linguistic affinity of c and c', calculated using (1), and of their contextual affinity, calculated as the average of the property best matching values computed using (5), that is:

$$SA_{c,c'} = W_{la} \cdot \mathcal{A}(n_c, n_{c'}) + (1 - W_{la}) \cdot \frac{\sum_{i=1}^{|P(c)|} m(p_i)}{\mid P(c) \mid} \tag{6}$$

Example. The matching models from shallow to intensive have been applied to the running example using the following weights values for contextual features:

strong properties and weak properties have been associated with the weights $W_{\mathsf{sp}} = 1.0$ and $W_{\mathsf{wp}} = 0.5$, respectively. For semantic relations, $W_{\mathsf{same-as}} = 1.0$, $W_{\mathsf{kind-of}} = 0.8$, $W_{\mathsf{part-of}} = 0.5$, and $W_{\mathsf{associates}} = 0.3$. These are the default weights in H-MATCH. For their definition, we followed a method similar to the one used for weighting terminological relationships: we defined specific values for each weight and then we tested them on several real cases, by choosing as default values those that exhibited best behavior in most cases. Furthermore, for the shallow matching and the other two remaining models, we have configured H-MATCH with a W_{la} value of 0.5, which is used as a default in H-MATCH as it guarantees an equilibrated balancing of linguistic and contextual affinity in the matching process.

All the results produced for the running example using the shallow matching model are shown in Table 3. As an example of semantic affinity evaluation using the shallow matching model, we continue to consider the case of matching concept Apple of Apple-q with the concept Grape of Apple-o. Both the concepts have a weak property in their contexts, i.e., origin for Apple and provenance for Grape. By applying the term affinity function (1) and the closeness function (3), the matching value of these two properties is evaluated as follows $\mathcal{A}(\mathsf{origin},\mathsf{provenance}) \cdot \mathcal{C}(\mathsf{weak\_property},\mathsf{weak\_property}) = 0.8 \cdot 1.0 = 0.8$. Since these properties are the only elements in the contexts of both Grape and Apple, this is also the best matching value. According to this, the semantic affinity between Apple and Grape is computed as $(0.5 \cdot 0.64) + (0.5 \cdot 0.8) = 0.72$. As we can see,

Apple-q/Apple-o	Apple	Banana	Brazil	Computer	Edible_Fruit	Grape	IBM
Apple	0.5	0.72	-	-	0.4	0.72	-
Food	-	-	-	-	-	-	-
Fruit	0.32	0.32	-	-	0.4	0.32	-
Italy	-	-	-	-	-	-	-

Apple-q/Apple-o	Italy	Mobile_Computer	Pineapple	Product	Region	USA
Apple	-	-	0.72	0.256	-	-
Food	-	-	-	-	-	-
Fruit	-	-	0.32	0.4	-	-
Italy	0.5	-	-	-	-	-

Table 3. Shallow matching results for the running example

with this model the semantic affinity value of Apple and Grape is increased with respect to the previous value obtained using the surface model. This because we have taken into account also the presence of matching properties in their context. We also note that the two apple concepts are now less matching than before, because we are able to capture differences due to the context properties.

Deep Matching. The deep matching model is defined to take into account concept names and the whole context of concepts, that is, both properties and semantic relations. Each element $e_i \in Ctx(c)$ (i.e., a property or an adjacent) is compared against all elements $e_j \in Ctx(c')$ using (1) and (3) and the best matching value $m(e_i)$ is considered for the evaluation of $SA_{c,c'}$, as follows:

$$m(e_i) = \max\{\mathcal{A}(n_{e_i}, n_{e_j}) \cdot \mathcal{C}(e_i, e_j)\}, \forall e_j \in Ctx(c') \tag{7}$$

where n_{e_i} and n_{e_j} denote the names of e_i and of e_j, respectively. With the deep matching model, $SA_{c,c'}$ is evaluated as the weighted sum of the linguistic affinity of c and c', calculated using (1), and of their contextual affinity, calculated as the average matching value for the elements of the context of c using (7), that is:

$$SA_{c,c'} = W_{la} \cdot \mathcal{A}(n_c, n_{c'}) + (1 - W_{la}) \cdot \frac{\sum_{i=1}^{|Ctx(c)|} m(e_i)}{\mid Ctx(c) \mid} \tag{8}$$

Example. All the results produced by the deep matching on the running example ontologies are shown in Table 4. Considering the contexts of Apple in Apple-q and of Grape in Apple-o, H-MATCH searches in the context of Grape for the best matching element for each element of the context of Apple. It is simple to verify that the best matching element for Fruit is Edible_Fruit, and that the best matching element for origin is provenance, while Food has not any matching element in the context of Grape. By applying the term affinity function (1) and the closeness function (3), we obtain that the best matching value is 0.8 both for Fruit and for origin. Then, considering that the context of Apple is composed by 3 elements, the contextual affinity is given by $\frac{0.8+0.8}{3} = 0.53$. The linguistic affinity value between Apple and Grape is 0.64, so that the final semantic affinity is calculated as $(0.5 \cdot 0.64) + (0.5 \cdot 0.53) = 0.59$. The main advantage of the deep model in the example is that it emphasizes the difference between Banana, Pineapple, Grape, and Apple in Apple-o that is due to the fact that the former three are fruits in the ontology, while the last one is a computer brand. This is evident by taking into account the results obtained for Fruit in Apple-q which has a high semantic affinity value with Banana, Pineapple, and Grape, and a low affinity value with Apple in Apple-o.

Intensive Matching. The intensive matching model is defined to take into account concept names, the whole context of concepts, and also property values, in order to exhibit the highest accuracy in semantic affinity evaluation. In fact, by adopting the intensive model not only the presence and cardinality of properties, but also their values are considered to produce the resulting semantic affinity value. Given two concepts c and c', the intensive matching calculates a comprehensive matching value for the elements of the context of c such as in (7) as well as a matching value $v(p_i)$ for each property $p_i \in P(c)$. The matching value $v(p_i)$ is calculated as the highest value obtained by composing the affinity of the name n_{p_i} and the value v_{p_i} of p_i with the name n_{p_j} and the value v_{p_j} of each property

Apple-q/Apple-o	Apple	Banana	Brazil	Computer	Edible_Fruit	Grape	IBM
Apple	0.5	0.59	-	-	0.53	0.59	-
Food	-	0.4	-	-	0.32	0.4	-
Fruit	0.32	0.72	-	-	0.72	0.72	-
Italy	-	-	-	-	-	-	-

Apple-q/Apple-o	Italy	Mobile_Computer	Pineapple	Product	Region	USA
Apple	-	-	0.59	0.386	-	-
Food	-	-	0.4	0.4	-	-
Fruit	-	-	0.72	0.8	-	-
Italy	0.5	-	-	-	-	-

Table 4. Deep matching results for the running example

$p_j \in P(c')$, respectively. For property values comparison, we exploit the term affinity function (1) if the property value is the name of a referenced concept, and the datatype compatibility function (2) if the property value is a datatype, that is:

$$v(p_i) = \begin{cases} \max\{\mathcal{A}(n_{p_i}, n_{p_j}) \cdot \mathcal{A}(v_{p_i}, v_{p_j})\}, \forall p_j \in P(c') \text{ iff } v_{p_i} \text{ is a reference name} \\ \max\{\mathcal{A}(n_{p_i}, n_{p_j}) \cdot \mathcal{T}(v_{p_i}, v_{p_j})\}, \forall p_j \in P(c') \text{ iff } v_{p_i} \text{ is a datatype} \end{cases} \tag{9}$$

$SA_{c,c'}$ is evaluated by the intensive matching as the weighted sum of the linguistic affinity of c and c', calculated using (1), and of their contextual affinity, calculated as the average of the matching values for the elements of the context of c using (7) and for the property values calculated using (9), that is:

$$SA_{c,c'} = W_{la} \cdot \mathcal{A}(n_c, n_{c'}) + (1 - W_{la}) \cdot \frac{\sum_{i=1}^{|Ctx(c)|} m(e_i) + \sum_{j=1}^{|P(c)|} v(p_j)}{|Ctx(c)| + |P(c)|} \tag{10}$$

Example. All the results produced by the intensive matching for concepts of the running example ontologies are shown in Table 5. Let us consider again Apple in Apple-q and Grape in Apple-o of the running example. Using the intensive matching, the contextual affinity of these two concepts must consider also the value of the properties origin and provenance, i.e., the concept Italy. As shown above, the affinity between these two property values is given by the following formula: $\mathcal{A}(\text{origin},\text{provenance}) \cdot \mathcal{A}(\text{Italy},\text{Italy}) = 0.8 \cdot 1.0 = 0.8$. The context of Apple is composed by 3 elements, but, in the intensive model, we have to sum to this number also the number of properties, that is 1. For this reason, the contextual affinity of Apple and Grape is given by $\frac{0.8+0.8+0.8}{4} = 0.6$. The linguistic affinity value between Apple and Grape is 0.64, so that the final semantic affinity is calculated as $(0.5 \cdot 0.64) + (0.5 \cdot 0.6) = 0.62$. The main advantage of the intensive model with respect to the deep model, is that we now capture the difference between Banana, Pineapple and Grape. In fact, all these concepts have in common with Apple of Apple-q the fact that they are fruits, but Grape has a higher se-

mantic affinity with Apple because they both origin from Italy, while Banana and Pineapple origin from Brazil.

Apple-q/Apple-o	Apple	Banana	Brazil	Computer	Edible_Fruit	Grape	IBM
Apple	0.5	0.52	-	-	0.5	0.62	-
Food	-	0.4	-	-	0.32	0.4	-
Fruit	0.32	0.72	-	-	0.72	0.72	-
Italy	-	-	-	-	-	-	-

Apple-q/Apple-o	Italy	Mobile_Computer	Pineapple	Product	Region	USA
Apple	-	-	0.52	0.356	-	-
Food	-	-	0.4	0.4	-	-
Fruit	-	-	0.72	0.8	-	-
Italy	0.5	-	-	-	-	-

Table 5. Intensive matching results for the running example

3.4 Matching Policies

The set of parameters to configure the current execution of H-MATCH for a given matching case is called *matching policy*. A matching policy is a 4-tuple of the form $\langle model, W_{la}, t, mapping \rangle$, where:

- $model \in$ {surface, shallow, deep, intensive} denotes the matching model to be used for H-MATCH execution;
- $W_{la} \in [0, 1]$ denotes the linguistic affinity weight to be used for setting the relevance of the linguistic affinity, and, consequently, the one of the contextual affinity;
- $t \in (0, 1]$ denotes the matching threshold value to be used in order to cut-off from the results the matching concepts having a low value of semantic affinity, and thus considered poorly relevant;
- $mapping \in$ {one-to-one, one-to-many} denotes the kind of mapping to be determined at the end of the matching process.

In the context of open networked systems, each node can specify its own policy directly within the knowledge discovery request, in order to force the configuration of H-MATCH at the destination node as described in Section 4.

3.5 Considerations on the Running Example

The main challenging issue in the running example that we have presented above is to capture the difference between the meaning of the concept Apple in Apple-q and the meaning of Apple in Apple-o. The two concepts have the same name, but

in Apple-q apple is a kind of fruit, while in Apple-o it is a brand of computer. For this reason when we compare Apple-q and Apple-o, the fruit Apple is expected to be more similar to Banana, Pineapple, and Grape than to Apple in Apple-o. Moreover, we expect to have a higher similarity between Apple and Grape, since they both origin from Italy, while Banana and Pineapple origin from Brazil. In order to achieve these goals, the matching based on the linguistic features alone would fail. In fact with the surface matching, we obtain that the best matching for Apple in Apple-q is Apple in Apple-o, due to the synonymy between their names. However, the surface matching enriches the information provided in the thesaurus, because it captures a semantic affinity between Apple and the other fruits even if they do not have any terminological relationship between their names in the thesaurus. A first refinement of the results is given by the shallow matching model. In this model, we are able to disambiguate the meaning of the apple concepts, by detecting also a high affinity between Apple and the other fruits. With the shallow model, we do not capture the affinity between the concept Fruit and the different types of fruits in Apple-o because we do not have any property in the context of Fruit. With the deep model, we consider also semantic relations. This gives us the possibility to strengthen the affinity between Fruit and Banana, Pineapple, and Grape, by exploiting the semantic relation that holds between Fruit and Apple in Apple-q. The best matching concepts for Apple are still Banana, Pineapple, and Grape, although we do not capture the higher affinity between Apple and Grape. This goal is achieved by means of the intensive matching model, because it captures the fact that these concepts have two similar properties (i.e., origin and provenance) and that these properties have the same value (i.e., Italy). The example shows how the matching models of H-Match can be used to adapt the algorithm to the specific features of the ontologies to be matched. A further discussion on the applicability of the different matching models is given in Section 4.

4 Application of H-Match to Knowledge Discovery in Open Networked Systems

In this section, we present the query-based approach to knowledge discovery and sharing we developed in the framework of the Helios open system which relies on H-Match for ontology matching. Subsequently, we discuss the design principles that we followed for the implementation of H-Match in Helios.

4.1 Query-Based Knowledge Discovery

In Helios, independent peers with equal role and capabilities cooperate by sharing their information resources (e.g., data, documents) described through peer ontologies. Each node provides its own ontology describing the information resources to be shared and interacts with the other members of the system by sending *probe queries*. A probe query provides an ontological description of target concept(s) of interest for the peer. The Helios probe query template is reported in Figure 3 and it is composed of the following clauses:

- *Find:* list of target concept(s) names.
- *With:* (optional) list of properties of the target concept(s).
- *Where:* (optional) list of conditions to be verified by the property values, and/or (optional) list of concepts related to the target by a semantic relation.
- *Matching policy:* (optional) specification of the H-MATCH configuration requested for the evaluation of the query.

Probe query template

Find	target concept name [, ...]
[*With*	⟨property name⟩ [, ...]]
[*Where*	condition,
	⟨related concept, semantic relation name⟩ [, ...]]
[*Matching policy*	⟨ model, W_{la}, t, mapping ⟩]

Fig. 3. The reference probe query template

The answer to a probe query is list of concepts that match the target. As described in Figure 4, the structure of the HELIOS answer template contains the following clauses:

- *Concept:* name of the matching concept.
- *Properties:* (optional) list of properties of the matching concept.
- *Adjacents:* (optional) list of concepts related to the matching concept by a semantic relation.
- *Matching:* set of pairs ⟨target concept, affinity value⟩, specifying the target concept with which the matching concept matches, together with the corresponding affinity value.
- *Matching policy:* (optional) the matching policy adopted for the evaluation of the query.

Probe answer template

{*Concept*	matching concept name
[*Properties*	⟨property name⟩ [, ...]]
[*Adjacents*	⟨related concept, semantic relation name⟩ [, ...]]
Matching	⟨target concept, affinity value⟩[, ...]
[*Matching policy*	⟨ model, W_{la}, t, mapping ⟩]}

Fig. 4. The reference probe answer template

If a peer is interested in discovering nodes capable of providing information resources semantically related to a given target, it composes and submits to the

system a probe query according to the query template of Figure 3. Receiving a probe query, a peer invokes the H-MATCH algorithm to compare such a request against the concepts contained in its peer ontology in order to identify whether there are concepts matching the target. In particular, the *Find*, *With*, and *Where* clauses are used to derive a H-MODEL description of the target concept(s), while the matching policy to apply is derived from the *Matching policy* clause of the probe query, if specified [7]. As a result, for each target concept of the probe query, H-MATCH returns a (possibly empty) ranked list of matching concepts semantically related to the target (that is, those concepts whose semantic affinity value exceeds the threshold specified in the adopted matching policy). Depending on the kind of mapping specified in the policy, this ranked list can contain either one single concept (one-to-one mapping policy) which is the best-matching concept for the target, or a set of concepts (one-to-many policy), which are all best-k matching concepts for the target. Finally, the results of H-MATCH are organized according to the probe answer template of Figure 4, and such an answer is replied back to the requesting peer. Collecting query replies from answering peers, the requesting peer evaluates the results and decides whether to further interact with those peers found to be relevant in order to access the specific information resources. A discussion on how the access to information resource data takes place once the knowledge discovery process has been completed is out of the scope of this paper. For further details, the reader can refer to [20].

4.2 H-MATCH Implementation Design

The H-MATCH algorithm and related techniques have been implemented in C++ in the framework of the HELIOS project. In order to acquire the OWL ontology descriptions in H-MODEL, we exploit the OWL APIs commonly used to this end also by other ontology matching tools [10]. In this section, we discuss the design principles that we followed for implementing H-MATCH.

Creation and management of the thesaurus. This functionality exploits the WordNet C library, distributed by the WordNet group at the Princeton University [8]. The library provides the basic functionalities to access the lexical database and to find the relations holding among terms. Our implementation extracts from the ontologies to be matched the names of the elements, defines the thesaurus entries, and exploits WordNet for the definition of the terminological relationships among them, as described in Section 3. The thesaurus is represented as a graph where the nodes are the entries derived from entity names and the edges are the weighted terminological relationships. A specific problem that is addressed by the thesaurus management functionality is related to terms that are not included in WordNet (e.g., some acronyms). We have taken into account three main options:

[7] If the matching policy is not specified in the query, the receiving peer can autonomously select the policy to apply according to internal criteria (e.g., workload, bandwidth).

[8] http://wordnet.princeton.edu/

i) off-line manual extension of the WordNet-based thesaurus with the main terms of the peer ontology domain not included in WordNet; ii) semi-automated extension of the WordNet-based thesaurus with the main terms of the peer ontology domain not included in WordNet by referring to a domain-specific vocabulary, if available; iii) use of syntax-based techniques (e.g., string similarity) to recognize and manage these terms. Currently, we enforce the first option, by providing some basic editing functionalities for off-line thesaurus extension.

Policy management. This functionality has the aim of managing the matching policies for the H-MATCH configuration. The matching policies are managed through the configure method of the class HMatch.

Representation and storage of ontology concepts. This functionality is based on the idea to represent an ontology as a set of C++ objects, called ConceptVectors. Each concept vector represents a concept in the ontology, together with its context in terms of properties and semantic relations. In our implementation, each H-MATCH execution is characterized by two set of concept vectors, called target and ontology, respectively. The first set contains the concepts that are the target of the matching process, while the second set contains the concepts that are the basis of the matching process.

Affinity functions. This functionality provides the functions that are used for evaluating both the linguistic and the contextual affinity between two concepts, working on concept vectors. In particular, for each target concept, the matching process calculates the semantic affinity value with the concepts that compose the ontology.

A main portion of the UML class diagram of the C++ classes implementing the H-MATCH functionalities is shown in Figure 5. The HMatch class is used for representing the process of matching two ontologies by means of the H-MATCH algorithm. The class is configured by means of a configure function that sets the value associated with the weight of the linguistic affinity (WLA) and the threshold from a MatchingPolicy object. Each HMatch instance is then associated with an instance of the LinguisticAffinity object, which is used to represent the $\mathcal{A}(n, n')$ function described in (1). This association gives HMatch the capability to evaluate the linguistic affinity holding between two concepts. The LA method returns the linguistic affinity between two ConceptVector as a float number in the range [0,1]. In a similar way, the CA function calculates the contextual affinity between two concepts and requires a third parameter that specifies the matching model to be used. Finally, the comprehensive semantic affinity value is provided by means of the SA method that invokes LA and CA, respectively. The evalResults method is exploited for calculating the semantic affinity value for each target conceptand each concept of the ontology. The complete set of results are then stored, together with WLA, threshold, and model used, in the Results object. Each result is a triple of the form ⟨query concept, ontology concept, semantic affinity

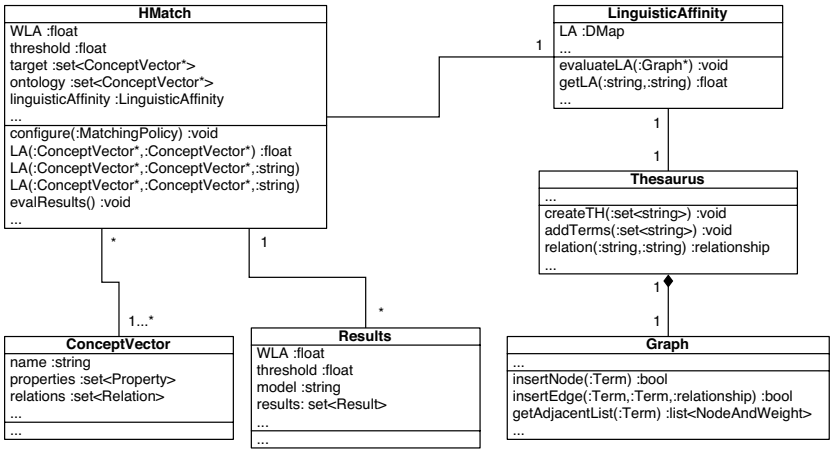


Fig. 5. A significant portion of the H-MATCH UML class diagram

value⟩. The thesaurus of terminological relationships is represented by the Thesaurus object, which is composed by a Graph. The graph is implemented by means of adjacency lists in the class Graph. The class Thesaurus supports basically two main tasks: i) the creation of the thesaurus given a set of strings, and ii) the mapping between the relations found in WordNet and the terminological relationships supported by H-MATCH, as described in Section 3. A further functionality of the Thesaurus is given by the addTerms method that is used for updating a predefined thesaurus with new term(s) and terminological relationship(s) entries. This function is adopted in the open networked scenario for extending the thesaurus that has been pre-calculated for a peer ontology. In particular, we have a WordNet pre-processing based on the contents of a given peer ontology which is performed off-line. At the time the matching is requested, the extension of the thesaurus is performed with new terms/terminological relationships of the target ontology. Usually the target ontology (i.e., a probe query) contains a limited number of concepts and the extension process can be performed on-line easily. Finally, the thesaurus is associated with a LinguisticAffinity object that exploits a Dijkstra-based algorithm for evaluating the $\mathcal{A}(n, n')$ function described in (1) over the thesaurus graph. These results are then stored in a map that associates a float value in the range [0,1] with each pair of terms in the thesaurus. The map is then refreshed to reflect the thesaurus extensions.

4.3 H-MATCH Optimization

Let n_c be the number of target concepts in the probe query and m_c be the number of concepts in the peer ontology. We developed some optimizations for H-MATCH to avoid to perform $n_c \times m_c$ matchings. This problem has been addressed by other approaches proposed in literature. For example, the QOM tool [27] pro-

poses several strategies for reducing the number of matching performed by the matchmaker starting from an initial set of candidate mappings. Similarly, we want to reduce the number of atomic matchings required for a probe query by contemporary guaranteeing a high level of accuracy of the results. The idea is to match a target concept $\overline{c}$ of the probe query only against the concepts of the ontology that have a high probability to have a high semantic affinity with $\overline{c}$. To this end, we observed that the problem is analogous to the problem of finding resources over the Web using the PageRank algorithm [28]. In PageRank, the *importance* of a page p is based both on the number of other pages that point to p and on the importance of those other pages [29]. A ranking scheme is then exploited for matching a query against the pages that have a high importance in the rank. For H-Match optimization, we have defined an algorithm, called ConceptRank, that is based on the idea of building a ranking scheme of concepts in a peer ontology and of exploiting it for matching a target concept *only against* the concepts that have a high level of importance in the rank. The rank of a concept is determined by taking into account the number and the weight of ontology relations that point to it. In this step, the difference with respect to PageRank is that the relevance of an edge is given by the weight associated with it. The importance of a concept c is a measure proportional to the number of concepts c_i that have a semantic relation with c and to the importance of c_i. Probe queries are processed by matching each target concept against the peer ontology concepts in the ranking scheme starting from the most important ones and by stopping the matching process when a given number TF of atomic matchings produce a semantic affinity value under the matching threshold. The number TF is called *tolerance factor* and is determined experimentally in order to obtain the best balance between the required reduction of atomic matchings and the quality of the results obtained for a probe query. This process has to take into account the fact that, in the ranking scheme, the most important concepts are the ones that are mostly referenced by the other concepts. For this reason, the most important concepts in the rank have generally the most generic meaning in the ontology. When a probe query is searching for a concept with a specific meaning, the top ranked concepts (i.e., the ones with the most generic meaning) could not be the best answers for the query. In order to address this problem, we have defined two main strategies:

- *Ranking-based strategy.* A target concept $\overline{c}$ is matched against the concepts c_i in the rank starting from the top ranked concept c_0. The matching process ends when the number of atomic matchings that do not provide a semantic affinity value exceeding the threshold is higher than or equal to the tolerance factor TF. A variation of this strategy, called *Surface Ranking-based Strategy*, avoids to match $\overline{c}$ against the peer ontology concepts with the most generic meaning through a revision of the ranking scheme obtained by exploiting the surface matching model.
- *Graph-based Strategy.* In this strategy, we use the ranking scheme just to find the most important concept c_0 that is seen as the concept that has the highest probability to be relevant for the probe query. Then, the concepts that have

to be matched against $\overline{c}$ are chosen by visiting the ontology graph starting from c_0. The process is iterated until the number of atomic matchings that do not provide a semantic affinity value exceeding the threshold is higher than or equal to the tolerance factor TF. Also in this case, a variation, called *Surface Graph-based Strategy*, has been developed for the same purpose of the previous one.

We have compared these strategies in order to determine their impact on the matching process. The results of this experimentation show how the first strategy guarantees a higher level of performance and a lower level of accuracy than the last one. Moreover, the graph-based strategy is more adequate to obtain the most relevant results. The optimization strategies are available in the HELIOS query processing module to pre-configure the peer capabilities at the initialization time.

5 Experimental Results

We analyze the behavior of H-MATCH by performing different tests devoted i) to evaluate the matching models with respect to performance and quality of results in terms of precision and recall, and ii) to compare the H-MATCH results with the results produced by selected ontology matching tools available on the Web.

5.1 Experimental Evaluation of H-MATCH Models

In order to evaluate the effectiveness of the four H-MATCH models, two kind of tests have been executed: i) a *quality test*, where the experiments were devoted to examine precision and recall of the models on real case studies; and ii) a *performance test*, where the H-MATCH models have been analyzed with respect to computation time and scalability in manifold scenarios with different level of complexity.

Quality Test. For what concern the H-MATCH quality test, we have considered two OWL ontologies from the publication domain (i.e., Ka [9], Portal [10]), and we asked to users (i.e., students of our Ontologies and Semantic Web course) to manually define a set of target ontologies (i.e., twenty target ontologies) related to the publication domain, too. Each target ontology describes publication-related concepts with some properties, semantic relations, and property values according to the domain knowledge and experience of the ontology creator. To avoid to be influenced, target ontologies have been composed without considering Ka and Portal ontology contents. Note that the choice of manually constructing small-size target ontologies for extensive experimentation is motivated by the fact that H-MATCH is used for knowledge discovery in the HELIOS open networked system. In such a context, the typical problem is to match a probe query embedding a

[9] http://protege.stanford.edu/plugins/owl/owl-library/ka.owl
[10] http://www.aktors.org/ontology/portal

small-size ontology describing target concept(s) against a large ontology. After the target ontology definition, we asked users to consider Ka and Portal ontologies, and to map each concept of the target ontologies they have defined on one or more concepts belonging to the two reference ontologies according to their intuitive understanding of concept similarity. In this way, we have collected a set of 1:1 mappings between the target ontologies and the reference ontology concepts. The goal of the quality test is to evaluate the effectiveness of H-MATCH matching process by verifying the overlapping between the results produced by the different H-MATCH models and the manual mappings. Moreover, we intend to analyze the impact of different threshold values t and linguistic affinity weights W_{la} on the level of overlapping. To this end, we use *precision* and *recall*, which are the measures commonly adopted for matching evaluation [10] derived from the classical definitions of Information Retrieval [30]. In particular, *precision* is defined as the ratio of the number of relevant matching concepts automatically found by H-MATCH to the total number of matching concepts automatically found. *Recall* is defined as the ratio of the number of relevant matching concepts automatically found by H-MATCH to the total number of matching concepts (i.e., mappings) manually defined.

In Figure 6, we show the precision of the H-MATCH matching models with a linguistic affinity weight which varies from $W_{la} = 0.1$ to $W_{la} = 0.8$ and the threshold $t = 0.6$. These values show the number of manual mappings which

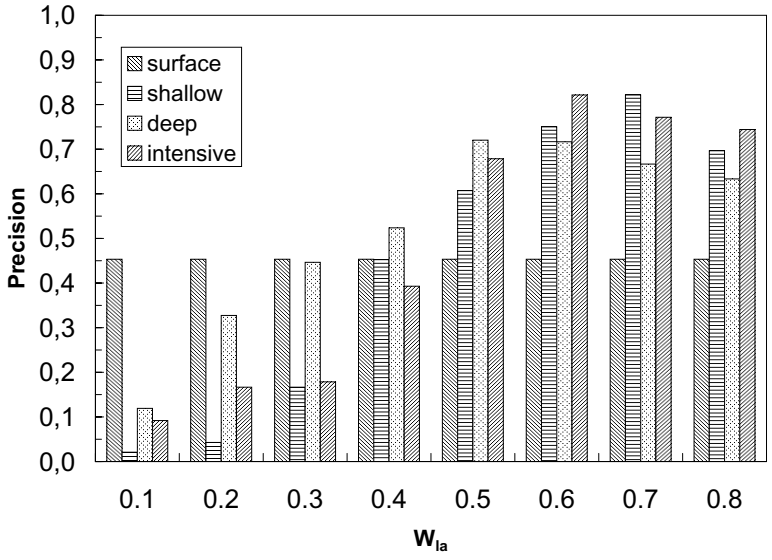


Fig. 6. The precision of the H-MATCH matching models with $t = 0.6$

have been correctly identified by H-MATCH with respect to the total number of results provided by the algorithm. We observe that the surface matching is not affected by the variation of the W_{la} parameter and can ensure a precision equal to 45%. This is due to the fact that the surface matching always works with the parameter $W_{la} = 1$ and contextual features are not considered in the matching evaluation. For what concern the other matching models, H-MATCH can guarantee a high level of precision in correspondence of high W_{la} values. In particular, we observe that with $W_{la} \geq 0.4$, H-MATCH can achieve a precision of 82%.

In Figure 7, we measure the H-MATCH recall in correspondence of different linguistic affinity weights and with the threshold $t = 0.3$. The results show the

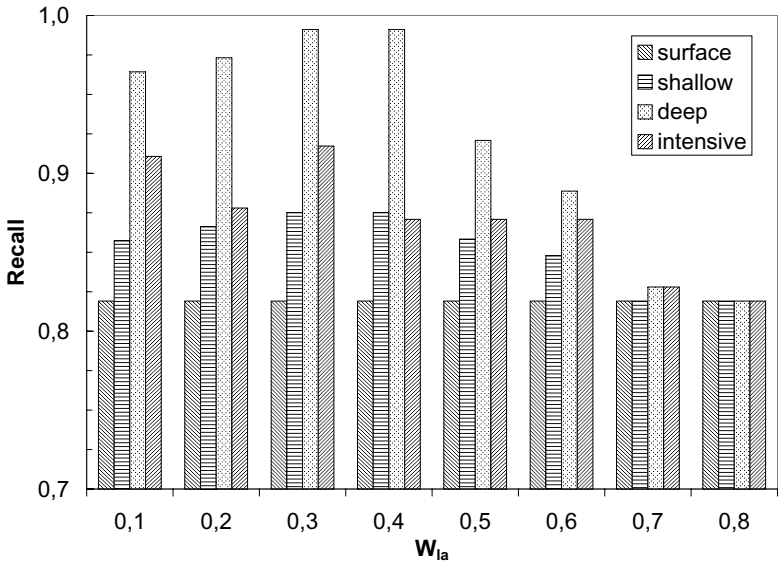


Fig. 7. The recall of the H-MATCH matching models with $t = 0.3$

number of manual mappings which have been correctly identified with respect to the total number of mappings manually defined. Independently from the values of W_{la}, we observe that the recall of each matching model is over the 80%, and the deep matching model can achieve a recall equal to 99.1% when $W_{la} = 0.4$.

From these figures, we observe that the choice of the correct value for the threshold t and for the linguistic affinity weight W_{la} depends on the goal of the matching case. If we are interested in very precise results, we have to set high values (i.e., ≥ 0.6) both for threshold and for linguistic affinity weight. On the opposite,

if we are interested in accuracy of results, lower values (i.e., $0.3 \leq t, W_{la} \leq 0.5$) for both parameters work better. In both cases, deep and intensive matching are to be preferable and can provide better results than surface and shallow models. We want to stress that H-MATCH has been implemented to be interactively configured, in order to suit the matching process to the requirements of the specific matching case.

Performance Test. The semantic affinity evaluation performed by H-MATCH is based on the comparison of all the concepts and corresponding contexts contained in the two ontologies to be considered. For this reason, the computation time is affected by the number of elements to be compared and by the adopted matching model. The goal of the performance test is to focus the attention on the performance differences between the four matching models for the aspects related to the contextual affinity evaluation. To this end, we analyze the matching models when comparing a large reference ontology with a great number of small target ontologies which differ in complexity of concept contexts. In this test, the linguistic affinity is computed only once as it is common to all matching models and, as such, it is not considered in discriminating between the different matching models [11]. The test is characterized by the following features:

- The reference ontology is the W3C Wine ontology [12]. We have selected this well known ontology because it provides a relevant number of concepts (more than one hundred concepts) with an adequate richness in terms of properties, semantic relations, and property values per concept (an average of five properties, two semantic relations, and six property values per concept, respectively).
- The target ontologies used in the comparison with the reference Wine ontology are randomly composed and vary according to four different complexity measures: number of concepts in the ontology and number of properties, semantic relations, and property values per concept, respectively.
- The test has been executed on a Dual Xeon machine at 2.80 GHZ with 1GB RAM and SCSI disks.

The diagrams reported in Figure 8 show the computation time of the H-MATCH models by varying one complexity measure while keeping fixed the other three measures. In Figure 8(a), we have measured the computation time required for comparing the reference ontology with a target ontology which varies in the number of concepts and with a fixed number of properties, semantic relations, and property values per concept. In the remaining diagrams, we have fixed the number of target ontology concepts and the complexity measure which varies

[11] However, based on the test cases performed on real examples, such as those presented in Section 5.2, where the linguistic affinity is calculated each time, we found that the average time required for matching (including LA computation) is in the range of 0.6 seconds for quite complex ontologies (ontologies with more than sixty concepts to one hundred and more concepts)

[12] http://www.w3.org/TR/2003/WD-owl-guide-20030210/wine.owl

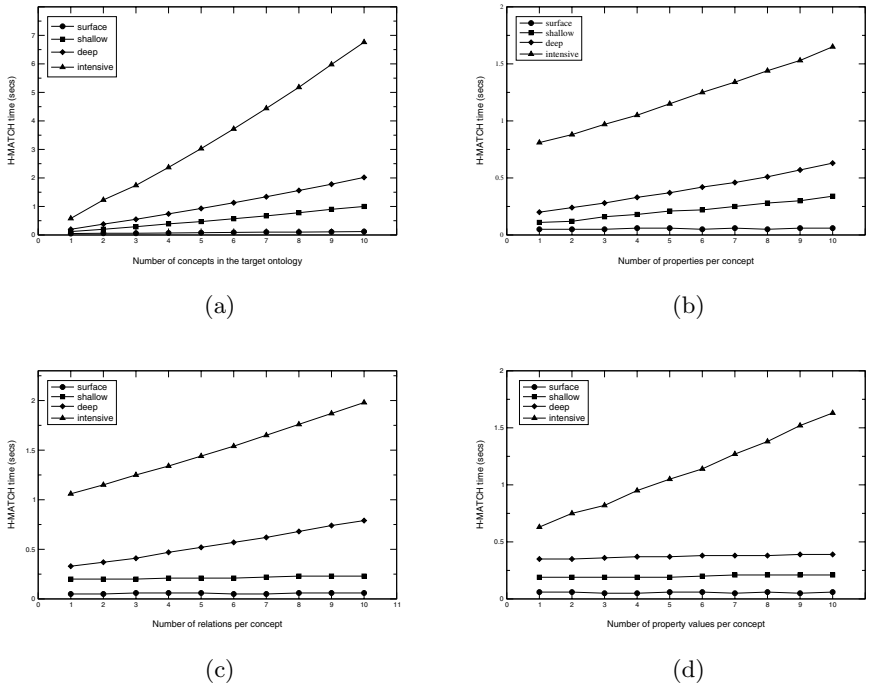


Fig. 8. Comparison of the H-MATCH models

is the number of properties (Figure 8(b)), semantic relations (Figure 8(c)), and property values (Figure 8(d)), respectively.

We stress that, in the worst case, all the models, including the intensive matching, observe a linear growth of their computation times in correspondence of the increase of the elements to be evaluated. Furthermore, the surface, shallow, and deep matching are also scalable, in that the semantic affinity evaluation restricted to concept names, properties, and semantic adjacents is a very efficient task. On the contrary, the intensive matching computation times are higher and not comparable with the corresponding values of the other matching models. This means that the semantic affinity evaluation of property values has a great impact on the matching performances. Anyway, we observe that there is a real difference between the computation times of Figure 8(a) and the remaining diagrams: the real impact on matching performances is due to the number of concepts to be processed in the target ontology rather than to the variations in concept context definition.

Considerations. Performance and quality tests show that the choice of the most suitable matching model is a key factor for obtaining relevant matching results. This depends on the level of detail of the ontology descriptions to be

compared as well as on the expected degree of precision and recall of the results. Furthermore, the appropriate configuration of the threshold and of the linguistic affinity weight have an impact on the quality of H-MATCH results. When adopting H-MATCH, the trade-off between high performances, high precision, and high recall has to be evaluated. In scenarios where a rapid response time is required (e.g., open networked systems with discovery queries), some lacks in matching precision and recall can be admitted in turn of high performances during the semantic affinity evaluation. On the opposite, when computation time is not a critical constraint, we can apply the matching model which best suits the particular application scenario. In Table 6, we summarize the main features of the matching models and their corresponding suggested scenarios. The surface

	Surface	**Shallow/Deep**	**Intensive**
Ontological description	Poorly structured ontologies with very simple resource description	Schematic ontologies with taxonomic resource description	Articulated ontologies with rich resource description
Kind of matching	Linguistic-driven matching	Linguistic and context-driven matching	Linguistic, value, and context-driven matching
Advantages	High performances	More accurate characterization of matching concepts	High precision and recall

Table 6. Applicability of the matching models

model is useful when only concept names are to be considered. It requires few computational resources since neither concept properties nor semantic relations are considered. This model is well suited, for example, to perform an initial ontology comparison to decide whether it is worth to perform a deeper analysis. If the ontology is constituted mainly by concepts with a few number of properties and hierarchical relation among concepts, the shallow and deep model allow a good degree of precision without requiring great amount of computational resources. In presence of an articulated ontology, with rich resource descriptions and where relations among concepts are described through property values, the intensive model guarantees the most precise and accurate results, although being the most expensive in term of computation.

5.2 Comparative Evaluation with Other Tools

In this test, we compare the H-MATCH algorithm with similar ontology matching tools and we analyze the results produced by applying them to the same test case. The tools considered are *FOAM* [13] (*Framework for Ontology Alignment*

[13] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/

and Mapping) and *OLA* [14] (*OWL Lite Alignment*). The choice of these tools has been led by the following considerations:

- The ontology mapping approach proposed by these tools is similar to H-MATCH [15].
- A prototype of these tools is free for download or is available through a Web Service.
- Different test cases and associated experimental results in term of precision, recall, and F1 measure are available on the respective Web site.

Two different experimentations have been performed to compare H-MATCH with FOAM and OLA, respectively. The goal of each test is to compare two OWL ontologies in order to identify the pairs of matching elements. A list of expected mappings is also specified for evaluating the results provided by the tools in terms of precision, recall, and F1 measure. For what concern precision and recall we refer to the definitions provided in Section 5.1, while *F1 measure* is derived from the classical definition of Information Retrieval [30] and it is defined as follows:

$$F1 = \frac{2pr}{p + r} \tag{11}$$

where p and r represent precision and recall measures, respectively. For each test, we execute H-MATCH under many different matching policies varying the adopted matching model, the linguistic affinity weight W_{la}, the threshold t, and the kind of mapping determined (i.e., one-to-one, one-to-many). For each test, we report the H-MATCH policy that provides the best results in terms of precision, recall, and F1 measure and we compare such results with the corresponding results produced by the observed tool.

Comparison with FOAM. The test case adopted in this comparison has been selected from the FOAM Web site and regards the animal domain. In Table 7, we show the results produced by FOAM and H-MATCH, respectively. The FOAM measurements are obtained by submitting the test case to the FOAM Alignment Web Service [16]. The point of maximum precision of H-MATCH is obtained with the matching policy ⟨intensive, 0.8, 0.7, one-to-one⟩, while the points of maximum recall and F1 measure are obtained with the policy ⟨deep, 0.5, 0.3, one-to-one⟩. We can observe that FOAM has better results than H-MATCH for what regards precision, while recall and F1 measure are a little bit higher for H-MATCH. In general, these results confirm the impression we stressed in the experiments of Section 5.1: deep and intensive matching can ensure appreciable results in terms of precision and recall when W_{la} and t are properly set. In particular, precision increases with high values both for linguistic affinity weight and threshold (e.g., $W_{la} = 0.8$ and $t = 0.7$), while low values for both these

[14] http://www.iro.umontreal.ca/∼owlola/

[15] FOAM and OLA are also discussed in Section 6 where an analytical comparison with H-MATCH is provided.

[16] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/service.htm

	Precision	Recall	F1 measure
FOAM			
Point of maximum precision	1.0	0.58	0.74
Point of maximum recall	0.76	0.67	0.71
Point of maximum F1	1.0	0.58	0.74
H-MATCH			
Point of maximum precision	0.86	0.67	0.75
Point of maximum recall	0.78	0.78	0.78
Point of maximum F1	0.78	0.78	0.78

Table 7. Comparison of H-MATCH and FOAM results

parameters (e.g., $W_{la} = 0.5$ and $t = 0.3$) are required for obtaining high values of recall. Finally, we note that the one-to-one mapping strategy is always associated to maximum measures of H-MATCH. Such a mapping strategy allows to obtain more balanced results of precision and recall, thus it positively affects the F1 measures.

Comparison with OLA. The test case adopted in this comparison has been selected from the *Ontology Alignment Contest* [17] and regards the comparison of a reference bibliographic ontology [18] with the more complex Karlsruhe ontology [19]. In Table 8, the results of the comparison between H-MATCH and OLA are summarized. Precision, recall, and F1 measure related to OLA on this test

	Precision	Recall	F1 measure
OLA			
	0.5	0.31	0.38
H-MATCH			
Point of maximum precision	0.82	0.74	0.78
Point of maximum recall	0.82	0.74	0.78
Point of maximum F1	0.82	0.74	0.78

Table 8. Comparison of H-MATCH and OLA results

case are obtained from [8] where the authors indicate that the tool configuration has the intention to led to the best alignment with respect to precision. For what concern H-MATCH, the points of maximum precision, recall, and F1 measure are all obtained when the adopted matching policy is ⟨intensive, 0.8, 0.8, one-to-one⟩.

[17] http://co4.inrialpes.fr/align/Contest/
[18] http://co4.inrialpes.fr/align/Contest/101/onto.rdf
[19] http://co4.inrialpes.fr/align/Contest/303/onto.rdf

We note that H-MATCH can provide appreciable results both in terms of precision and recall. With respect to our considerations in Section 5.1, we note that the H-MATCH configuration for the point of maximum recall is anomalous. In this test case, this is due to the fact that the expected mappings are strongly dependent from linguistic features. This means that when the linguistic affinity weight is high (e.g., $W_{la} = 0.8$), even if the threshold has a high value (e.g., $t = 0.8$), the accuracy of H-MATCH can achieve relevant results.

6 Related Work

In this section, we perform a comparative analysis of the state of the art tools for ontology matching, providing also a critical comparison with H-MATCH. We perform the comparison in the light of three main criteria: i) the ontology representation formalism adopted by the tools; ii) the semantic complexity supported by the matching process of each tool; iii) the mechanism adopted by each tool for the composition of different similarity measures. For comparison we have selected *PROMPT* [12], *FOAM/QOM* [10, 27, 31], *OLA* [8], *S-Match* [11, 23], *GLUE* [7], and *COMA++* [22]. PROMPT is a framework for multiple ontology management; FOAM/QOM is a tool to full- or semi-automatically align two or more ontologies; OLA is an ontology alignment tool tailored to OWL Lite ontologies; S-Match is a semantic matchmaker tailored to graph-like structures; GLUE is an approach based on machine learning techniques for schema and ontology matching purposes; COMA++ is a combined framework for schema and ontology matching.

Ontology Representation Formalism. In Table 9, we compare the different selected tools with respect to the internal formalism adopted for the ontology representation and with the ontology languages supported. In particular, we show which OWL dialect is supported by each tool and which OWL features are considered in the matching process. The tools presented in Table 9 generally adopt an internal representation, either a tree-based or a graph-based model, of the ontology contents, which in several cases is defined to capture the features of OWL. S-Match and GLUE do not refer explicitly to OWL, but their reference model is compatible with OWL Lite. With respect to the ontology representation formalism, H-MATCH is in line with the state of the art systems, in that it refers to a graph-based model and provides direct support for OWL.

Features of the Matching Process. Table 10 shows the comparison of the tools with respect to the ontology elements, the linguistic features, and the contextual features that characterize the matching process in each tool. With respect to linguistic features, we compare the tools by taking into account the techniques adopted for determining linguistic similarities and to the level of linguistic analysis provided by each tool. Contextual features refer to the number and type of semantic relations among concepts that are used by each tool in

	OWL Dialect	Internal representation
PROMPT	OWL Lite OWL DL (partial) OWL Full (partial)	Frame-based Graph-based
FOAM/QOM	OWL Lite OWL DL	Karlsruhe Ontology Model [27] (Graph-based)
OLA	OWL Lite	OL-Graph (Graph-based)
S-Match	OWL Lite (no specific support for OWL)	Tree-based
GLUE	OWL Lite (no specific support for OWL)	Tree-based
COMA++	OWL Lite	Graph-based
H-MATCH	OWL Lite OWL DL (partial) OWL Full (partial)	H-Model (Graph-based)

Table 9. Comparison on ontology representation

order to determine the contextual similarity between two concepts. Regarding ontology elements that can be matched, all the tools perform the matching process by taking into account concepts and properties. Instances are considered in PROMPT, FOAM/QOM, OLA, GLUE, and COMA++. In the case of H-Match, the choice of considering concepts and properties is motivated by the fact that the algorithm is conceived for matching knowledge requests in the context of open networked systems, where probe queries are used with the aim of acquiring new knowledge from other peers. In fact, a probe query provides basically the schema-level description of one or more concepts of interest. However, H-Match can be easily extended to consider also instance level information in the ontology matching process. This can be achieved by exploiting a combination of linguistic and property value similarity measures already available in the intensive matching model. Considering linguistic features, we distinguish between tools that exploit an external dictionary or a thesaurus taking into account terminological relationships and domain knowledge for linguistic matching, and tools that rely on the syntactic features of labels and identifiers through string matching. PROMPT and GLUE do not adopt any external support for the linguistic analysis. FOAM and COMA++ adopt domain specific vocabularies that can be used for refining the matching results obtained by syntactic analysis. H-Match, as COMA++, S-Match, and OLA, adopts a language-based approach, in particular for the preprocessing of compound terms. S-Match, OLA, and H-Match are similar also with respect to the intensive use of WordNet for the linguistic matching. With respect to contextual features, all the tools take into account the semantic relations holding between concepts. S-Match adopts a logic-based approach that exploits the relations between concepts for automatic reasoning purposes. All the tools refer to property domain and range as well as to taxonomic relations among concepts. FOAM/QOM takes also into account property

	Ontology elements	Linguistic features		Contextual features
		Type of matching	External dictionary	
PROMPT	Concepts Properties Instances	Syntactic (String matching)	-	Property domain Property range Kind-of (Among concepts)
FOAM/QOM	Concepts Properties Instances	Syntactic (String matching)	Domain specific dictionary	Property domain Property range Property hierarchy Same-as Kind-of Individual identity
OLA	Concepts Properties Instances	Syntactic (String matching) Language- based (tokenization, compound terms)	WordNet	Property domain Property range Kind-of
S-Match	Concepts	Syntactic (String matching) Language- based (lemmatization, tokenization, compound terms)	WordNet	Same-as Kind-of Mismatch Overlapping
GLUE	Concepts Properties Instances	-	-	Property domain Property range Same-as Kind-of
COMA++	Concepts Properties Instances	Syntactic (String similarity) Language- based (tokenization)	Domain specific dictionary	Property domain Property range Same-as Kind-of
H-MATCH	Concepts Properties	Language- based (Tokenization, compound terms)	WordNet	Property domain Property range Same-as Kind-of Part-of Associates

Table 10. Comparison on semantic complexity

hierarchies and identity relations among instances. H-MATCH takes into account property domain and range, and the semantic relations of same-as, kind-of, part-of, and associates in the context of concepts. In particular, the part-of and associates

relations are useful to deal with object-oriented ontology specifications, where these relations are typically used. H-MATCH and FOAM/QOM adopt a similar strategy in discarding some kind of contextual features in order to increase the matching performance. The main difference between the two is related to the way the strategy is enforced. In H-MATCH, the strategy is chosen at run-time, by selecting the most adequate matching model that has to be adopted at a given invocation time.

Similarity Measures Composition. In Table 11, we compare the different approaches with respect to: i) the mechanism adopted for deriving a comprehensive similarity value out of the different similarity measures and ii) the mechanism adopted for cutting off the useless results. With respect to the similarity mea-

	Similarity measure composition	Cut-off
PROMPT	Cumulative approach	Highest value
FOAM/QOM	Weighted sum Process iteration	Threshold-based
OLA	Iterative process	Highest value
S-Match	Logic-based (SAT)	-
GLUE	Machine learning approach	Probability-based
COMA++	Average value Iterative approach	Highest value
H-MATCH	Weighted sum	Threshold-based

Table 11. Comparison on similarity measure composition

sures composition, PROMPT, OLA, and GLUE, even if in different ways, adopt a cumulative and iterative strategy for deriving the comprehensive degree of similarity between two ontology elements. In S-Match the structural matching is seen as a logic proof based on the previously determined relations among concept labels. COMA++ and FOAM are frameworks based on the idea of combining different measures of similarity. In COMA++ the similarity measures are composed in a comprehensive measure by evaluating their average value. In FOAM, the strategy is to perform a weighted sum of the different similarity measures, where the factors are functionally computed. H-MATCH adopts a weighted sum between linguistic and contextual measures of similarity, where the weights associated with linguistic and contextual affinity are constant. The problem of cutting off the results that are not used to determine mappings is typical of the approaches that provide a measure of similarity between ontology elements. S-Match is based on the idea to discover a semantic relation between two elements of different ontologies, so that there is no need of a cutting off mechanism. In GLUE, mappings represent the probability that an instance of a given element

is an instance also of another element. GLUE chooses the most probable mapping, that is the mapping between the elements with the highest probability to represent the same real object in the domain. In PROMPT and OLA, the matching value used for determine the mappings is the highest value that is computed between two elements by means of a cumulative strategy. FOAM/QOM, COMA++, and H-Match provide a measure of similarity between ontology elements in the range $[0, 1]$. In COMA++, mappings are determined between the elements with the highest matching values. In FOAM and H-Match, the cut-off mechanism is based on a threshold that is set as a parameter of the algorithm.

7 Concluding Remarks

In this paper, we have presented the H-Match algorithm and related techniques for matching of independent ontologies in open networked systems. H-Match has been implemented and used for probe query processing in the framework of the Helios networked system for supporting dynamic knowledge discovery and ontology-addressable content retrieval in peer-based systems [19, 20]. The novel contributions and distinguishing features of H-Match can be summarized as follows:

- Fully-automated matching process that can be used as: i) a matchmaker engine of a peer for matching probe queries against peer ontologies for knowledge discovery as occurs in the Helios open networked system; ii) a conventional ontology matching tool for the alignment of two independent ontologies;
- Capability of satisfying as a whole both general requirements of ontology matching per se and peculiar requirements of matching in open networked context. This is achieved by providing a wide spectrum of metrics suited for dealing with many different matching scenarios where the number and type of ontology features that can be exploited during the matching process is not known in advance;
- Capability of being dynamically configured for adaptation to the semantic complexity of the ontologies to be compared. This is achieved by automatically selecting the matching configuration according to a policy embedded in the incoming request which can vary, also for the same ontology, each time a new request is submitted to the system.

Future work will regard the enrichment of the ontology features supported by H-Match; the semantic query routing; the semantic community definition. With respect to the first issue, we will consider also instance-level information of ontology specifications in the matching process of H-Match, starting from current intensive matching techniques. Other two research issues are related to the use of H-Match for enforcing semantic query routing and for semantic community formation, which are hot research topics in open networked systems. Some initial results on these issues are presented in [32, 33].

Acknowledgments. The authors would like to thank the anonymous referees for their detailed and insightful comments. A special acknowledgment is due to Gianpaolo Racca for his invaluable collaboration to the development of H-MATCH in the framework of the HELIOS project. This paper has been partially funded by "Wide-scalE, Broadband, MIddleware for Network Distributed Services (WEB-MINDS)" FIRB Project funded by the Italian Ministry of Education, University, and Research, and by NoE INTEROP, IST Project n. 508011 - 6th EU Framework Programme.

References

[1] Broekstra, J., et al.: A Metadata Model for Semantics-Based Peer-to-Peer Systems. In: Proc. of the 1st WWW Int. Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID 2003), Budapest, Hungary (2003)

[2] Nejdl, W., et al.: EDUTELLA: a P2P Networking Infrastructure Based on RDF. In: Proc. of the 11th Int. World Wide Web Conference (WWW 2002), Honolulu, Hawaii, USA (2002)

[3] Mitre, J., Navarro-Moldes, L.: P2P Architecture for Scientific Collaboration. In: Proc. of the 13th Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2004), Modena, Italy, IEEE Computer Society (2004) 95–100

[4] Iamnitchi, A., Ripeanu, M., Foster, I.T.: Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations. In: Proc. of the 1st Int. Workshop on Peer-to-Peer Systems IPTPS 2002, Cambridge, MA, USA (2002) 232–241

[5] Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration. In: Proc. of the 23rd ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 2004), Paris, France (2004) 241–251

[6] Motik, B., Maedche, A., Volz, R.: A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications. In Springer, ed.: Proc. of Confederated Int. Conferences DOA, CoopIS and ODBASE 2002, Irvine, California, USA (2002) 1082–1099

[7] Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to Map between Ontologies on the Semantic Web. In: Proc. of the 11th Int. World Wide Web Conference (WWW 2002), Honolulu, Hawaii, USA (2002) 662–673

[8] Euzenat, J., Loup, D., Touzani, M., Valtchev, P.: Ontology Alignment with OLA. In: Proc. of the 3rd ISWC Workshop on Evaluation of Ontology-based Tools (EON 2004), Hiroshima, Japan (2004)

[9] Do, H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. In: Proc. of 28th Int. Conference on Very Large Databases (VLDB 2002), Hong Kong, China (2002)

[10] Ehrig, M., Sure, Y.: Ontology Mapping - An Integrated Approach. In: Proc. of the 1st European Semantic Web Symposium, Heraklion, Greece, Springer Verlag (2004) 76–91

[11] Giunchiglia, F., Shvaiko, P.: Semantic Matching. Knowledge engineering review **18** (2003) 265–280

[12] Noy, N.F., Musen, M.A.: The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. International Journal of Human-Computer Studies **59** (2003) 983–1024

[13] Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal **10** (2001) 334–350
[14] Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches. Journal on Data Semantics (JoDS) (2005)
[15] Noy, N.F.: Semantic Integration: a Survey of Ontology-based Approaches. SIGMOD Record Special Issue on Semantic Integration (2004)
[16] Kalfoglou, Y., Schorlemmer, M.: Ontology Mapping: the State of the Art. The Knowledge Engineering Review **18** (2003)
[17] INTEROP - Network of Excellence: State of the Art and State of the Practice Including Initial Possible Research Orientations. Deliverable D8.1, NoE INTEROP - IST Project n. 508011 - 6th EU Framework Programme (2004)
[18] Smith, M.K., Welty, C., McGuinness, D.L., (eds.): OWL Web Ontology Language Guide (2004) World Wide Web Consortium (W3C), http://www.w3.org/TR/owl-guide/.
[19] Castano, S., Ferrara, A., Montanelli, S., Zucchelli, D.: HELIOS: a General Framework for Ontology-based Knowledge Sharing and Evolution in P2P Systems. In: Proc. of the 2nd DEXA Int. Workshop on Web Semantics (WEBS 2003), Prague, Czech Republic, IEEE Computer Society (2003)
[20] Castano, S., Ferrara, A., Montanelli, S.: Dynamic Knowledge Discovery in Open, Distributed and Multi-Ontology Systems: Techniques and Applications. In: Web Semantics and Ontology. Idea Group (2005) To Appear.
[21] Castano, S., De Antonellis, V., De Capitani Di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 277–297
[22] Aumueller, D., Do, H., Massmann, S., Rahm, E.: Schema and Ontology Matching with COMA++. In: Proc. of SIGMOD 2005 - Software Demonstration, Baltimore, USA (2005)
[23] Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: an algorithm and an implementation of semantic matching. In: Semantic Interoperability and Integration, Schloss Dagstuhl, Germany (2005)
[24] Miller, G.A.: WordNet: A Lexical Database for English. Communications of the ACM (CACM) **38** (1995) 39–41
[25] Ouksel, A.M., Naiman, C.F.: Coordinating Context Building in Heterogeneous Information Systems. Journal of Intelligent Information Systems **3** (1994) 151–183
[26] Lauer, M.: Designing Statistical Language Learners: Experiments on Noun Compounds. In: Proc. of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995), Cambridge, Massachusetts, USA (1995) 47–54
[27] Ehrig, M., Staab, S.: QOM - Quick Ontology Mapping. In: Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004), Hiroshima, Japan (2004)
[28] Page, L., Brin, S., Motwani, R., Winograd, T.: The Pagerank Citation Ranking: Bringing Order to the Web. Technical report, Computer Science Department, Stanford University (1998)
[29] Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S.: Searching the web. ACM Transactions on Internet Technology (2001)
[30] Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley (1989)
[31] Ehrig, M., Haase, P., Stojanovic, N., Hefke, M.: Similarity for Ontologies - A Comprehensive Framework. In: Proc. of the 13th European Conference on Information Systems, Regensburg, Germany (2005)

[32] Castano, S., Ferrara, A., Montanelli, S., Pagani, E., Rossi, G.P., Tebaldi, S.: On Combining a Semantic Engine and Flexible Network Policies for P2P Knowledge Sharing Networks. In: Proc of the 1st DEXA Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems (GLOBE 2004), Zaragoza, Spain, IEEE Computer Society (2004) 529–535

[33] Castano, S., Montanelli, S.: Semantic Self-Formation of Communities of Peers. In: Proc. of the ESWC Workshop on Ontologies in Peer-to-Peer Communities, Heraklion, Greece (2005)

A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems

Jordi Conesa and Antoni Olivé

Universitat Politècnica Catalunya
Departament de Llenguatges i Sistemes Informàtics
Jordi Girona 1-3 E08034 Barcelona (Catalonia)
{jconesa|olive}@lsi.upc.edu

Abstract. In the past, most conceptual schemas of information systems have been developed essentially from scratch. Currently, however, several research projects are considering an emerging approach that tries to reuse as much as possible the knowledge included in existing ontologies. Using this approach, conceptual schemas would be developed as refinements of (more general) ontologies. However, when the refined ontology is large, a new problem that arises using this approach is the need of pruning the concepts in that ontology that are superfluous in the final conceptual schema. This paper proposes a new method for pruning ontologies in this approach. We also show how to adapt the method to prune ontologies in other contexts. Our method is general and it can be adapted to most conceptual modeling languages. We give the complete details of its adaptation to the UML. On the other hand, the method is fully automatic. The method has been implemented. We illustrate the method by means of its application to a case study that refines the Cyc ontology.

1 Introduction

Most conceptual schemas of information systems have been developed essentially from scratch. The current situation is not very different: most industrial information systems projects are being developed using a methodology that assumes that the conceptual schema is created every time from scratch. However, it is well-known that substantial parts of conceptual schemas can be reused in different projects, and that such reuse may increase the conceptual schema quality and the development productivity [21].

Several research projects explore alternative approaches that try to reuse conceptual schemas as much as possible [5, 18, 29, 31]. The objective is similar to that of projects in the artificial intelligence field that try to reuse ontologies. There are several definitions of the term "ontology". We adopt here the one proposed in [12, 34], in which an ontology is defined as the explicit representation of a conceptualization. A conceptualization is the set of concepts (entities, attributes, processes) used to view a domain. An ontology is the specification of a conceptualization in some language. In this paper, we consider a conceptual schema as the ontology an information system needs to know.

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 64-90, 2006.

Ontologies can be classified in terms of their level of generality into[13]:

- Top-level ontologies, which describe domain-independent concepts such as space, time, etc.
- Domain and task ontologies which describe, respectively, the vocabulary related to a generic domain and a generic task.
- Application ontologies, which describe concepts depending on a particular domain and task.

We call top-level, domain and task ontologies general ontologies. One example of general ontology is Cyc [16].

General ontologies can play several roles in conceptual modeling [31]. One of them is the base role. We say that a general ontology plays a base role when it is the basis from which the conceptual schema is developed. In general, the development requires three main activities [10]: refinement, pruning and refactoring which are reviewed in section 3. The objective of the refinement activity is to extend the base ontology with the particular concepts needed in a conceptual schema, and that are not defined in that ontology.

In general, when the base ontology is large, the extended ontology cannot be accepted as the final conceptual schema because it includes many superfluous concepts. The objective of the pruning activity is then to prune the unnecessary concepts. In this paper, we propose a new method for pruning ontologies in the development of conceptual schemas. To the best of our knowledge, ours is the first method that is independent of the conceptual modeling language used and of the base ontology. The method can be used in other contexts as well, and we will show that it has several advantages over similar existing methods. Our method can be adapted to most languages, and we give the complete details of its adaptation to the UML [25]. We illustrate the method by means of its application to a case study that refines the Cyc ontology.

The structure of the paper is as follows. In the next section we present the case study used to exemplify our approach. Section 3 reviews the three main activities in the development of a conceptual schema from a base ontology, with the objective of defining the context of the pruning activity, the focus of this paper. Section 4 presents the pruning method we propose and proves it is correct. Section 5 compares our method with similar ones. Section 6 extends our method to make it independent of the selection strategy used to identify the concepts which are of direct interest for the information system. Finally, Section 7 gives the conclusions and points out future work.

2 Case Study

In the case study we create the conceptual schema of a recipe information system by refining the Cyc ontology. The information base must represent information about:

- *Recipes*: A recipe is a guide that explains how to create a given meal. They are published in documents written by one or more authors. Each recipe also indicates which ingredients are necessary to create the described meal for a given number of persons.

- *Ingredients*: A given quantity of an ingredient consists of one or more quantities of distinct nutrients.
- *Restaurants*: A restaurant is an organization whose main activity is to serve and prepare meals. Each restaurant offers a list of dishes available for a meal. The dishes are prepared by cookers. A restaurant can only offer the meals its cookers know prepare. Restaurants are located in cities. The name of a restaurant must be unique in the city where it is located.
- *Menus*: Restaurants offer menus, which are composed for a subset of the list of dishes. The menus must have at least one first dish, one second dish and one dessert. The price of a menu cannot exceed the addition of the individual prices per dish.

The information system must answer queries such as:

- Kilocalories of an ingredient.
- Amounts of lipid, carbohydrate, mineral, protein, vitamin, water and cholesterol an aliment has.
- For a given city, all the restaurants whose cookers have published a recipe.
- The recipe of a given meal with the lower number of calories.
- The restaurant of a given city that offers a given meal at the lowest price.
- All the vegetarian menus offered in a given city.
- For a given restaurant, the cheapest combination of first dish, second dish, and dessert.

More details will be given when they arise. The complete details of the case study are reported in [7].

3 The Context

In this section we briefly review the three activities required to develop a conceptual schema from a general ontology: refinement, pruning and refactoring. Normally, these activities will be performed sequentially (see Fig. 1), but an iterative development is also possible [10].

3.1 Refinement

Normally, a general ontology O_G will not include completely the conceptual schema CS required by a particular information system. The objective of the refinement activity is then to obtain an extended ontology O_X such that:

- O_X is an extension of O_G, and
- O_X includes the conceptual schema CS.

The refinement is performed by the designer. S/he analyzes the IS requirements, determines the knowledge the system needs to know to satisfy those requirements, checks whether such knowledge is already in O_G and, if not, makes the necessary extensions to O_G, thus obtaining O_X.

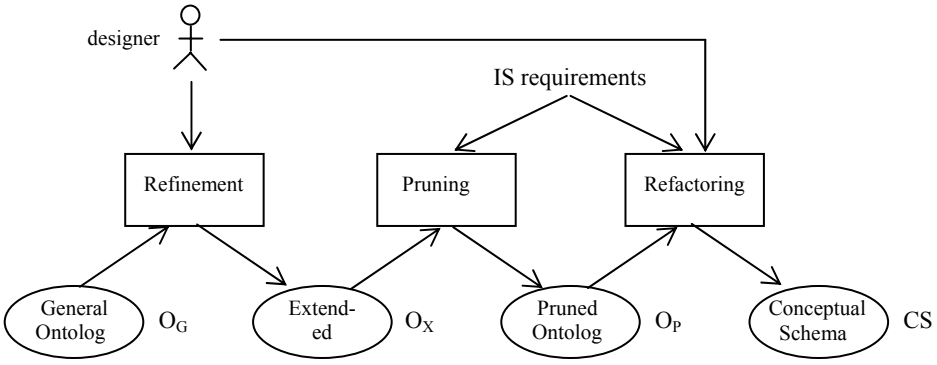


Fig. 1. The three activities in the development of conceptual schemas from general

In our case study, we adopted as general ontology O_G *OpenCyc* [26], the public version of the *Cyc* ontology. *OpenCyc* includes over 2900 entity types and over 1300 relationship types. Even if these numbers are large (and even larger in other ontologies such as *Cyc*) it is likely that additional entity or relationship types may be needed for the CS of a particular IS.

For example, our case study deals with recipes, its ingredients and the nutrients that compose those ingredients. The concept *Nutrient* exists in the base ontology, but their specializations into *Mineral*, *Lipid*... do not exist in OpenCyc. Then, we have to add a concept for each nutrient type: *Mineral*, *Lipid*, *Protein*, *Carbohydrates*, *Vitamin* and *Water-Ingestible* (see figure 2). Note that *Water-Ingestible* is also a *Drink*.

In our system, quantities of *EdibleStuff* must be expressed in some reference unit (such as gram). For this purpose we have defined attribute *referenceUnit* of type *UnitOfMeasure* (which is a datatype already defined in *OpenCyc*).

We need a concept that represents all kind of edible stuff element, because *EdibleStuff* represents also nutrients, and *Food* does not represent the condiments or preservatives that can be considered as ingredients. Then, we define an entity type called *NonNutrientEdibleStuff*. We define this type between *EdibleStuff* and its children: *CerealFood*, *FoodIngredientsOnly* and *FoodOrDrink*.

The nutritional composition of recipe ingredients is represented in the association between *NonNutrientEdibleStuff* and *Nutrient*. The association is reified in order to represent the quantity of nutrient included in the base quantity of *NonNutrientEdibleStuff*. For example *"100 gr. of rice have 7.3 gr. of proteins"*, where *rice* is an instance of *NonNutrientEdibleStuff* , with *baseQuantity* 100 gr., and the *nutrientQuantity* of proteins is 7.3 gr.

The complete refinement of *OpenCyc* for the case study is described in [7]. In summary, we have added twelve entity types (*Mineral, Lipid, Protein, CarboHydrates, Vitamin, Water-Ingestible, NonNutrientEdibleStuff, Recipe, RecipeDocument, FirstDish, SecondDish, Dessert, Menu, CateringCompany*), nine attributes (attributes *referenceUnit* of *EdibleStuff*, *baseQuantity* of *NonNutrientEdibleStuff*, *nutrientQuantity* of *NutritionalComponent* shown in Figure

2) and eight associations (one of them is *NutritionalComponent* in Figure 2). We have also added two association refinements and six general integrity constraints.

3.2 Pruning

Normally, an extended ontology O_X will contain many irrelevant concepts for a particular information system. The objective of the pruning activity is then to obtain a pruned ontology O_P such that:

- O_P is a subset of O_X, and
- O_P includes the conceptual schema *CS*, and
- The concepts in O_X but not in O_P would have an empty extension in the information system, or they are unnecessary for the information system.

In the case study, we find that the *OpenCyc* ontology contains thousands of concepts irrelevant for recipes. For example, the entity and relationship types dealing with Chemistry. Our information system is not interested in these concepts and, therefore, their extension in the information base would be empty. The objective of the pruning activity is to remove such concepts from O_X. In the next section we present a method for the automatic pruning of ontologies. The input of the method is either the formal specification of the IS requirements (domain events, queries) or the explicit definition of the concepts (entity and relationship types) of interest.

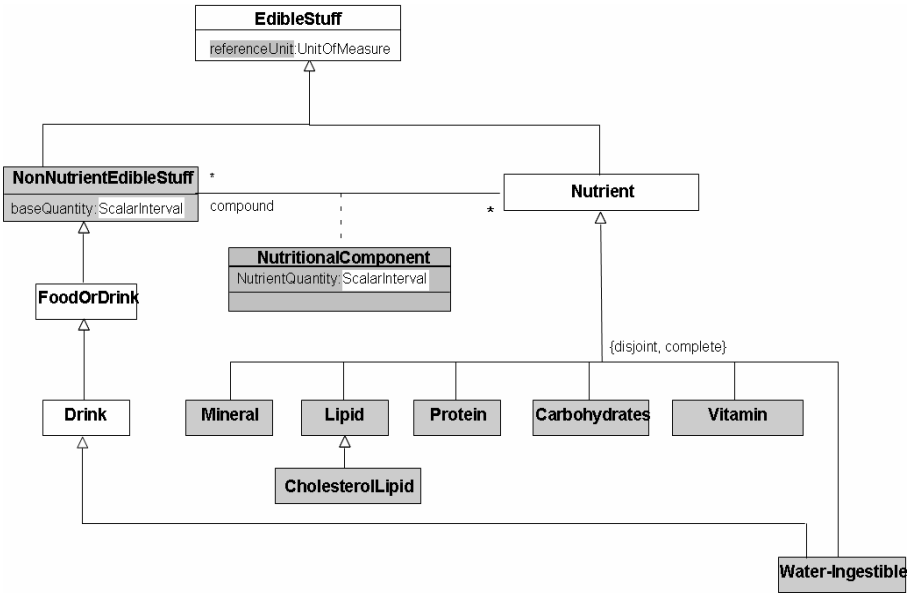


Fig. 2. Partial refinement of OpenCyc in the case study. The grayed boxes are refined concepts.

3.3 Refactoring

Normally, a pruned ontology O_P cannot be accepted as a final CS because it can be improved in several aspects. The objective of the refactoring activity is then to obtain a conceptual schema *CS* that is externally equivalent to O_P yet improves its structure. The purpose of ontology (or conceptual schema) refactoring is equivalent to that of software refactoring [11]. The refactoring is performed by the designer, but important parts of the activity can be assisted or automated, provided that the IS requirements are formalized. Refactoring consists in the application of a number of refactoring operations to parts of an ontology. Many of the software refactoring operations can be adapted to conceptual modeling, but this will not be explored in this paper.

4 Pruning the Extended Ontology

In this section, we define the problem of pruning the extended ontology and we propose a new method for its solution. The starting point of the pruning activity is an extended ontology O_X and the functional requirements of the IS. We explain also the adaptation of the problem and the method to the UML, currently one of the most widely used languages for conceptual modeling.

4.1 The Extended Ontology

We assume that, in the general case, an ontology O_X consists of sets of the following elements [33]:

- Concepts. There are two kinds of concepts:
  - Entity types.
  - Relationship types. We will denote by $R(p_1{:}E_1,\ldots,p_n{:}E_n)$ a relationship type R with participant entity types E_1, …, E_n playing roles p_1, …, p_n respectively.
- Generalization relationships between concepts. We denote by $IsA(C_1,C_2)$ the generalization relationship between concepts C_1 and C_2, where C_1 is the subtype and C_2 the super type. IsA^+ will be the transitive closure of IsA. We admit multiple specialization and multiple classification.
- Integrity constraints[1].

Adaptation to the UML. In the UML an ontology O_X consists of sets of the following elements (see Figure 2):

- Concepts:
  - Entity types.
  - Data types.
  - Attributes.
  - N-ary associations.

[1] The generalization relationships are (inclusion) constraints also, but we give them a special treatment due to its prominent role in taxonomies and in conceptual modeling.

  - Association classes, which reify associations. An association class and its reifying association are a single element.
- Generalization relationships between de above concepts. Attributes cannot be generalized.
- Constraints.

In the UML, some constraints are predefined (they have a particular language construct) and others may be user-defined. In our method we deal with constraints of the following kinds:

- Cardinality constraints of associations and attributes.
- Completeness and disjointness of sets of generalizations.
- Redefinition of association ends and attributes (redefinition constraints). Figure 3 shows an example of association redefinition: the association *ObjectFoundInLocation* is redefined in *City*.
- General constraints. We assume that general constraints are defined by constraint operations and specified in the OCL, as explained in [23]. The adaptation of our method to constraints defined as invariants is straightforward. An example is the constraint that the name of a restaurant must be unique into the city where it is located. Its formal specification is:

```
Context FoodServiceOrganization::uniqueName() : TruthValue
 body: FoodServiceOrganization.allInstances()->forAll(o1,o2|
           (o1 <> o2 and o1.name = o2.name) implies
                                            o1.City<>o2.City)
```

In the case study, O_X consists of:

- 2,715 Entity types and 255 Data types.
- 255 Attributes and 1,397 Associations.
- 6 general integrity constraints.

4.2 Concepts of Direct Interest

Usually, the extended ontology O_X will be (very) large, and only a (small) fraction of it will be needed for the CS of a particular IS. The objective of the pruning activity, as we will define it below, is to remove some non-needed elements from O_X.

The pruning activity needs to know which concepts from O_X are of direct interest in the IS. A concept is of direct interest in a given IS if its users and designers are interested in representing its population in the Information Base of the IS or inferring information from it. Our pruning method needs to know the concepts of direct interest, independently of how they have been obtained. We study in section 6 how to use several selection strategies to select the concepts of direct interest in an easy and reusable way.

When the functional requirements of an IS are formally specified, then the concepts of direct interest *CoI* may be automatically extracted from them [31]. The details of the extraction process depend on the method and language used for that specification. We explain here the process when the IS behavior is specified by system operations, as is done in many methods such as Larman's method [15], the B method [1] or Fusion [6]. A similar process can be used when the behavior is specified by statecharts, event operations or other equivalent methods.

In general, the formal specification of a system operation consists of:

- A signature (name, parameters, and result). The types of the parameters and the result are entity types defined in O_X.
- A set of preconditions. Each precondition is a boolean expression involving concepts defined in O_X.
- A set of postconditions. As above, each postcondition is a boolean expression involving concepts defined in O_X.

The concepts of direct interest *CoI* are then defined as:

- The types of the parameters and result of the system operations.
- The concepts appearing in the pre or postconditions.

In some cases the formal specification may not be available or may be incomplete. In these cases, the designers may wish to define the concepts *CoI* explicitly or to add new concepts to those determined from the functional specification.

If a relationship type is a concept of direct interest then we require that its participant entity types are in *CoI* also. Formally, we say that a set of concepts of direct interest *CoI* is *complete* if for each relationship type $R(p_1{:}E_1,\ldots,p_n{:}E_n) \in CoI$ the participant entity types $\{E_1, \ldots, E_n\} \subset CoI$.

In O_X there may be some concepts that generalize those in *CoI* and which are not part of *CoI*. We are interested in these generalized concepts because they may be involved in constraints that affect instances of the concepts *CoI*. To this end, we call set of *generalized* concepts of interest $G(CoI)$ the concepts of a complete set *CoI* and their generalizations. Formally:

$$G(CoI) = \{c \mid c \in CoI \vee \exists sub\ (\mathrm{IsA}^{+}(sub,c) \wedge sub \in CoI)\}$$

Adaptation to the UML. The adaptation is straightforward. We assume that the pre/postconditions are written in the OCL. For example, consider the system operation *howMuchCholesterol*, whose purpose is to return the quantity of cholesterol of a given meal. Its formal specification may be:

```
Context System::howMuchCholesterol (f:NonNutrientEdibleStuff):
                                                    NonNegativeNumber
  body: f.NutritionalComponent->
        select(nutrient.oclIsType(CholesterolLipid)).
                                              nutrientQuantity->sum()
```

The *CoI* inferred from this operation are: *NonNutrientEdibleStuff*, *NonNegativeNumber*, *NutritionalComponent*, *Nutrient*, *CholesterolLipid, NutrientQuantity* and *ScalarInterval*.

4.3 Constrained Concepts

We call constrained concepts of an integrity constraint *ic*, $CC(ic)$, the set of concepts appearing in the formal expression of *ic*. By abuse of notation we write $CC(O)$ to denote the set of concepts constrained by all the integrity constraints defined in ontology *O*. Formally,

$$CC(O) = \{c \mid c \text{ is a concept} \wedge c \in O \wedge \exists ic\ (ic \text{ is a constraint} \wedge ic \in O \wedge c \in CC(ic))\}$$

Adaptation to the UML. If *ic* is a cardinality constraint of an attribute or association, then *CC*(*ic*) will be the attribute or association, and the entity and data types involved in it.

If *ic* is a completeness constraint with a common supertype *super* and subtypes $sub_1, \ldots, sub_n$, then $CC(ic) = \{super, sub_1, \ldots, sub_n\}$.

A disjointness constraint with a common supertype *super* and subtypes $sub_1, \ldots, sub_n$, corresponds to $n(n-1)/2$ disjunction constraints each of which constraints two subtypes, sub_i and sub_j, and *super*. Strictly speaking, these constraints do not involve the supertype *super*, but in the UML they are attached to sets of generalizations having the same supertype.

If *ic* is a redefinition of an association or attribute then *CC*(*ic*) will be the redefined association or attribute, and the entity and data types involved in the association or attribute.

The constrained concepts of a general constraint will be the entity types, data types, attributes, associations and association classes appearing in the OCL expression that defines it. For example, if *uniqueName* is the general constraint defined in 4.1, and figure 3 represents the relevant fragment of the O_X for this integrity constraint, then CC(*uniqueName*) = {*FoodServiceOrganization, TruthValue, name, SomethingExisting, ObjectFoundInLocation, City*}. Note that the entity types *SomethingExisting* and *ObjectFoundInLocation* have been selected because they participate directly in the selected relationship types, which are *name* and *ObjectFoundInLocation* respectively.

4.4 The Pruning Problem

Given an extended ontology O_X and a complete set of concepts of direct interest *CoI*, as explained above, the pruning problem consists in obtaining a pruned ontology O_P such that:

(a) The elements in O_P are a subset of those in O_X. We do not want to add new elements to O_X in the pruning activity. Additions can be done in the refinement or in the refactoring activities.

$$\forall c\ (c \in O_P \rightarrow c \in O_X)$$

(b) O_P includes the concepts of direct interest *CoI*. These concepts must be included in O_P. The information system needs such concepts in order to perform its functions.

$$\forall c\ (c \in CoI \rightarrow c \in O_P)$$

(c) If C_1 and C_2 are two concepts in O_P and there is a direct or indirect generalization relationship between them in O_X, then such relationship must also exist in O_P. Otherwise, the child concept C_1 would lose the relationship types and constraints defined in C_2 or in its parents. Formally:

$$\forall c_1,c_2\ (c_1 \in O_P \wedge\ c_2 \in O_P \wedge \text{IsA}^+(c_1,c_2) \in O_X\ \rightarrow \text{IsA}^+(c_1,c_2) \in O_P)$$

(d) O_P includes all constraints defined in O_X whose constrained concepts are in $G(CoI)$. The rationale is that the constraints in O_X which constraint the Information Base of O_P must be part of it. The constraints in O_X that involve one or more concepts not in $G(CoI)$ cannot be enforced and, therefore, are not part of O_P.

$$\forall_{IC} ({}_{IC} \in O_X \wedge CC(_{IC}) \in G(CoI) \rightarrow {}_{IC} \in O_P)$$

(e) O_P is syntactically correct [17] , that is, it is a valid instance of the conceptual modeling language in which it is specified (metamodel).

(f) O_P is minimal, in the sense that if any of its elements is removed from it, the resulting ontology does not satisfy (b-e) above.

For each O_X and CoI there is at least an ontology O_P that satisfies the above conditions and, in the general case, there may be more than one.

To the best of our knowledge, there does not exist a method that obtains O_P automatically in a context similar to ours. In what follows we describe a method for the problem. In the next section we will compare it with existing similar methods.

4.5 The Pruning Algorithm

Our algorithm obtains O_P in three steps. The algorithm begins with an initial ontology O_0 which is exactly O_X (that is, $O_0 := O_X$) and obtains O_P. The steps are:

- Pruning irrelevant concepts and constraints. The result is the ontology O_1.
- Pruning unnecessary parents. The result is the ontology O_2.
- Pruning unnecessary generalization paths. The result is O_P.

Pruning irrelevant concepts and constraints. The concepts of direct interest for the IS are given in the set CoI, and $G(CoI)$ is the set of concepts in which the IS is directly or indirectly interested in. However, O_0 may include other concepts, which are irrelevant for the IS. Therefore, in the first step we prune from O_0 all concepts which are not in $G(CoI)$, that is, we prune the set of concepts:

$$\text{IrrelevantConcepts} = \{c \mid c \text{ is a concept } \wedge c \in O_0 \wedge c \notin G(CoI)\}$$

Pruning a concept C implies pruning of all generalization relationships $IsA(C_1,C)$ and $IsA(C,C_1)$ in which C participates. The super types and subtypes C_1 of C are not affected by the pruning of C.

Similarly, we prune the constraints in O_0 that are not relevant for the IS, because they constraint one or more concepts not in $G(CoI)$. That is, we prune the set of constraints:

$$\text{IrrelevantConstraints} =$$

$$\{ic \mid ic \text{ is a constraint} \wedge ic \in O_0 \wedge \exists c\, (c \in CC(ic) \wedge c \notin G(CoI)\}$$

The result of this step is the ontology O_1:

$$O_1 = O_0 - \text{IrrelevantConcepts} - \text{IrrelevantConstraints}$$

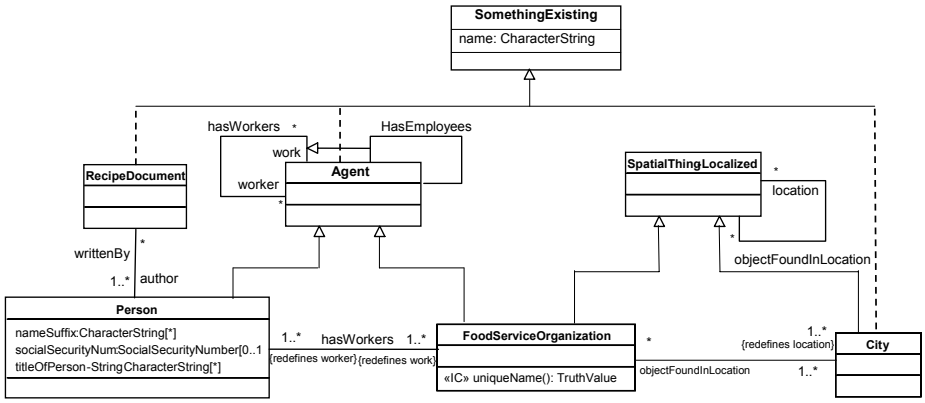


Fig. 3. Fragment of the extended ontology with relevant elements and constraints

In the example of Figure 3, we have that *HasWorkers* is a concept of interest and, therefore, $\{HasWorkers\} \subseteq G(CoI)$. However, *HasEmployees*, a subtype of *HasWorkers*, is not an element of $G(CoI)$ and therefore it is pruned in this step. Likewise, *Person* is a concept of interest but its subtypes (*Student*, *HumanChild*, *HumanAdult*, *FemalePerson*, *MalePerson*, etc. not shown in Figure 3) are not, and therefore they are also pruned in this step. The same happens to "lateral" concepts such as *Atom* or *Electron*.

In the case study, after the application of this step we have an ontology O_1 consisting of:

- 140 Entity types and 22 Data types.
- 15 Attributes and 30 Associations.
- 6 general integrity constraints.

Pruning unnecessary parents. After the previous step, the concepts of the resulting ontology (O_1) are exactly $G(CoI)$. However, not all of them are needed in the CS. The concepts strictly needed are given by:

$$\text{NeededConcepts} = CoI \cup CC(O_1)$$

The other concepts are potentially not needed. Formally:

$$\text{PotentiallyUnneededConcepts} = G(CoI) - \text{NeededConcepts}$$

We can prune the parents of *NeededConcepts* which are not children of some concept in *NeededConcepts*. Formally,

$$\text{UnnecessaryParents} = \{c \mid c \in \text{PotentiallyUnneededConcepts} \wedge \neg \exists c' (c' \in \text{NeededConcepts} \wedge \text{IsA}^{+}(c,c'))\}$$

As we have said before, the pruning of a concept implies the pruning of all generalization relationships in which that concept participates.

The result of this step is the ontology O_2:

$$O_2 = O_1 - \text{UnnecessaryParents}$$

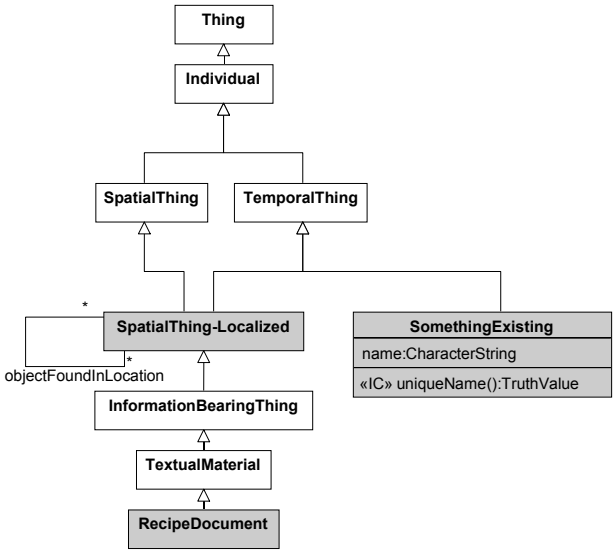


Fig. 4. Detecting and deleting the unnecessary parents. The grayed boxes are needed concepts

In Figure 4, examples of unnecessary parents are the entity types *SpatialThing*, *TemporalThing, Individual* and *Thing* which are the parents of *SpatialThing-Localized* and *SomethingExisting*. In the case study, *SpatialThing* neither is a needed concept of O_1, nor is a child of some needed concept, and therefore it is pruned in this step. The same happens for *Thing, Individual* and *TemporalThing*. Note that although the entity types *InformationBearingThing* and *TextualMaterial* are not unnecessary, they cannot be deleted, because of their common necessary parent *SpatialThing-Localized*.

In the case study, after the application of this step we have an ontology O_2 consisting of:

- 106 Entity types and 19 Data types.
- 15 Attributes and 11 Associations.
- 6 general integrity constraints.

Pruning unnecessary generalization paths. In some cases, the ontology O_2 may contain generalization paths between two concepts such that not all of them are necessary. The purpose of the third step is to prune these paths.

We say that there is a generalization path between C_1 and C_n if:

- C_1 and C_n are two concepts from O_2,
- $IsA^+(C_1,C_n)$ and
- The path includes two or more generalization relationships $IsA(C_1,C_2)$, …, $IsA(C_{n-1},C_n)$.

A generalization path $IsA(C_1,C_2)$, …, $IsA(C_{n-1},C_n)$ between C_1 and C_n is potentially redundant if none of the intermediate concepts C_2, …, C_{n-1}:

- Is member of the set $CoI \cup CC(O_2)$
- Is the super or the sub of other generalization relationships.

A potentially redundant generalization path between concepts C_1 and C_n is redundant if there are other generalization paths between the same pair of concepts. In this case, we prune the concepts C_2, …, C_{n-1} and all generalization relationships in which they participate. Note that, in the general case, this step is not determinist.

The output of this step is the pruned ontology, O_P.

Figure 5 shows four generalization paths between the concepts of direct interest *Restaurant* and *SomethingExisting*. None of these paths can be deleted, because at least one of their elements participate in more than one generalization relationship. Concretely, the entity types *FoodServiceOrganization*, *CommercialServiceOrganization*, *CommercialOrganization*, *LegalAgent, Organization* and *SocialBeing*. However, there exist three specialization paths between the entity types *Organization* and *FoodServiceOrganization*: P_1={*Organization*, *FoodAndBeverageOrganization*, *FoodServiceOrganization*}, P_2={*Organization*, *Service*, *FoodServiceOrganization*} and P_3={*Organization*, *CommercialOrganization, FoodServiceOrganization*}. The intermediate concepts of all the paths are not members of $CoI \cup CC(O_2)$. Furthermore, *FoodAndBeverageOrganization* is the only intermediate concept which does not participate in other generalization relationships, so the path P_1 is the only path that is potentially redundant. Therefore, it can be pruned from the ontology. After this, the algorithm will detect another duplicated specialization path between the concepts *Organization* and *CommercialServiceOrganization* composed by *{Organization, ServiceOrganization and CommercialServiceOrganization}*, and as a consequence the concept *ServiceOrganization* will be pruned.

In the case study, after the application of this step we have an ontology O_P consisting of:

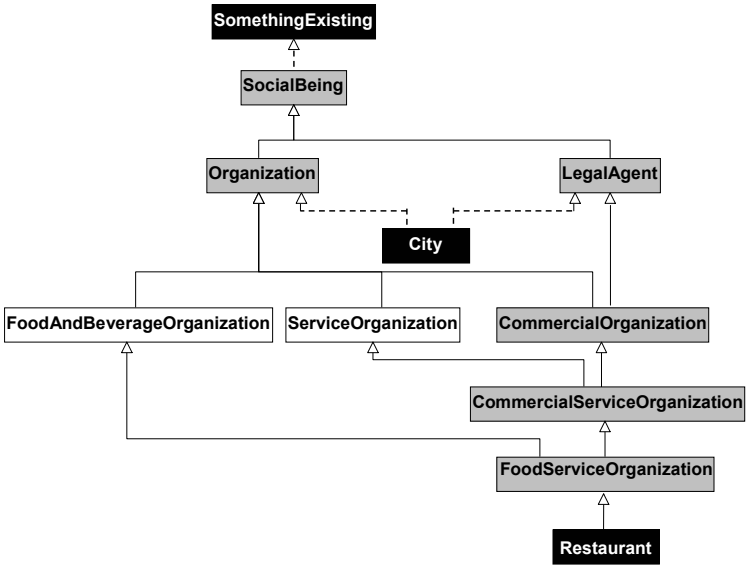


Fig. 5. Detecting and deleting the unnecessary duplicated paths between *Restaurant* and *Organization*. The white boxes are the concepts to prune and the black ones are the necessary concepts.

- 75 Entity types and 15 Data types.
- 15 Attributes and 11 Associations.
- 6 general integrity constraints.

4.6 Correctness of the Pruning Algorithm

In this section we argue that the above pruning algorithm is correct. We assume that the input to the algorithm is a syntactically correct extended ontology O_X and a set *CoI* of concepts of interest, with $CoI \subseteq O_X$. The pruning algorithm is correct if its output (the pruned ontology O_P) satisfies the conditions defined in section 4.4. In the following paragraphs we argue that O_P satisfies all of these conditions.

The elements in O_P are a subset of those in O_X. The algorithm only removes elements (concepts, constraints) from O_X. It never adds new elements. Therefore, in the general case, O_P will be a subset of O_X. In the rare case that all O_X concepts are of direct interest, then O_P and O_X would be the same.

O_P includes the concepts of direct interest *CoI*. None of the algorithm steps deletes any concept of direct interest:

- The pruning irrelevant concepts and constraints step removes the concepts not included in $G(CoI)$. $G(CoI)$ includes all the concepts of direct interest, therefore *CoI* concepts cannot be deleted in this step.
- The pruning of unnecessary parents step deletes a subset of the set of potentially unneeded concepts, which contains the concepts that are not of direct interest for the information system and do not appear in any relevant constraint. Obviously, this step cannot delete *CoI* concepts because of its exclusion of the potentially unneeded concepts set.
- The pruning unnecessary generalization paths step removes a subset of the potentially redundant elements set. This step cannot delete concepts of direct interest because that set does not include the *CoI* concepts.

Obviously, if no step can eliminate *CoI* concepts, then all *CoI* concepts will be included in the pruned ontology.

All O_P concepts with an *IsA* relationship in O_X, must also have an *IsA* relationship in O_P. None of the deletions done in the pruning steps results in a loss of specialization path between two needed concepts:

- The pruning irrelevant concepts and constraints step removes the *IsA* relationships relating irrelevant concepts. The irrelevant concepts neither are of direct interest, nor have children of direct interest, so we can affirm that *all the specializations of an irrelevant concept are also irrelevant*. As a consequence, whenever an irrelevant element is deleted, all its children are deleted as well. Then, it is obvious that none specialization path between survival elements may be deleted.
- The pruning of unnecessary parents step deletes the *IsA* relationships that relate unneeded elements, which are elements without necessary parents. Then, when the method deletes an unneeded element all its parents are deleted as well. As a consequence, it is obvious that their deletion do not break any specialization path.

- The pruning unnecessary generalization paths step removes the specialization paths between two concepts which satisfy a set of conditions, one of which is that the whole path is redundant. Then, eliminating a generalization path implies that there exists another generalization path between the same elements, so it is impossible to break a generalization path in this step.

Therefore, we can say our pruning activity does not delete necessary generalization paths between O_P concepts.

***O_P* includes all constraints defined in *O_X* whose constrained concepts are in *G*(*CoI*).** The pruning irrelevant concepts and constraints is the only step that deletes constraints. In particular, this phase only deletes the constraints whose concepts includes one or more irrelevant concepts, which are exactly the concepts not included into *G*(*CoI*). Then, for definition, we can conclude all the constrained concepts of the survival constraints are members of the set *G*(*CoI*).

***O_P* is syntactically correct.** An ontology is syntactically correct if all the constructions used to describe it are compliant with the grammar of its ontology language. For instance, an UML ontology is syntactically correct if it is a valid instance of the UML's metamodel and satisfies all its integrity constraints, including the well-formedness rules (WFR).

We assume O_X is syntactically correct. Then, in order to prove that O_P is also syntactically correct, we must prove that all possible deletions of the pruning method preserve the syntactic correctness of the ontology. In the following, we prove this for the deletion operations over the ontology elements:

- Concepts: Deleting a concept, implies deleting also all the generalization relationships where it participates. On the other hand, in our method, the deletion of a concept that participates in a given relationship or integrity constraint, also implies deleting its related concepts. Therefore, it is not possible to delete any concept that participates either in a relevant relationship type or a relevant integrity constraint.
- Generalization relationships: If a concept uses a relationship type defined in any of its parents, a deletion of a taxonomic relationship between the concept and its parent, may result in a syntactically incorrect ontology, because the child may lose the referred relationship type. Nevertheless, this particular case cannot occur in our algorithm, because, as we proved before, it does not allow breaking the generalization path between concepts.
- Integrity Constraints: They only restrict the possible instantiations of the ontology, so their deletion will result in a new ontology, less restricted, but not syntactically incorrect.

As a consequence, we can say that if the O_X is correct, the O_P will be correct as well.

***O_P* is minimal**. In order to prove that O_P is minimal, we are going to see which violations can be produced by the elimination of each kind of O_P element:

- Concepts: The concepts of the pruned ontology may be:
  - Concepts of Direct Interest: We cannot delete these concepts, because they must be included in the O_P (condition *b*). Their deletion may also produce the

violation of condition *c* (if the concept participates in one generalization and one specialization) or *e* (if any relationship type or constraint uses it).
  - Needed Concepts which are not *CoI*: These concepts are needed because they participate in one or more relevant constraints. Then, their deletion produces a syntactically incorrect ontology, because they are referred to in some relevant constraints.
  - Other concepts: These concepts are necessary to maintain a generalization path between, at least, two necessary concepts. Therefore, their deletion will break a non redundant specialization path, violating condition *c* of the method. Their deletion can also violate condition *e*.
- Integrity Constraints: The integrity constraints of the pruned ontology are those which can be evaluated using only elements of the *G*(*CoI*) set. Therefore, we cannot delete any of them without violating condition *d*.
- Generalization/specialization relationships. They are part of a non redundant generalization/specialization path between two (or more) necessary concepts. Obviously, we cannot delete any of them, without violating condition *c*.

Therefore, we have proved that the removal of any of the pruned ontology elements results in the violation of at least one condition of the pruning method.

5 Comparison with Previous Work

The need for pruning ontologies has been described in several research works in the fields of information systems and knowledge bases development. We may mention Swartout et al. [32], Knowledge Bus [27], Text-To-Onto [14, 19], Ontology Derivation Rules [39], MOVE [3, 4, 38], the ODS (Ontology-Domain-System) approach [36], DODDLE-II [30, 40], Mena et al. [20], the Dynamic Ontologies [28, 37] and OntoLearn [22]. In the following we explain the main differences among the pruning methods; we present a table that summarizes their main characteristics, and finally, give some comments and comparisons on the pruning methods which are more related to ours.

Even if the above works differ in the context in which the need for pruning arises, the ontology language, the particular ontology used as base, or the selection of the concepts of interest, we believe that (at least parts of) our pruning method can be adapted to be used successfully in all those works. The reason are: (1) we deal with any base ontology; (2) our method can be adapted to any ontology language (in [8] we show the adaptation of our method to the OWL (Web Ontology Language) [2]); (3) we take into account the specificity of entity types, relationship types, generalizations and constraints present in all complete conceptual modeling languages; and (4) although we obtain the concepts of interest from the functional specifications, our method can use any selection strategy to obtain the concepts of direct interest, as we will see in the next section.

Usually the pruning methods are intended to be applied in a particular context. The ontology context determines mainly its foremost properties: 1) the base ontology the method is able to prune, 2) how the method selects the concepts of direct interest, and 3) how many elements are pruned. The methods that use pruning techniques to support the information systems development, which are *Knowledge Bus*, *Ontology*

derivation rules, *MOVE*, *Ontology Domain System* and our method, allow pruning more expressive ontologies than the others. These methods also tend to do a more effective pruning, because their pruned ontologies are used directly for humans, and obviously, humans cannot deal easily with large ontologies (with more than a thousand concepts). Furthermore, those pruning methods, with the exceptions of *Knowledge Bus* and our method, have not been defined to prune very large ontologies. Examples are the *ODS* approach, which is totally manual, or the *Ontology derivation rules*, which works very well for small ontologies, but with too manual intervention to make it usable with large ontologies such as *OpenCyc*. In these methods, the user tends to participate very actively in the selection of the concepts of direct interest. The rationale is that in this context the user knows all the concepts that are relevant for the final information system and that must exist in the final ontology.

The goal of the other methods, which are *Swartout et al*, *Text-to-onto*, *Ontolearn* and *DODDLE-II*, is the creation of a domain ontology, whose information will be used to support users in a given task. These methods use linguistic ontologies as a basis, which are less expressive than those used by the above methods, but contain more concepts that the other ones. For example, SENSUS ontology, which is the ontology used by Swartout, et al., has more than 50,000 concepts, while *OpenCyc* does not have more than 5,000 concepts. These methods have more efficient selection processes, this is because they use the semantic relationships (synonyms, antonyms, ...) among concepts that the linguistic ontologies have. These methods are not interested in generating very small pruned ontologies, because their pruned ontologies are used for programs to infer information, and then, they should contain the concepts of direct interest and all their related concepts. For this reason, these pruning methods are equivalent to the first step of our pruning method, with the exception of DODDLE, whose pruning activity contains also a restructuring step.

Table 1 shows a few characteristics of the main current pruning methods. For each method we give: 1) the base ontology the method uses; 2) whether or not the method takes into account the integrity constraints (in one case we are unsure about this); 3) how automatic the method is; 4) the selection strategy used for selecting the concepts of direct interest; and finally, 5) the efficiency of the pruning activity, that is how many elements the pruned ontology has.

In the following we give some additional comments on the works which are the more closely related to ours, and that describe a comparable pruning method.

The purpose of the *Ontology Derivation Rules* is to generate domain or application ontologies using a set of rules over a base ontology. The base ontology can be any ontology written in the UML language. The designer is responsible of selecting the concepts of direct interest by-hand, which are called permanent elements in this method. The pruning method also restructures the ontology to minimize its volume. The restructuring is done by applying a set of rules, such as *when a non permanent class c1 contains a permanent attribute a1 and there is a permanent class c2 child of c1, then move the attribute a1 to c2*. After applying these rules, the method generates all the possible associations among the permanent classes of the ontology, following the associative property of the UML associations. Once all these hybrid relationships are created, the designer must identify the relevant ones. Finally, the method uses this information to delete the ontology irrelevant elements, and asks to the designer for a name for the survival hybrid associations. The results of this method are quite good, but its process is too manual to be usable with medium and large ontologies.

Table 1. Comparison of the main current pruning methods

	Base ontology	Integrity Constraints	Automation	Selection Strategy	Final Ontology
Knowledge Bus [27]	*OpenCyc*	✓	TOTAL	By-hand	TOO LARGE
Swartout et al. [32]	*SENSUS*, but applicable to any ontology	✘	SEMI-AUTOMATIC	By-hand	TOO LARGE
Ontology Derivation Rules [39]	Any UML ontology	?	SEMI-AUTOMATIC	By-hand	DESIRED
MOVE [3, 4, 38]	Any	Cardinality constraints	SEMI-AUTOMATIC	By-hand	CUSTOMIZABLE
Text-to-Onto [14, 19]	Any RDF ontology	Predefined in the language	SEMI-AUTOMATIC	Automatic: using text-mining algorithms	LARGE
OntoLearn [22]	WordNet	✘	SEMI-AUTOMATIC	Automatic: using text-mining algorithms	LARGE
Ontology Domain System [36]	Any UML ontology	✓	NONE	By-hand	DESIRED
DODDLE – II [30, 40]	WordNet	✘	SEMI-AUTOMATIC	By-hand	LARGE
Our method [9]	Any	✓	TOTAL	Any	CLOSE TO DESIRED

The reason is the high number of hybrid relationships that appears in its pruning process, and the hard work of the designers in identifying which is relevant and which is not. The same results or better can be obtained with the execution of our pruning and refactoring activities. For example, with reference to our case study, an ontology with over 4000 concepts and over 30 concepts of direct interest, the method will generate several hundreds of anonymous hybrid relationships (note that in an ontology with the magnitude of OpenCyc it can exits easily a chain of relationships relating almost all the entity types of the ontology). In addition, as far as we know, it does not exist the formal definition of the whole method, so it is unclear the efficiency of the pruning method for real cases. Furthermore, our method is more automatic and efficient than this one, because the restructuring activity is done after the pruning method.

MOVE uses the *ontology derivation rules* approach to generate a view of a given ontology that satisfies a set of requirements. The method can prune any kind of ontology written in the IOS language. This language allows to represent concepts, attributes, binary relationships and cardinality constraints over relationship types and attributes. The concepts of direct interest (called "selected") are selected by-hand by the user. The user also has to select the concepts that cannot appear in the final ontology (called "unselected"). Then, the pruning is executed by taking into account four optimization schemas: 1) RCOS, which uses the ontology derivation rules to guaranty the final ontology satisfies the users requirements (the selection); 2) SCOS, which validates the semantic completeness of the ontology, that is, if a concept is defined using other concepts, we cannot delete the last without losing information of the former; 3) WFOS, which contains the rules that guarantees the syntactic correctness of the final ontology; and 4) TSOS, which guarantees the obtained ontology is the smallest that can be obtained. Up to now, as far as we know, only the two first phases have been defined, so their results are neither proved, nor guaranteed to be minimal. The pruning method may be customized by changing the given optimization schemas or adding new ones [38]. As in the previous case, this method can be compared to our pruning and refactoring activities together, and the same efficiency reasoning of the previous method can be applied also here.

The purpose of Swartout et al. is the development of specialized, domain specific ontologies from a large base ontology. The base ontology is SENSUS, a natural language based ontology containing well over 50,000 concepts. The elements of the ontology are only entity types and generalization relationships. The concepts of interest are assumed to be a set of entity types (called the "seed") selected explicitly by domain experts, and all entity types that generalize them. The pruning method corresponds roughly to our first step (pruning irrelevant concepts and constraints). Using our method, the domain experts could select the seed, as before, but also the generalized entity types of interest. The two other steps of our method could then be applied here, thus obtaining more specific domain ontologies.

The purpose of Knowledge Bus [27] is to generate information systems from application-focused subsets of a large ontology. The base ontology is Cyc, and the ontology language is CycL. The concepts of interest are assumed to be the set of entity types defined in a context (a subset of Cyc), also called the "seed" set, and all the entity and relationship types that can be "navigated" directly or indirectly from them. For example, with reference to Figure 3, if the seed set were only {*FoodServiceOrganization*} then all entity and relationship types shown in that figure would be considered concepts of interest. If we consider not only the fragment shown in that figure but the complete OpenCyc, then over 700 entity types and 1300 relationship types would be considered concepts of interest. The pruning method (called the sub-ontology extractor) corresponds here also to our first step (pruning irrelevant concepts and constraints). The result is that (as the authors recognize) many superfluous types are extracted from Cyc. Using our method, the domain experts can be more precise with respect to the concepts of interest. They could select the seed, as before, but also the generalized entity and relationship types of interest. The two other steps of our method could then be applied here as well, thus improving the specificity of the sub-ontology extraction process.

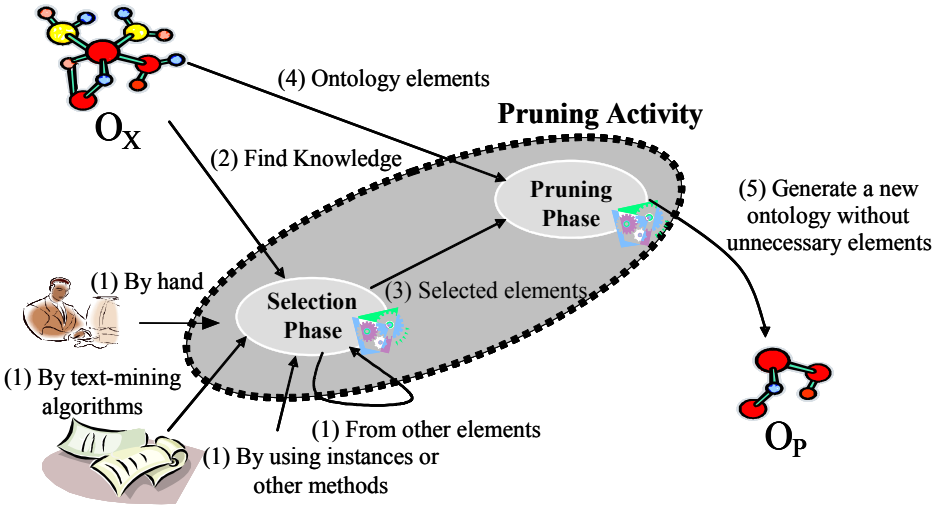


Fig. 6. Separating the pruning and selection phase in the pruning activity

6 Adapting Our Pruning Method to Different Selection Methods

Up to now, we have defined and solved the pruning problem in the context of developing the conceptual schema for an information system. In this section we show that our method can be used in other contexts as well.

Before pruning an ontology, it is necessary to select those elements that must be included in the final result. In our context, this selection is done using the information system requirements but in other contexts other selection strategies may be more suitable. For example, in the semantic web it can be necessary to select the elements using text mining algorithms [19], or manually [20]. In many methods, the selection of elements is an integral part of the pruning process. This implies that the selection strategy cannot be changed without re-implementing the pruning process. Here, we propose to separate the phases of selection and pruning (figure 6). This will allow us to do the pruning activity applicable for any strategy selection and able to reuse other selection methods.

In what follows we present a taxonomy (summarized in Figure 7) that describes the different ways of concept selection. Then we study how to use the taxonomy to reuse selection methods written by others. Finally we use the taxonomy to classify the main actual pruning methods.

6.1 Taxonomy of Relevant Concepts Selection Methods

According to its granularity, selection methods may be classified as individual or composite selection. An **individual** selection (also known as primitive selection) computes a selection based on a single selection criteria, and may be classified into

manual or automatic selection. In the manual selection, the designer must select by hand the elements of O_X that are necessary to the final ontology. The manual selection may be classified into:

- Unassisted selection: this is the most usual selection method. The designer chooses the necessary concepts without any system assistance. This method is used in [3, 9, 27, 32, 38, 40], where the designer selects manually the set of concepts relevant for the final ontology.
- Assisted selection: The system supports the user by proposing concepts to select. This kind of selection is usually combined with other selection methods (composite selection). We can see an example in the last step of the Swartout et al. approach [32], in which the system may propose the selection of ontology subtrees.

In the automatic selection, the concepts of direct interest are selected automatically by the system. This kind of selection must use some information to detect automatically new concepts of direct interest. This information can be taken from:

- Other selected concepts: The concepts of direct interest previously selected are used to select new concepts. An example of this kind of selection can be seen in [27], where the set of selected classes (CoI) is used to obtain all the relationships applicable to the classes of the CoI set (that is, the relationships whose participants are contained into CoI).
- Other ontology elements: Sometimes the non concept elements of the ontology (the ones that are not entity types or relationship types: individuals, classification relationships, …) are used to select new concepts. This is one of the most forgotten techniques of selection on pruning algorithms, but we think that it may be interesting in some cases to obtain the concepts of direct interest from the instances of the ontology, its integrity constraints, or its generalization relationships.
- External sources: The concepts of direct interest may also be obtained from information that lies in external sources. This is one of the most common techniques to select concepts of direct interest in pruning algorithms. Examples of this kind of selection are [22, 35], where the concepts of direct interest are obtained applying text-mining algorithms to several documents. There is another example in

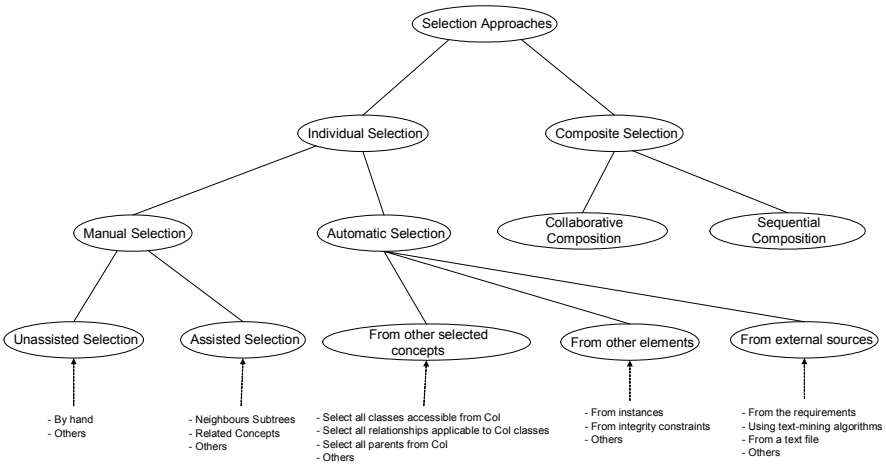


Fig. 7. Selection methods to detect the concepts of direct interest

the case study of this document, where the concepts of direct interest are detected automatically from the requirements of the IS, formalized by means of system operations [15] written in OCL [24].

Composite selection: A composite selection method includes more than one selection approach (that can be individual or composite). A composite selection may be:

- Collaborative composition: Several selection approaches are used collaboratively to detect the elements of direct interest. In this approach the outputs of the different selection approaches are evaluated to determine which concepts to select. Although this technique is not used nowadays in the pruning activity, we think it provides a very powerful way to detect the concepts of direct interest. On the other hand it seems that this selection technique needs a high participation of the ontology designer to define which elements to select, and this may be a drawback in the pruning of large ontologies.
- Sequential composition: A sequential composition is composed of a sequence of selection approaches, in which the output of each selection approach is the input of the next one. This technique is the most used at the moment. An example of this approach is Swartout et al. [32], where the selection process is a sequential composition of three individual selections: 1) a manual selection where the user selects without assistance a set of concepts of direct interest, 2) an automatic selection that selects all the parents of the elements selected in the previous process, and finally 3) a neighbour subtrees selection where the user can select subtrees whose neighbours have been selected in the previous steps.

6.2 Allowing General Purpose Selection

Current pruning approaches do not separate the selection and pruning phase. Therefore, the pruning methods are hooked to a selection strategy, which cannot be changed without re-implementing the pruning method. The problem grows when the pruning algorithm is specific to a selection strategy or a base ontology (its language or its structure). For example, a non generic pruning algorithm may contain a rule like "*delete a concept when none of its synonyms has been selected as relevant*". This rule is part of a selection strategy, in fact we may classify this rule in our taxonomy as a selection from other selected component. In addition, a strategy selection tends to be dependent to a given ontology. In the example the use of the synonym relationship, which is particular of linguistic ontologies, makes the pruning algorithm not applicable to all ontologies.

Separating the selection and the pruning phase makes the pruning algorithm more concise and independent of both selection strategies and the ontology used. In the previous example we may state the previous rule in the selection phase "*select the synonyms of the relevant elements*", and the pruning phase will contain a rule like "*delete the non relevant elements*". It is obvious that this way of defining a pruning algorithm is more generic than the previous one.

This separation reports also reusability benefits, because it allows to reuse individual selection approaches defined and implemented by others. To define a composite selection strategy, an ontology designer has to obtain the primitive methods (reusing them or developing them from scratch) needed in the composition,

and write a program that executes these primitive methods sequentially, giving the result of each method to the next one, and finally returning the results of the selection to the pruning phase.

Now that a taxonomy of selection is defined (see figure 7), it is possible to define a framework that supports the designer in the definition of selection strategies. A selection strategy, which combines several kinds of selection strategies, may be specified by means of a high level language based on the selection taxonomy.

We say our pruning method is generic, because the set *CoI*, which is necessary to our pruning activity, may be obtained as a result of applying any selection strategy that could be expressed as an instance of the presented taxonomy.

6.3 Expressing the Actual Pruning Methods as a Combination of Primitive Selection Methods

We think our taxonomy is complete with regards to the pruning methods defined until now in the literature. In order to validate this affirmation we show in this subsection how the selection phase of the main pruning methods can be expressed as an instance of our taxonomy.

Knowledge Bus

The selection begins with the selection by-hand of the relevant classes. Then, the system executes a fix point algorithm that selects all the classes that can be accessed from the relevant classes following relationships. Finally, all the associations whose participants have been selected in the previous steps are selected as well.

It is easy to see that the knowledge bus selection strategy may be represented by a Sequential Composition of: 1) an unassisted by hand method that selects the classes of direct interest (*CoI*). 2) An automatic selection that obtains the classes accessible from the *CoI* classes through relationships (*Select all the classes accessible from CoI*), and 3) another automatic selection that selects all the relationships whose participants were selected in the previous steps (*Select all relationships applicable from CoI*).

Swartout et al.

In this approach the relevant concepts for the target domain are manually selected by the user. Then, for each selected concept, the system automatically selects the elements contained in the path defined between the root of the ontology and the concept. After that, the designer may select some subtrees of the ontology such that almost all their neighbours (concepts with the same parents) have been selected, assuming that if all the neighbours of a concept have been selected, then the concept probably must be selected as well.

This selection process can be defined as a sequential composition of: 1) an unassisted by hand method that selects the concepts of direct interest, 2) an automatic selection that uses the previous one to obtain all the parents of the selected concepts (*Select all parents Of*), and 3) an assisted selection that assists the designer to select the needed ontology subtrees whose neighbours have been selected (*Neighbour Subtrees*).

Note that the first step is the same that the first step in Knowledge Bus, so both approaches may reuse the same implementation of the primitive selection method *by hand*.

Our Approach
The selection process of our method can be defined as a sequential composition of: 1) an automatic selection that selects all the concepts referred to in the requirements of the IS (*From the Requirements*), and 2) an unassisted by hand method that selects the rest of concepts necessary to the IS that were not selected in the previous step (this might be the same method used in Knowledge Bus and Swartout et al. approaches).

Due to space limitations we cannot define here all the pruning methods in terms of our taxonomy, but the application to the other pruning approaches is straightforward.

7 Conclusions

We have tried to contribute to the approach of developing conceptual schemas of information systems by reusing existing ontologies. We, as many others, believe that this approach offers a great potential for increasing both the conceptual schema quality and the development productivity.

We have focused on the problem of pruning ontologies. The problem arises when the reused ontology is large and it includes many concepts which are superfluous for the final conceptual schema. The objective of the pruning activity is to remove these superfluous concepts.

We have presented a new formal method for pruning an ontology. The input to our method is the ontology and the set of concepts of interest. When the functional requirements are formally specified, the concepts of interest can be automatically extracted from them. From this input, our method obtains automatically a pruned ontology, in which most of the superfluous concepts have been removed. We have shown that the method is correct.

We have formalized the method independently of the conceptual modeling language used. However, the method can be adapted to most languages, and we have shown the details of its adaptation to the UML. A prototype to prune UML ontologies has been implemented[2]. The adaptation of the pruning method to the OWL [2] is described in [8]. On the other hand, our method can be used with any ontology. The method has been illustrated by means of its application to a case study that refines the public version of the Cyc ontology. Our method improves on similar existing methods, due to its generality and greater pruning effectiveness.

The pruning method has been generalized in order to prune ontologies in other contexts. Changing the context of pruning application may result in a change of the way the concepts of direct interest are selected. Our pruning method is independent of the context, in the sense that it may be customized to be applied in any context and taking into account any way of selecting the concepts of direct interest.

[2] http://www.lsi.upc.es/~gmc/Downloads/jconesa/Program.zip

We plan to continue our work in (at least) two directions. First, we would like to implement our pruning method into a CASE tool. This will allow the designer to use the pruning method in a automatic and usable way. Finally, we plan to work on the activity that follows pruning: refactoring. The large amount of existing work on schema transformation can be "reused" for that purpose.

Acknowledgments

We would like to thank Jordi Cabot, Xavier de Palol, Dolors Costal, Cristina Gómez, Anna Queralt, Ruth Raventós, Maria Ribera Sancho and Ernest Teniente for their many useful comments to previous drafts of this paper.

This work has been partly supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

1. J. R. Abrial, *The B-Book*, 1996.
2. S. Bechhofer, F. v. 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/: W3C, December, 2003.
3. M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, and D. Taniar, "A Distributed Approach to Sub-Ontology Extraction," in *Proceedings of the 18th International Conference on Advanced Information Networking and Application*. Fukuoka, Japan, 2004.
4. M. Bhatt, C. Wouters, A. Flahive, W. Rahayu, and D. Taniar, "Semantic Completeness in Sub-ontology Extraction Using Distributed Methods," in *Proceedings of ICCSA 2004*, 2004, pp. 508-517.
5. S. Castano, V. D. Antonellis, and B. Zonta, "Classifying and Reusing Conceptual Schemas," in *ER'92*, vol. 645, *Lecture Notes in Computer Science*, G. Pernul and A. M. Tjoa, Eds., 1992, pp. 121-138.
6. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Guilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented Development. The Fusion Method*: Prentice Hall, 1994.
7. J. Conesa, "Ontology Driven Information Systems Development: Pruning and refactoring of ontologies. PhD Thesis (In preparation)," in *LSI - Llenguatges i Sistemes Informàtics*. Barcelona: UPC, 2005.
8. J. Conesa and A. Olivé, "A General Method for Pruning OWL Ontologies," in *ODBASE'04*, vol. 3291, *Lecture Notes in Computer Science*. Larnaca, Cyprus, 2004, pp. 981-998.
9. J. Conesa and A. Olivé, "Pruning Ontologies in the Development of Conceptual Schemas of Information Systems," in *ER2004*, *Lecture Notes in Computer Science*. Shangai, China, 2004.
10. J. Conesa, X. d. Palol, and A. Olivé, "Building Conceptual Schemas by Refining General Ontologies," in *DEXA'03*, vol. 2736, *Lecture Notes in Computer Science*, 2003, pp. 693 - 702.
11. M. Fowler, *Refactoring: Improving the Design of Existing Code*: Addison-Wesley, 1999.
12. T. R. Gruber, "Toward Principles for the Design of Ontolgies for Knowledge Sharing," in *International Journal of Human and Computer Studies*, vol. 43 (5/6), 1995, pp. 907 - 928.

13. N. Guarino, "Formal Ontology and Information Systems," in *Proc. FOIS'98*: IOS Press, 1998, pp. 3-15.
14. J.-U. Kietz, A. Maedche, and R. Volz, "A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet," in *Proceedings of EKAW-2000 Workshop*, *Springer Lecture Notes in Artificial Intelligence (LNAI)*, 2000.
15. C. Larman, *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*: Prentice Hall, 1998.
16. D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "CYC: Toward Programs With Common Sense," *Communications of the ACM*, vol. 33, pp. 30-49, 1990.
17. O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding Quality in Conceptual Modeling," *IEEE Software*, vol. 11, pp. 42-49, 1994.
18. M. Lloyd-Williams, "Exploiting Domain Knowledge During the Automated Design of Object-Oriented Databases," in *Proc. ER '97*, vol. 1331, *Lecture Notes in Computer Science*, D. W. Embley and R. C. Goldstein, Eds.: Springer, 1997, pp. 16-29.
19. A. Maedche and S. Staab, "Ontology Learning for the Semantic Web," *IEEE Inteligent Systems*, vol. 16, pp. 72 - 79, 2001.
20. E. Mena, J. A. Royo, A. Illarramendi, and A. Goñi, "An Agent-based Approach for Helping Users of Hand-Held Devices to Browse Software Catalogs," in *In Cooperative Information Agents VI, 6th International Workshop CIA 2002*. Madrid, Spain, 2002.
21. H. Mili, F. Mili, and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE TSE*, vol. 21, pp. 528-562, 1995.
22. R. Navigli, "Automatically Extending, Pruning and Trimming General Purpose Ontologies," in *International Conference on Systems, Man and Cybernetics*. Tunisy, 2002.
23. A. Olivé, "Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages," in *ER'03*, vol. 2813, *Lecture Notes In Computer Science*, 2003, pp. 349 - 362.
24. OMG, "OMG Revised Submission, UML 2.0 OCL,"
25. OMG, *UML 2.0 Superstructure Specification, 2.0 edition*: OMG, August, 2003.
26. OpenCyc, "OpenCyc, the public version of Cyc,"http://www.opencyc.com/
27. B. J. Peterson, W. A. Andersen, and J. Engel, "Knowledge Bus: Generating Application-focused Databases from Large Ontologies," in *KRDB'98*, *CEUR Workshop Proceedings*, 1998, pp. 2.1-2.10.
28. G. Sacco, "Dynamic Taxonomies: A Model for Large Information Bases," *IEEE Transactions on Data and Knowledge Engineering*, vol. 12, pp. 468-479, 2000.
29. V. C. Storey, R. H. L. Chiang, D. Dey, R. C. Goldstein, and S. Sundaresan, "Database Design with Common Sense Business Reasoning and Learning," *ACM TODS*, vol. 22, pp. 471-512, 1997.
30. N. Sugiura, M. Kurematsu, N. Fukuta, N. Izumi, and T. Yamaguchi, "A Domain Ontology Engineeering Tool with General Ontologies and Text Corpus," in *Proceedings of the 2nd Workshop on Evaluation of Ontology based Tools*, 2003, pp. 71-82.
31. V. Sugumaran and V. C. Storey, "Ontologies for conceptual modeling: their creation, use, and management," *Data & Knowledge Engineering*, vol. 42, pp. 251-271, 2002.
32. W. R. Swartout, R. Tatil, K. Knight, and T. Russ, "Toward Distributed use of Large-Scale Ontologies," in *Proc. 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop, Canada*, 1996.
33. M. Uschold, "Knowledge level modelling: concepts and terminology," *The Knowledge Engineering Review*, vol. 13, pp. 5-29, 1998.
34. M. Uschold and M. Gruninger, "Ontologies: Principles, Methods and Applications," *The Knowledge Engineering Review*, vol. 11, pp. 93-136, June, 1996.
35. R. Volz, R. Studer, A. Maedche, and B. Lauser, "Pruning-based Identification of Domain Ontologies," *Journal of Universal Computer Science*, vol. 9, pp. 520-529, 2003.

36. X. Wang, C. W.Chan, and H. J.Hamilton, "Design of Knowledge-Based Systems with the Ontology-Domain-System Approach," in *Software Engineering and Knowledge Engineering*, vol. 859. Ischia, Italy: ACM Press, 2002, pp. 233 - 236.
37. D. Wollersheim and W. Rahayu, "Methodology For Creating a Sample Subset of Dynamic Taxonomy to Use in Navigating Medical Text Databases," in *Proceedings of the 2002 International Symposium on Database Engineering & Applications*, 2002, pp. 276-289.
38. C. Wouters, T. Dillon, W. Rahayu, E. Chang, and R. Meersman, "Ontologies on the MOVE," in *Proceedings of the 9th International Conference on Database Systems for Advanced Applications*, vol. 2973, *Lecture Notes in Computer Science*. Jeju Island, Korea, 2004, pp. 812 - 823.
39. C. Wouters, T. S. Dillon, J. W. Rahayu, and E. Chang, "A Practical Walkthrough of the Ontology Derivation Rules," in *DEXA'02*, vol. 2453, *Lecture Notes in Computer Science*, 2002, pp. 259-268.
40. T. Yamaguchi, "Constructing Domain Ontologies Based on Concept Drift Analysis," in *IJCAI-99. Workshop on Ontologies and Problem-Solving Methods*, 1999.

XSLTGen: A System for Automatically Generating XML Transformations Via Semantic Mappings*

Stella Waworuntu and James Bailey

NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{stellavw, jbailey}@cs.mu.oz.au

Abstract. XML is rapidly emerging as a dominant standard for representing and exchanging information. The ability to transform and present data in XML is crucial and XSLT is a relatively recent programming language, specially designed to support this activity. Despite its utility, however, XSLT is widely considered a difficult language to learn.
In this paper, we present a novel system called XSLTGen, an automatic XSLT Generator. This system automatically generates an XSLT stylesheet, given a source XML document and a desired output HTML or XML document. It allows users to become familiar with and learn XSLT stylesheets, based solely on their knowledge of XML or HTML. Our method for automatically generating XSLT transformations is based on the use of semantic mappings between the input and output documents. We show how such mappings can be first discovered and then employed to create XSLT stylesheets. The results of our experiments show that XSLTGen works well with a number of different varieties of XML and HTML documents.

1 Introduction

XML (eXtensible Markup Language) [4] is rapidly emerging as the new standard for data representation and exchange on the Web. As the medium for communication between applications, an ability to transform XML to other data representations is essential. This data conversion can be performed by a language called XSLT (eXtensible Stylesheet Language: Transformations) [7]. XSLT plays an important role in transforming XML documents into HTML, text, or other types of XML documents. In this paper, we focus on transformations from XML to HTML (HyperText Markup Language) [26], since we are motivated by publishing applications, but our techniques are also applicable for XML to XML transformations.

Despite its capability of transforming documents having a certain structure, e.g. XML documents, into an HTML representation, XSLT is a relatively new

* Parts of results of this paper appeared in [30].

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 91–129, 2006.

language, and is widely considered difficult to learn [19]. Rendering to HTML using XSLT requires skills and knowledge of both XSLT programming and Web page styling. Our focus in this paper is on the development of algorithms and tools that can generate XSLT stylesheets automatically, given a user provided source XML document and a user provided target HTML document.

Automatic XSLT generation is an extremely useful facility for students and Web developers in the process of learning XSLT. With such a tool, they are able to see and understand how the XSLT stylesheet should look, in order to transform a particular XML document into a desired HTML document. In addition, this tool can also be useful for aiding the XSLT development process. Programmers may use the automatically generated XSLT stylesheet as a starting point for something more complex.

In this paper, we present *XSLTGen: An Automatic XSLT Generator*, a novel system that automatically generates an XSLT stylesheet, given a source XML document, and a desired output HTML document. The generated XSLT stylesheet contains rules needed to transform the given XML document to the HTML document and can be applied to other XML documents with similar structure, or to the given XML document after updates to it have been applied. The important feature of this system is that users can generate an XSLT stylesheet based solely on their knowledge of XML and HTML, i.e. users only need to create a desired output HTML document based on an input XML document. Moreover, users do not have to know anything about the syntax or programming of XSLT, or be aware of the XSLT rule generation process.

A naive solution to the problem of automatic XSLT generation is to create an XSLT stylesheet consisting of only one template rule, whose pattern matches the XML root element and whose template contains the HTML document markup (in other words create a stylesheet which is very specific to the desired output). This naive approach has a major drawback in terms of reusability. This stylesheet is specific for transforming the given XML document only and could not be used to transform other XML documents having similar structure. In contrast, we are interested in generating a more generic stylesheet, which can then be reused to transform other XML documents with similar structure. A more detailed discussion and illustration of both the naive and more generic solutions is given in Sect. 3.

In this paper, we show how XSLT stylesheets can be automatically generated by first discovering semantic mappings between the input and output. We make the following contributions:

1. We describe the use of text matching and structure matching for finding semantic mappings between an XML document and an HTML document generated from this XML document.
2. We introduce *sequence checking* to the matching context, and show that it enables the system to discover 1-m mappings between the two documents, in addition to its capability of discovering 1-1 mappings.
3. We describe a fully automatic XSLT generation system that generates XSLT rules based on the semantic mappings found.

4. We describe a technique for improving the accuracy of the XSLT stylesheet generated, that examines the differences between the original HTML document and the one produced by applying the generated XSLT stylesheet back to the XML document.
5. We conduct experiments on a number of datasets to validate the matching accuracy and quality of XSLT stylesheets generated by XSLTGen. The results show that XSLTGen works well with different varieties of XML and HTML documents.
6. This is the first paper that we are aware of that describes completely automatic XSLT generation from a source XML and a target HTML document.

This paper is structured as follows. In Sect. 2, we give a brief overview of XSLT and describe important definitions and terminology used in the paper. In Sect. 3, we present the definition of the problem of generating XSLT automatically. In Sect. 4, we present an overview of XSLTGen and then describe the details of each technique involved in the XSLTGen system. In Sect. 5, we conduct experiments to measure the similarity between the original HTML document and the one generated by the resulting XSLT stylesheet. We then evaluate the performance of XSLTGen. In Sect. 6, we survey related work in the field. Finally in Sect. 7, we offer concluding remarks and discuss possible extensions to the current system.

2 Background and Terminology

In this section, we provide a brief overview of XSLT. For a more complete description of XSLT, the reader is directed to [7], and [16]. Following this, we describe important definitions and terminology used in this paper.

2.1 XSLT

XSLT is a "high-level declarative language for transforming XML documents into other XML documents or HTML documents" [7]. The dominant features of XSLT as a declarative language are that it is a rule-based language, where the rules are not arranged in any particular order, and it is side-effect free which enables XSLT rules to be called any number of times and in any order.

XSLT uses XML syntax. The root element is `<xsl:stylesheet>`, which must include a namespace declaration for XSLT. The optional `<xsl:output>` element tells the type of the target document. The root element is then filled with *template rule*s, which describe how to transform elements in the source document, in this case, XML document. Each template rule consists of two parts: a *pattern* and a *template*. The pattern describes which XML element nodes should be processed by the rule. In some cases, patterns are specified using XPath expressions [8]. On the other hand, the template describes the HTML structure that should be generated when nodes that match the pattern are found. In an XSLT stylesheet, a template rule is represented by an `<xsl:template>` element. The

pattern is the value of its `match` attribute, and the template is the element's content.

The template may contain a sequence of text nodes and *literal result element*s[1] to be copied to the output, and instructions to be executed according to the rules of the particular instruction. The complete set of XSLT instructions can be found in Sect. 7 of [7]. The two XSLT instructions that will be used very often in the XSLT stylesheets generated by XSLTGen are `<xsl:value-of>` and `<xsl:apply-templates>`. The `<xsl:value-of>` instruction extracts the data content of an XML element and inserts it into the output. It has a `select` attribute which consists of a pattern. The `<xsl:apply-templates>` instruction finds all nodes that match the `select` attribute pattern, and processes each node in turn by applying the template rule that matches the node.

As described earlier, XSLT is a language specifically designed for transforming the structure of XML documents. The transformation process works as follows. A list of nodes from the source document is processed to create a result tree fragment. The result tree is constructed by processing a list containing just the root node. Within a list of source nodes, each list member is processed in order and the result tree structures are appended. A node is processed by finding all the template rules with patterns that match the node, and choosing the best amongst them; the template of the chosen rule is then instantiated with the node as the current node and with the list of source nodes as the current node list. A template typically contains instructions that select an additional list of source nodes for processing. This process is continued recursively until the list of source nodes is empty.

2.2 Definitions and Terminology

We now present definitions and terminology that will be be useful in describing the semantic mappings and their generation.

Let $m:\ (m.x, m.h)$ denote a mapping between an element in the source XML document to one or more elements in the output (destination) HTML document. The XML component of m is denoted by $m.x$, while the HTML component of m is denoted by $m.h$. Note that $m.h$ can be a sequence (concatenation) of one or more elements, while $m.x$ must be a single element. e.g. (`poem`,`body`) and (`line[1]`,`li++br`) are both mappings. If $m.x$ is an ATTRIBUTE_NODE, $owner(x)$ is defined to be the XML node that owns $m.x$.

Two mappings m_1 and m_2 are *distinct* if both $m_1.x.name \neq m_2.x.name$ and $m_1.h.name \neq m_2.h.name$ (where *name* refers to the tag name of the appropriate element); otherwise they are *coincide*. e.g. (`poem`,`body`) and (`line[1]`,`li++br`) are distinct mappings, whereas (`poem`,`body`) and (`poem`,`li++br`) coincide.

Exact mappings and substring mappings will be used later to describe textual correspondences in the XML source and HTML output. An *exact mapping* is a mapping e such that, $e.x$ is a TEXT_NODE or an ATTRIBUTE_NODE, $e.h$

[1] A literal result element is an element within a template that cannot be interpreted as an XSLT instruction.

is a TEXT_NODE, and $e.x.value \equiv e.h.value$ (where *value* refers to the string content of the appropriate element and $\equiv$ indicates that the two strings being compared have exactly the same characters located at the same positions, after leading and trailing whitespaces have been removed from the strings. e.g. (`"XML is good"`,`"XML is good"`) is an exact mapping.

A *substring mapping* is a mapping s such that, $s.x$ is a TEXT_NODE or an ATTRIBUTE_NODE, $s.h$ is a TEXT_NODE, and $s.x.value$ is a substring of $s.h.value$. In addition to this, the following conditions must be satisfied by a substring mapping in order to rule out meaningless cases, such as a mapping between an XML text "`the`" and an HTML text "`there`" since it is highly unlikely that an HTML text "`there`" is generated from an XML text "`the`":

1. if $s.x.value$ starts with a non-letter and non-digit character, then the character preceding its occurrence in $s.h.value$ (if any) must be either a letter or a digit; otherwise, the preceding character must be both non-letter and non-digit. And,
2. if $s.x.value$ ends with a non-letter and non-digit character, then the character following its occurrence in $s.h.value$ (if any) must be either a letter or a digit; otherwise, the following character must be both non-letter and non-digit.

(`"XML"`,`"XML is good"`) is an example of a substring mapping.

Special HTML elements are the elements `br` and `hr`, which are used to separate text in HTML document.

An *extra node* is a node that does not have any matching node in the other document. *Extra XML nodes* are those XML nodes that are ignored when generating the HTML document, while *extra HTML nodes* are HTML nodes that are added at the time the HTML document was generated and are not constructed from any part of the XML document. Examples of extra nodes are provided later in Table 2.

Let N be an XML or HTML node. We define the *precise node* of N, denoted by $precise(N)$, to be the node that is used to represent a transformation in the XSLT template rule match. For a mapping m, if $m.x$ is a TEXT_NODE, $precise(m.x)$ is the parent of $m.x$, whereas if $m.x$ is an ATTRIBUTE_NODE, $precise(m.x) = m.x$. Finding the precise node for $m.h$ is slightly more complicated since we need to look at the neighbourhood surrounding $m.h$, for the existence of *special* HTML elements. $precise(m.h)$ is discovered as follows:

1. If the next sibling of $m.h$ is a *special* HTML ELEMENT_NODE h_s, then $precise(m.h) = m.h$ `++` h_s.
2. If $m.h$ has no next sibling or the next sibling of $m.h$ is not *special* and $m.h$ has ELEMENT_NODE siblings, then $precise(m.h) = m.h$.
3. Otherwise, $precise(m.h)$ is the highest ancestor of $m.h$ such that each node lying between $precise(m.h)$ and $m.h$ path only has one *non-extra* child.

We also define the *sequence* of element N, denoted by $seq(N)$ to be a DTD[2] of N if N is the root of a document (e.g. In Figure 1, a possible DTD for the root element is `(author,date,title,stanza*)`). Otherwise, let D' be a DTD of the parent of N, then $seq(N)$ is equal to $trim_N(D')$. Since it is possible that D' involves symbols that represent elements other than N, the function $trim$ is employed here. $trim_N(E)$ removes both the largest prefix not containing the element N and the largest suffix not containing the element N from its argument E, which is a regular expression. Here, E is viewed as an ordered conjunction and so the largest prefix of E not containing N is the maximal prefix of conjuncts in E that don't contain N. Similar for the largest suffix. e.g. $trim_p(a, b, (e|f), p*, (p|a), a) = p*, (p|a)$.

3 Problem Formulation

In this section, we further formulate the problem of generating an XSLT stylesheet automatically from a source XML document and a desired output HTML document. We also discuss both the naive and non-naive ways of solving our problem.

Our primary focus in this paper is to automatically construct an XSLT stylesheet that transforms a source XML document to a desired output HTML document. Thus, the problem addressed in this paper can be stated as follows.

Problem statement. Given an XML document drawn from a class of XML documents, and an HTML document to be generated from the XML document, generate an XSLT stylesheet which contains rules needed to transform the given XML document into the HTML document and which also can be applied to other XML documents of the same document class.

The following example best illustrates the problem described above. Figure 1 shows an XML source for a poem *Song*. We want to create an XSLT stylesheet such that the poem appears in the browser as shown in Fig. 2.

A naive solution to the above problem is to create an XSLT stylesheet consisting of only one template rule whose pattern matches the XML root element `poem` and whose template contains the HTML document markup. Figure 3 shows this naive solution. However, this naive approach has a major drawback in terms of reusability. This solution is very specific for transforming the given XML document only and could not be used to transform other XML documents having similar structure. A particularly important kind of structurally similar document is an updated version of the original (after insertions or deletions). i.e. This naive solution would no longer be applicable if the XML document had another stanza added. This violates the initial purpose of developing XSLTGen, which is to generate a stylesheet based on the supplied source XML and desired output HTML documents, so that the resulting stylesheet can be reused with

[2] A DTD (Document Type Definition) is a grammar for describing the structure of a document. A DTD constrains the structure of an element by specifying a regular expression that its sub-element sequences have to conform to.

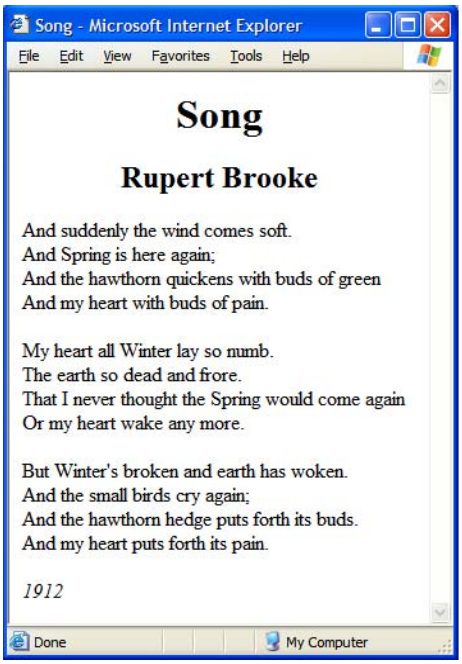
```
<poem>
  <author>Rupert Brooke</author>
  <date>1912</date>
  <title>Song</title>
  <stanza>
    <line>And suddenly the wind comes soft.</line>
    <line>And Spring is here again;</line>
    <line>And the hawthorn quickens with buds of green</line>
    <line>And my heart with buds of pain.</line>
  </stanza>
  <stanza>
    <line>My heart all Winter lay so numb.</line>
    <line>The earth so dead and frore.</line>
    <line>That I never thought the Spring would come again</line>
    <line>Or my heart wake any more.</line>
  </stanza>
  <stanza>
    <line>But Winter's broken and earth has woken.</line>
    <line>And the small birds cry again;</line>
    <line>And the hawthorn hedge puts forth its buds.</line>
    <line>And my heart puts forth its pain.</line>
  </stanza>
</poem>
```

Fig. 1. XML source for poem *Song*

other XML documents having a similar structure, or with the original document after updates to it have been applied.

In contrast, we are interested in generating a more generic stylesheet, which can then be reused to i) transform structurally similar XML documents and ii) transform the document even after it has been updated. In order to generate this solution, the first thing that we need to do is to discover the semantic mappings between the XML and HTML documents. A complete listing of the mappings found is presented in Table 1. The process of discovering these mappings will be explained in Sect. 4.

The next phase is to generate an XSLT stylesheet based on the mappings found. The stylesheet should start with the standard header. Following that, a template rule for each XML element specified in Table 1 is created. Finally, we finish off the stylesheet by closing the `<xsl:stylesheet>` element. This non-naive XSLT stylesheet is shown in Fig. 4. This stylesheet can then be reused to transform other XML documents having similar structure as the one in Fig. 1.




```
<html>
  <head><title>Song</title></head>
  <body>
    <div align="center"><h1>Song</h1></div>
    <div align="center">
      <h2>Rupert Brooke</h2>
    </div>
    <p>
      And suddenly the wind comes soft.<br>
      And Spring is here again;<br>
      And the hawthorn quickens with buds of
        green<br>
      And my heart with buds of pain.<br>
    </p>
    <p>
      My heart all Winter lay so numb.<br>
      The earth so dead and frore.<br>
      That I never thought the Spring would
        come again<br>
      Or my heart wake any more.<br>
    </p>
    <p>
      But Winter's broken and earth has
        woken.<br>
      And the small birds cry again;<br>
      And the hawthorn hedge puts forth its
        buds.<br>
      And my heart puts forth its pain.<br>
    </p>
    <p><i>1912</i></p>
  </body>
</html>
```

Fig. 2. Poem *Song* displayed in a browser and its HTML source

4 XSLTGen System

We now present an overview of the XSLTGen system. We then explain the details of each subsystem in the later subsections. We will use a running example based on a Soccer document.

4.1 System Architecture

The architecture of the XSLTGen system is illustrated in Fig. 5. Two input documents, a source XML document and a desired target HTML document, are given to XSLTGen in order to initiate the stylesheet generation process. The output of XSLTGen is an XSLT stylesheet consisting of rules for transforming the given XML document to the supplied HTML document. As shown in the figure, the system consists of six main components: *DOM Builder*, *Text Matching* subsystem, *Structure Matching* subsystem, *Sequence Checker*, *XSLT Stylesheet Constructor*, and *XSLT Stylesheet Refiner* subsystem.

DOM Builder. DOM is "a programming API for documents" [12]. It is based on an object structure that closely resembles the structure of the documents it models. In the DOM, documents have a logical structure which is very much like a tree. For each input document used in XSLTGen, the DOM builder constructs the DOM tree, which represents the structure of the specified document. We adopt the DOM package from W3C (`org.w3c.dom.*`) embedded with

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:template match="poem">
  <html>
    <head>
      <title>Song</title>
    </head>
    <body>
      <div align="center"><h1>Song</h1></div>
      <div align="center"><h2>Rupert Brooke</h2></div>
      <p>
        And suddenly the wind comes soft.<br/>
        And Spring is here again;<br/>
        And the hawthorn quickens with buds of green<br/>
        And my heart with buds of pain.<br/>
      </p>
      <p>
        My heart all Winter lay so numb.<br/>
        The earth so dead and frore.<br/>
        That I never thought the Spring would come again<br/>
        Or my heart wake any more.<br/>
      </p>
      <p>
        But Winter's broken and earth has woken.<br/>
        And the small birds cry again;<br/>
        And the hawthorn hedge puts forth its buds.<br/>
        And my heart puts forth its pain.<br/>
      </p>
      <p><i>1912</i></p>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Fig. 3. Naive solution of XSLT stylesheet for poem *Song*

Table 1. Mappings for poem *Song*

XML	HTML
poem	body
author	div[2]
date	p[4]
title	div[1]
stanza[i]	p[i], for $1 \leq i \leq 3$
line[i]	text()[i] ++ br, for $1 \leq i \leq 4$

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:template match="poem">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <xsl:apply-templates select="title"/>
      <xsl:apply-templates select="author"/>
      <xsl:apply-templates select="stanza"/>
      <xsl:apply-templates select="date"/>
    </body>
  </html>
</xsl:template>
<xsl:template match="title">
  <div align="center"><h1><xsl:value-of select="."/></h1></div>
</xsl:template>
<xsl:template match="author">
  <div align="center"><h2><xsl:value-of select="."/></h2></div>
</xsl:template>
<xsl:template match="date">
  <p><i><xsl:value-of select="."/></i></p>
</xsl:template>
<xsl:template match="stanza">
  <p><xsl:apply-templates select="line"/></p>
</xsl:template>
<xsl:template match="line">
  <xsl:value-of select="."/><br/>
</xsl:template>
</xsl:stylesheet>
```

Fig. 4. Non-naive solution of XSLT stylesheet for poem *Song*

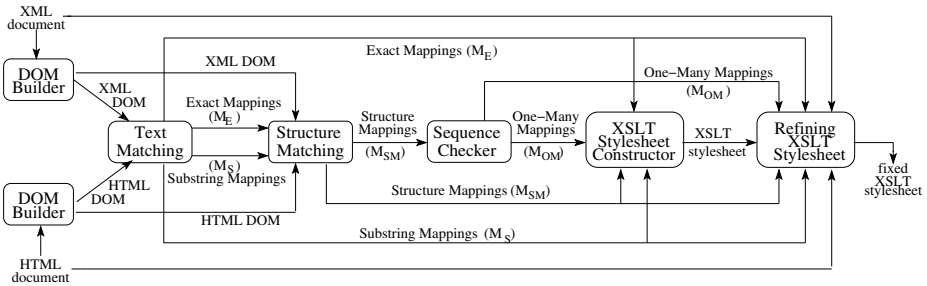


Fig. 5. Architecture of the XSLTGen system

in Java to accommodate this task. The input document given to DOM builder has to be strictly well-formed. Figure 6 shows an example of fragments of XML and HTML DOMs for our Soccer example.

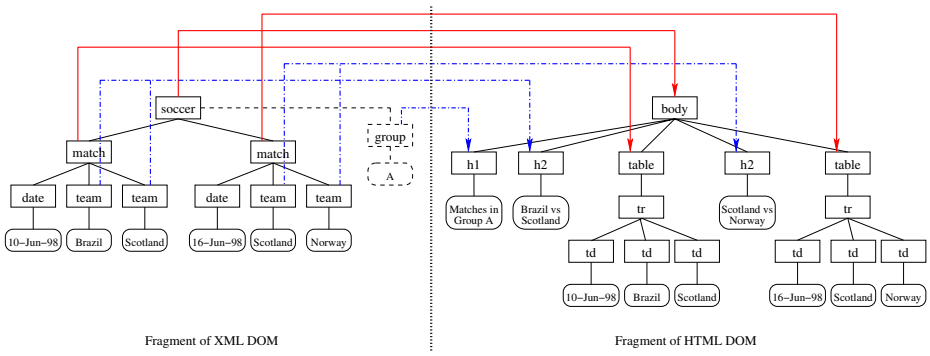


Fig. 6. Soccer example

Text Matching subsystem. The XML and HTML DOMs built by the DOM builder are input to the text matching subsystem. This subsystem discovers associations between nodes in the XML DOM and those in the HTML DOM, i.e. which nodes in the HTML DOM correspond to which nodes in the XML DOM and what transformations need to be applied to those XML nodes. In this subsystem, the mappings found are instances of text-based element-level matching, which can either be exact or substring mappings. Since the HTML document is generated based on the content of the XML document, it is possible to find elements within the HTML DOM that have the same text data or part of text data as the elements in the XML DOM.

Structure Matching subsystem. The structure matching subsystem also takes the XML and HTML DOM trees produced by the DOM builder as its input. Besides these DOM trees, this subsystem takes the list of exact and substring mappings found by the text matching subsystem into account, to discover structure-level associations between XML DOM nodes and HTML DOM nodes. The term structure-level matching refers to matching combinations of elements that appear together in a structure. Loosely speaking, two nodes, an XML and an HTML node, structurally match if their children also match.

Sequence Checker. This subsystem is responsible for verifying each structure mapping found and possibly discovering 1-m mappings (a mapping between an XML element and a concatenation of two or more HTML elements) depending on the result of the verification. In the verification process, the sequence checker checks whether the *sequence* of the XML component conforms to the *sequence* of the HTML component, by comparing an inferred DTD of the XML component and an inferred DTD of the HTML component. The *XTRACT* system [14] is used to infer the DTD of an element. If the verification process fails,

the sequence checker generates a new 1-m mapping based on the two extracted DTDs.

XSLT Stylesheet Constructor. This subsystem generates a template rule for each mapping discovered by the text matching, structure matching and sequence checker subsystems. Consecutively, the template rules are put together to construct an XSLT stylesheet.

XSLT Stylesheet Refiner. This subsystem is responsible for repairing the XSLT stylesheet generated by the XSLT stylesheet constructor, in circumstances where there are differences between the original HTML document and the one produced by applying the generated stylesheet back to the given XML document.

4.2 Text Matching

The quality of the XSLT stylesheet generated by the XSLT stylesheet constructor is dependent on the set of mappings input to it. The mappings found in text matching serve as the basis for discovering structure mappings, which in turn serve as the starting point for finding 1-m mappings. Therefore, it is crucial that the text mappings found be complete and accurate. The goal of the text matching subsystem is to achieve this objective by discovering a set of *exact mappings* M_e and *substring mappings* M_s (recall these were defined in section 2.2). As the content of the HTML document is created based on the content of the XML document, it is important to find both exact and substring mappings between the two documents, since there must be HTML elements that have the same string or substring as the XML elements.

The starting point to find a mapping is to compare nodes that have a *value* attribute, i.e. TEXT_NODEs and ATTRIBUTE_NODEs. In this paper, the nodes that we compare are an XML TEXT_NODE with an HTML TEXT_NODE and an XML ATTRIBUTE_NODE with an HTML TEXT_NODE. We do not consider HTML ATTRIBUTE_NODEs, since the attribute value of those nodes is usually specific to the display of the HTML document in the Web browsers and is not generated from the text within the XML document. Example 1 shows examples of both exact and substring mappings.

Example 1. In the Soccer example of Fig. 6,

1. An exact mapping occurs between XML TEXT_NODE "`10-Jun-98`" and HTML TEXT_NODE "`10-Jun-98`", because $X.value \equiv H.value$.
2. A substring mapping occurs between XML TEXT_NODE "`A`" and HTML TEXT_NODE "`Matches in Group A`", since $X.value$ is a substring of $H.value$.

As mentioned earlier, the text matching procedure takes two inputs: an XML DOM x and an HTML DOM h. It discovers as many text mappings as possible between the nodes in x and h. The text matching procedure is called twice, once to first discover all EXACT mappings between the XML nodes in x and the HTML nodes in h and again to discover all SUBSTRING mappings. The output

of our text matching algorithm is two lists of mappings: EXACT mappings (M_E) and SUBSTRING mappings (M_S).

Our text matching is implemented using a top-down approach that visits each node in the XML DOM in pre-order and uses the same traversal to explore the HTML DOM, to find a matching node. Note that in order to create an XSLT template rule, the XML node should be an ELEMENT_NODE (not a TEXT_NODE). The discovered exact or substring mapping between two TEXT_NODES cannot represent any transformation in the XSLT template rule. Therefore, we need to determine for each mapping found, the *precise node* of its XML and HTML component that best describes the transformation. This approach allows us to discover more precise mappings between the XML and HTML documents.

Moreover, we set a constraint that every HTML node that has been matched to an XML node during the exact matching process, cannot be considered as a matching candidate in the substring matching process. In this way, we reduce the number of possible matching elements for substring matching.

Referring back to the Soccer example (Fig. 6), some of the text mappings discovered are: (`<team>Brazil</team>`, `<td>Brazil</td>`) (exact mapping), (`<team>Brazil</team>`, `<h2>Brazil vs Scotland</h2>`) (substring mapping); and
(`group="A"`, `<h1>Matches in Group A</h1>`) (substring mapping).

4.3 Structure Matching

Structure matching discovers all structure mappings between the elements in the XML and HTML DOMs. Finding these mappings is more complicated than finding text matches. We adopt two constraints used in the GLUE system [10] as a guide to determine whether two nodes should be structurally matched:

1. *Neighbourhood Constraint*: "two nodes match if nodes in their neighbourhood also match", where the neighbourhood is defined to be the children.
2. *Union Constraint*: "if every child of node A matches node B, then node A also matches node B" (this constraint is derived from the taxonomy context. It relies on the property that element/concept A is the union of all its children).

Note that there could be a range of possible matching cases, depending on the completeness and precision of the match. In the ideal case, all components of the structures in the two nodes fully match. Alternatively, only some of the components are matched (a partial structural match). Examples of the two cases are shown in Table 2. In the case of partial structure matching between XML node X and HTML node H, there are some children of X that do not match with any children of H; and/or vice versa. However, these XML nodes must be *extra node*s, i.e. they do not have any match in the entire HTML document. Similarly with those children of H that do not match with any children of X, they should not have any match in the entire XML document either. In addition to this, partial structure matching is valid if at least one of the children of X and H matches. If all children of X are extra nodes, and/or all children of H are

extra nodes, then this is not a partial structure matching. We allow XSLTGen system to detect partial structure matching because *extra node*s do not give additional information about the mapping and we want to ignore them during the structure matching process, i.e. treat them as if they are not present in the document. Having or not having *extra node*s in the documents should not affect the mappings found.

Table 2. Example of full vs partial structural match

XML elements	HTML elements	
<book> <author>Michael Kay</author> <title>XSLT</title>	<tr> <td>Michael Kay</td> <td>XSLT</td> <td>Wrox</td> <td>34.99</td> </tr>	full structural match of book and tr
<price>34.99</price> <publisher>Wrox</publisher> </book>	<tr> <td>reference</td> <td>Michael Kay</td> <td>XSLT</td> <td>34.99</td> </tr>	partial structural match of book and tr

In order to be able to discover full and partial structure matching, the above constraints need to be modified to construct the definition of structure matching which accommodates both full and partial structure matching.

Definition 1. *A* **structure mapping** $m : (X, H)$ *exists in the following circumstances:*

1. Neighbourhood Constraint*: "X structurally matches H if H is not an* extra HTML node *and every* non-extra *child of H either text matches or structurally matches a* non-extra *descendant of X"; or,*
2. Union Constraint*: "X structurally matches H if every* non-extra *child of H either text matches or structurally matches X".*

As stated in the above definition, we need to examine the children of the two nodes being compared in order to determine if a structure matching exists. Therefore, structure matching is implemented using a bottom-up approach that visits each node in the HTML DOM in post-order and searches for a matching node in the entire XML DOM. This bottom-up approach also enables the system to apply the union constraint to a lower level structure mapping by invalidating and updating it with an upper level structure mapping, if the latter is found to be a better match than the former. If the list of substring mappings M_{SM} is still empty after the structure matching process finishes, we add a mapping from the

XML root element to the HTML body element, if it exists, or to the HTML root element, otherwise. The structure matching algorithm is presented in Fig. 7. It takes as input the exact mappings (M_E, the substring mappings (M_S), the XML DOM x and the HTML DOM h. It then returns a list of structure mappings M_{SM}.

In the Soccer example (Fig. 6), some discovered structure mappings are:

1. (`match`, `tr`) (neighbourhood constraint): since every child of `tr` text matches one of the children of `match`, i.e. (`date`, `td[1]`), (`team[1]`, `td[2]`), and (`team[2]`, `td[3]`)
2. mapping (1) will then be invalidated and updated by (`match`, `table`) (union constraint) since the only child of `table`, i.e. `tr`, structurally matches `match`
3. (`soccer`, `body`) (neighbourhood constraint): since every child of `body` either text matches or structurally matches one of the descendant of `soccer`.

4.4 Sequence Checking

Up to this point, the mappings generated by the text matching and structure matching subsystems have been limited to 1-1 mappings. In cases where the XML and HTML documents have more complex structure, these mappings may not be accurate and this can affect the quality of the XSLT rules generated from these mappings. Consider the following example:

In Fig. 6, we can see that the sequence of the children of XML node `soccer` is made up of nodes with the same name, `match`; whereas the sequence of the children of the matching HTML node `body` follows a specific pattern: it starts with `h1` and is followed repetitively by `h2` and `table`. Using only the discovered 1-1 mappings, it is not possible to create an XSLT rule for `soccer` that resembles this pattern, since the XML node `match` maps only to the HTML node `table` according to structure matching. In other words, there will be no template that will generate the HTML node `h2`.

Focusing on the structure mapping (`match`, `table`) and the substring mappings {(`team[1]`, `h2`), (`team[2]`, `h2`)}, we can see in the DOM trees that the children of `match`, i.e. `team[1]` and `team[2]`, are not mapped to the descendant of `table`. Instead, they map to the sibling of `table`, i.e. `h2`. Normally, we expect that the descendant of `match` maps only to the descendant of `table`, so that the notion of 1-1 mapping is kept. In this case, there is an intuition that `match` should not only map to `table`, but also to `h2`. In fact, `match` should map to the concatenation of `h2` and `table`, so that the sequence of the children of `body` is preserved when generating the XSLT rule. This is termed as a 1-m mapping, where an XML node maps to the concatenation of two or more HTML nodes.

The 1-m mapping (`match`, `h2 ++ table`) can be found by examining the sub-element sequence of `soccer` and the sub-element sequence of `body` described above. Note that the sub-element sequence of a node can be represented us-

```
procedure StructureMatching(M_E, M_S, x, h)
begin
1.   DoStructMatch(M_E, M_S, x, h)
2.   if (M_SM is empty)
3.        if (h has a descendant whose name is body)
4.             add (x, body) to M_SM
5.        else
6.             add (x, h) to M_SM
end

procedure DoStructMatch(M_E, M_S, x, h)
begin
1.   for each child c of h
2.        DoStructMatch(M_E, M_S, x, c)
3.   if h.type = element_node
4.        SearchStructMatch(M_E, M_S, x, h)
end

procedure SearchStructMatch(M_E, M_S, x, h)
begin
1.   E_x := {c_x : c_x ∈ children of x, c_x.type = element_node}
2.   for each node c_x in E_x
3.        SearchStructMatch(M_E, M_S, c_x, h)
4.   if (CheckMatch(M_E, M_S, x, h) = true)
5.        m := (x, h)
6.        if (MappingExist(m) = not_exist)
7.             add m to M_SM
8.        else if (MappingExist(m) = modify)
9.             if (x ≠ x_root)    /* x_root is the root of the XML document */
10.                 replace all a.h in {a : a ∈ M_SM, a.x = m.x, a.h.name ≠ body, a.h is a descendant
                    of m.h} with h
end

procedure CheckMatch(M_E, M_S, x, h)
begin
1.   if (h is an extra HTML node)
2.        return false
3.   for each child c_h of h
4.        if (c_h is not an extra HTML node)
5.             let x' be an XML node, such that ((x', c_h) ∈ M_E or (x', c_h) ∈ M_S or (x', c_h) ∈ M_SM)
6.             if (x' ≠ null)
7.                  if (x' is not a descendant of x)
8.                       return false
9.             else
10.                 return false
11.  return true
end

procedure MappingExist(m)
begin
1.   if (m ∈ M_SM or {a : a ∈ M_SM, a.x is a descendant of m.x, a.h = m.h} is not empty)
2.        return exist
3.   else if ({a : a ∈ M_SM, a.x = m.x, a.h.name ≠ body, a.h is a descendant of m.h} is not empty)
4.        return modify
5.   return not_exist
end
```

Fig. 7. The structure matching algorithm

ing a regular expression[3]. In this case, the regular expression representing the sub-element sequence of `soccer` is $match*$, whereas the one representing the sub-element sequence of `body` is $h_1(h_2, table)*$. We then check whether the elements in the first sequence conform to the elements in the second sequence, as follows: According to the substring mapping (`group`, `h1`), element h_1 conforms to an attribute of `soccer` and thus, we ignore it and remove h_1 from the second sequence. Comparing $match*$ with $(h_2, table)*$, we can see that element m should conform to elements $(h_2, table)$ since the sequence $match*$ corresponds directly to the sequence $(h_2, table)*$, i.e. they both are in repetitive pattern, denoted by $*$. However, element $match$ conforms only to element $table$, as indicated by the structure matching (`match`, `table`). The verification therefore fails, which indicates that the structure matching (`match`, `table`) is not accurate. Consequently, based on the sequences $match*$ and $(h_2, table)*$, we deduce the more accurate 1-m mapping: (`match`, `h2 ++ table`).

The main objective of the sequence checking subsystem is to discover 1-m mappings using the technique of comparing two sequences described above. The pseudo-code of the sequence checking procedure is presented in Fig. 8. The task begins with processing the structure mappings in M_{SM} which are provided as input to the procedure. Given a structure mapping m, the issue is to verify that the $seq(m.x)$ conforms to $seq(m.h)$. If $seq(m.x)$ does not conform to $seq(m.h)$, the verification process fails and this situation suggests that there are 1-m mappings that can instead be generated based on these sequences. These discovered mappings 1-m mappings (M_{OM}) are what is finally returned by the procedure.

As we mentioned above, a sequence can be represented using a regular expression. To obtain this regular expression, we adapt the technique for inferring a DTD of an element used by XTRACT [14].

4.5 XSLT Stylesheet Generation

This subsystem is responsible for constructing a template rule for each of the mappings previously discovered (the exact mappings M_E, structure mappings M_{SM}, and 1-m mappings M_{OM}) and then putting them all together to compose an XSLT stylesheet. We do not create template rules for the substring mappings in M_S, because in substring mappings, it is possible to have an HTML node whose text *value* is a concatenation of text *value*s of two or more XML nodes. This makes it impossible to create template rules for those XML nodes. Moreover, as the term *substring* implies, there can be some *extra string*s presented in the HTML text *value*. Considering these situations, we implement a procedure that generates a *template* for each distinct HTML node in M_S.

The XSLT stylesheet generation process begins by generating the list of substring rules. We then construct a stylesheet by creating the `<xsl:stylesheet>` root element and subsequently filling it with template rules for the 1-m mappings

[3] A regular expression is a combination of symbols representing each sub-element and metacharacters: $|, *, +, ?, (,)$. $|$ denotes OR, $*$ denotes zero or more, $+$ denotes one or more, ? denotes zero or one, and ( ) are used for grouping

```
procedure CheckSequence(M_SM)
begin
1.   check_root = FALSE
2.   for each mapping m in M_SM
3.       if (m.x = x_root and check_root = FALSE)
4.           x_seq = Xtract(m.x)
5.           h_seq = Xtract(m.h)
6.       else if (m.x ≠ x_root)
7.           x_dtd = Xtract(parent of m.x)
8.           x_seq = Trim_{m.x}(x_dtd)
9.           h_dtd = Xtract(parent of m.h)
10.          h_seq = Trim_{m.h}(h_dtd)
11.          if (parent of m.x = x_root)
12.              check_root = TRUE
13.      else
14.          continue
15.      if (x_seq = (s_x)* and h_seq = (x_h)*)
16.          x_seq = s_x
17.          h_seq = s_h
18.      X_SEQS = ∅
19.      if (x_seq = s_1|s_2|...|s_n)
20.          add each s_i to X_SEQS
21.      else
22.          add x_seq to X_SEQS
23.      h_seq = remove symbols representing special HTML elements in h_seq
24.      for each sequence x_i in X_SEQS
25.          if (number of symbols in x_i = 1 and number of symbols in h_seq > 1)
26.              x = XML node represented by symbol x_i
27.              for each symbol h_j in h_seq
28.                  h = h ++ HTML node represented by symbol h_j
29.              m = (x, h)
30.              add m to M_OM
end
```

Fig. 8. The sequence checking algorithm

in M_{OM}, the structure mappings in M_{SM} and the exact mappings in M_E. The template rules for the 1-m mappings have to be constructed first, since within that process, they may invalidate several mappings in M_{SM} and M_E (e.g. rule A and B of Table 6 and thus, the template rules for those omitted mappings do not get used. In each mapping list (M_{OM}, M_{SM}, and M_E), the template rule is constructed for each *distinct* mapping to avoid having conflicting template rules.

Even though the mappings between the nodes in the XML document and the nodes in the HTML documents have been discovered previously, creating a template rule for a mapping m is not an easy task. This is due to the fact that we have to appropriately locate the suitable XSLT instructions, taking into account the structures or the subtrees of the XML and HTML components of m. Therefore, in the next four subsections, we describe the detail of substring rule generation, followed by explanation on how to construct template rules for exact mapping, structure mapping, and 1-m mapping.

Substring Rule Generation. The substring rule generator creates a template from an XML node or a set of XML nodes to each distinct HTML node presented in the substring mapping list M_S. The result of this subsystem is a list of substring rules SUB_RULES, where each element is a tuple (*html_node*, *rule*).

The set of XML nodes that map to the same HTML node could be large, since within that set, it is possible to have two XML nodes with different names but the same string, or two nodes with the same name and string but different locations in the XML DOM. As an example, consider the substring mappings in Fig. 9. In that figure, both XML nodes `artist` (node **I**) and `owner` (node **B**) have the same text value "`Kylie`" and map to the same HTML node `<td>Kylie's by Kylie (2002)</td>` (node **R**). On the other hand, two XML nodes named `date` (nodes **E** and **H**) have the same text value "`2002`" but can either map to HTML node `<td>Aquarium by Aqua (2002)</td>` (node **P**) or `<td>Kylie's by Kylie (2002)</td>` (node **R**).

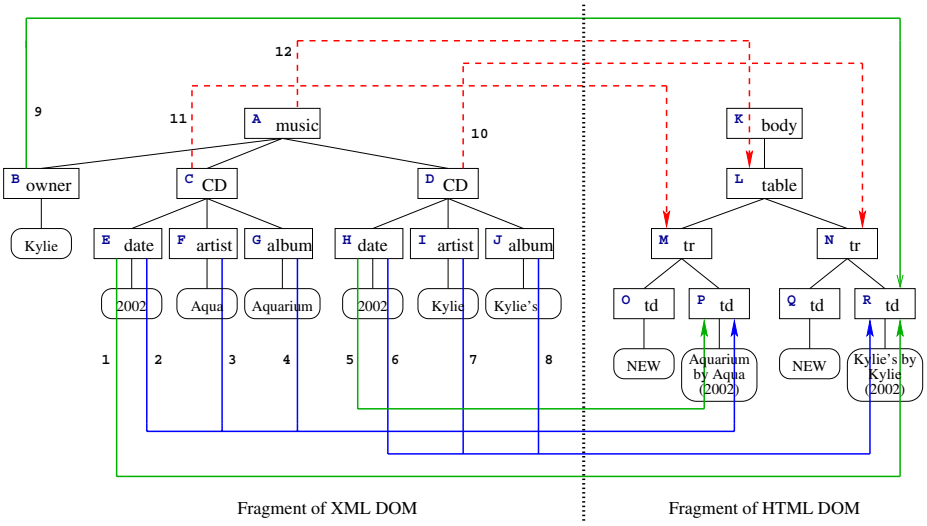


Fig. 9. Music example

This situation may cause some ambiguities in the substring rules generated. Consider the mappings where the HTML component is node **R**, i.e. mappings **1**, **6**, **7**, **8**, **9**. There can be four different combinations of the XML components in these mappings that result in two similar substring rules representing the HTML component. Combinations of nodes **E**, **I**, and **J**; or nodes **H**, **I**, and **J** produce the substring rule:

```
<tr><xsl:value-of select="album"/> by <xsl:value-of select="artist"/>
    (<xsl:value-of select="date"/>)</tr>.
```

Alternatively, combinations of nodes **B**, **E**, and **J**; or nodes **B**, **H**, and **J** yield the substring rule:

```
<tr><xsl:value-of select="album"/> by <xsl:value-of select="owner"/>
    (<xsl:value-of select="date"/>)</tr>.
```

Clearly, the problem that we face here is choosing the combination of XML nodes that best describe an HTML component. Based on the XML and HTML DOM in Fig. 9, we can see that the combination of XML nodes **H**, **I**, and **J** is the one that best represents the HTML node **R**. This is because the region of these XML nodes is related to the region of HTML node **R**, i.e. the parent of these XML nodes structurally maps to the parent of HTML node **R**, as indicated by the structure mapping **10**. In addition, the mappings **1**, **5**, and **9** should not be considered in the substring rule generation, since the regions between the XML and HTML nodes are not related and hence, they are not accurate.

In the following subsection, we have devised a heuristic strategy for selecting a subset of such HTML component mappings that yields an unambiguous substring rule. We then present a brief description of the algorithm for generating the substring rule itself.

Selecting a Subset of the Mappings to the Same HTML Component The main motivation behind choosing a subset of the mappings that go the same HTML component is that we want to construct a unique and unambiguous substring rule between a set of XML nodes and an HTML node. This purpose is served only if the region of the XML nodes and the region of the HTML node is related. For example, in order to generate a substring rule for HTML node **P** in the Music example (Fig. 9), there are four substring mappings that can be used: mappings **2**, **3**, **4**, and **5**, since the HTML component of these mappings is the HTML node **P**. However, the subset of these mappings that we select are the mappings **2**, **3**, and **4**, since the XML components of these mappings are located in the region that is related to the region of the HTML node **P**. The two regions that we discuss are the subtree of XML node **C** and the subtree of HTML node **M**, which are structurally mapped as indicated by the structure mapping **11**. Mapping **5** is not selected since the XML node **F** is not located in the subtree of XML node **C**.

In this subsection, we describe how we choose a subset of substring mappings that best describes an HTML node based on the related regions. Note that the set of mappings that we are focusing on, is the one that has the same HTML component for every mapping in that set. This set of mappings is in fact a subset of M_S. Splitting M_S into sets of mappings with the same HTML component is not a complicated task.

Intuitively, given a set of mappings M_H that all map to the same HTML node, a subset G of M_H is a good subset if the combination of the XML components in G produces a single substring rule that represents the HTML component.

Therefore, given a set of substring mappings M_H, which has the same HTML component h for every mapping in it, we first find a mapping s in M_{SM} or in M_{OM}, such that $s.h$, is the closest ancestor of h. Then, a subset G of M_H is a "good" subset for generating a unique substring rule, if for every mapping m in G, $m.x$ is a descendant of $s.x$.

Referring back to the Music example describe in Sect. 4.5, to facilitate the substring rule generation of HTML node **R**, we need to examine the set M_H of substring mappings **1**, **6**, **7**, **8**, and **9**, since these mappings lead to the HTML

node **R**. As stated above, the first thing that we need to do is find a structure mapping or a 1-m mapping s, whose HTML component is the closest ancestor of node **R**. In this case, the mapping s is the structure mapping between node **D** and node **N** (mapping **10**). We then select the mappings in M_H whose XML component is a descendant of node **D** and we obtain a good subset G consisting of mappings **6**, **7**, and **8**.

Once we have a "good" subset G, a substring rule R is generated by invoking the algorithm described in the next subsection, and is then added to the list of substring rules SUB_RULES.

Algorithm for Generating a Substring Rule. In this subsection, we show how the substring rule for a set of mappings G can be derived. Recall that all mappings in G have the same HTML component, h. Concisely, the main idea of the substring rule generation is to construct the string of h by combining the strings of the XML components of the mappings in G. Since the mappings in G are found within the text matching subsystem, h only has one string, denoted by h_string, which is the *value* of the TEXT_NODE located at the leaf of the subtree of h. For example, the h_string of the HTML node `td` (node **R**) in the Music example (Fig. 9) is "`Kylie's by Kylie (2002)`". The same condition applies to the XML component of each mapping in G. If the XML component is an ELEMENT_NODE, the XML node also has only one string x_string, which is the *value* of the TEXT_NODE located at the leaf of the subtree of the XML node. For instance, the x_string of the XML node `date` (node **E**) in Fig. 9 is "`2002`". In the case where the XML component is an ATTRIBUTE_NODE, the string x_string is simply the *value* of the XML node.

As the term *substring* implies, there must be parts of h_string that do not have any matching x_string in G, i.e. these are *extra string*s. For each substring of h_string that exactly matches an x_string, an XSLT `<xsl:value-of>` statement is generated to extract the x_string from the corresponding XML component. Thus, we can conclude that the generated substring rule is a sequence of *extra string*s and `<xsl:value-of>` statements that together represents h_string.

The substring rule s_rule for the mappings in G is constructed recursively until h_string is empty. At each step, we search for a mapping m' whose x_string of $m'.x$ matches the longest prefix of h_string and then update the substring rule s_rule using the rules specified in Table 3. When h_string is empty, the construction of s_rule of G is finished and a tuple (h, s_rule) is returned.

We believe that this algorithm is a reasonable method for constructing substring rules, since it is capable of capturing *extra string*s and matching x_strings accurately. The following example best illustrates the key steps of our algorithm.

Example 2. Consider the set G of substring mappings **6**, **7**, and **8** shown in the Music example of Fig. 9. In this case, the h_string of G is "`Kylie's by Kylie (2002)`", while the set of x_strings is {"`2002`", "`Kylie`", `Kylie's`}. Below, we list how Rules (A) and (B) described in Table 3 are recursively applied to derive the substring rule s_rule.

Table 3. Rules for generating a substring rule

Rule	Condition	Update
A	m' is not found	Extract the first character of *h_string* and add it to *s_rule*, since it is suspected that the prefix of the current *h_string* is part of the *extra strings*
B	m' is found, $m'.x$ is an ELEMENT_NODE	Add `<xsl:value-of select="`$m'.x.name$`"/>` to *s_rule* and delete the prefix of *h_string* matching *x_string*
C	m' is found, $m'.x$ is an ATTRIBUTE_NODE	Add `<xsl:value-of select="`$owner(m'.x).name$`/@`$m'.x.name$`"/>` and delete the prefix of *h_string* matching *x_string*

1. Mapping **8** is chosen since its *x_string* "`Kylie's`" is the longest string that matches the prefix of *h_string* "`Kylie's by Kylie (2002)`". According to **Rule (B)**,
   s_rule = `<xsl:value-of select="album"/>`
   h_string = "` by Kylie (2002)`".
2. Apply **Rule (A)** recursively for characters in substring "` by `" since it is an *extra string*. At the final execution of **Rule (A)**,
   s_rule = `<xsl:value-of select="album"/> by`
   h_string = "`Kylie (2002)`".
3. Mapping **7** is chosen and we apply **Rule (B)**.
4. Apply **Rule (A)** recursively for characters in *extra string* "` (`".
5. Mapping **6** is chosen and we apply **Rule (B)**.
6. Apply **Rule (A)** for character "`)`". *h_string* is now empty and *s_rule* is
   `<xsl:value-of select="album"/> by <xsl:value-of select="artist"/> (<xsl:value-of select="date"/>)`

Constructing a Template Rule for an Exact Mapping in M_E. As described in Sect. 2.1, each template rule begins with an XSLT `<xsl:template>` element and ends by closing that element. For a mapping m, the pattern of the corresponding template rule is the name of $m.x$. The difficult task lies in determining the appropriate template, i.e. which XSLT instructions to be used and where they should be placed.

We now describe how we construct a template rule for an exact mapping m. Compared to the procedures for the other two mappings (structure mapping and 1-m mapping), this procedure is the simplest and the most straightforward, since there is only one XSLT instruction used in the template: `<xsl:value-of>`. In this procedure, we only construct a template rule when m is a mapping between an XML ELEMENT_NODE to an HTML node or a concatenation of HTML nodes. The reason that we ignore mappings involving XML ATTRIBUTE_NODEs is that the template for this mapping will be generated directly within the construction of the template rule for structure mapping and 1-m mapping.

Since there could be mappings from an XML node to a concatenation of HTML nodes in text matching, we need to create a template for each HTML

node h_i in $m.h$. Given an exact mapping m, the rules for creating a template representing transformation between $m.x$ and h_i are listed in Table 4.

Table 4. Rules for creating the template for exact mappings

Rule	Condition	Template
A	h_i is a *special* HTML element	<$h_i.name$/>
B	h_i is not a *special* HTML element, $m.x$ is an ELEMENT_NODE	<$h_i.name$><xsl:value-of select="."/> </$h_i.name$>
C	h_i is not a *special* HTML element, $m.x$ is an ATTRIBUTE_NODE	<$h_i.name$> <xsl:value-of select="@$m.x.name$."/> </$h_i.name$>

Example 3. Consider the exact mapping (`line`, `text() ++ br`). The steps performed to construct the template rule are:

1. Create a template rule: `<xsl:template match="line">`
2. Create a template for each HTML node in (`text() ++ br`): for HTML node `text()`, apply **Rule (B)** since the XML node `line` is an ELEMENT_NODE. The template body is: `<xsl:value-of select="."/>`
   Note that there are no opening and closing tags, since `text()` is a TEXT_NODE.
3. Apply **Rule (A)**, since `br` is a *special* HTML element: `<br/>`
4. Close the template rule: `</xsl:template>`

Constructing a Template Rule for a Structure Mapping in M_{SM}. We next explain how the template rule for a structure mapping m is constructed. Recall from the structure matching subsystem that one of the mappings in M_{SM} must be the mapping that has the root of the XML document as its XML component. Let r denote this special mapping. For the mapping r, the template begins with copying the root of the HTML document and its subtree, excluding the HTML component $r.h$ and its subtree.

The next step in constructing the template for mapping r follows the steps performed for the other mappings in M_{SM}. For any mapping m in M_{SM}, the opening tag for $m.h$ is created. The process continues with creating a template for each child c_i of $m.h$ and finishes by closing the $m.h$ tag.

For each HTML child c_i, we need to determine whether an XSLT instruction is needed; and if so, which XSLT instruction should be used. This task depends fully on the similarities and differences between the structure of the XML component $m.x$ and the structure of the HTML node h_i. Given an XML component $m.x$ and a child c_i of the HTML component $m.h$, the rules for creating a template that represents the transformation from $m.x$ to c_i are listed in Table 5.

Table 5. Rules for creating the template for structure mappings

Rule	Condition	Template
A	c_i is an *extra node* but not a *special* HTML element	Node c_i complete with its subtree
B	$\exists e \in M_E$, $e.x$ is a descendant of $m.x$, $e.h = c_i$	`<xsl:apply-templates select="`{XPath describing $e.x$ in the context of $m.x$}`"/>`
C	$\exists e \in M_E$, $owner(e.x) = m.x$, $e.x$ is an ATTRIBUTE_NODE, $e.h = c_i$	Apply the algorithm for constructing a template rule for an exact mapping to e
D	$\exists e \in M_E$, $owner(e.x)$ is a descendant of $m.x$, $e.x$ is an ATTRIBUTE_NODE, $e.h = c_i$	`<`$c_i.name$`><xsl:value-of select="`{XPath describing $owner(e.x)$ in the context of $m.x$}`/@`$e.x.name$`"/></`$c_i.name$`>`
E	$\exists s \in M_{SM}$, $s.x$ is a descendant of $m.x$, $s.h = c_i$	`<xsl:apply-templates select="`{XPath describing $s.x$ in the context of $m.x$}`"/>`
F	$\exists sr : (c_i, sr.rule) \in$ SUB_RULES	`<`$c_i.name$`>` $sr.rule$ `</`$c_i.name$`>`
G	Otherwise	`<`$c_i.name$`>` {Create template for $(m.x, c_i)$, i.e. apply **Rules (A) - (G)** to each child of c_i} `</`$c_i.name$`>`

Example 4. Consider the structure mapping (`match`, `table`) and the exact mappings (`date`, `td[1]`), (`team[1]`, `td[2]`), (`team[2]`, `td[3]`) discovered in the Soccer example (Fig. 6) . The steps required to generate the template rule for this structure mapping are:

1. Create a template rule: `<xsl:template match="match">`
2. Start the template body by generating an opening tag for HTML node `table`, since `match` is not the root of the XML document: `<table>`
3. Apply **Rule (G)** since `tr` has no matching XML node in any mapping lists:
   (a) Generate an opening tag for HTML node `tr`: `<tr>`
   (b) For the first child of `tr`, i.e. `td[1]`, apply **Rule (B)** because (`date`, `td[1]`) $\in M_E$ and `date` is a descendant of the XML node `match`:
   `<xsl:apply-templates select="./date"/>`
   (c) Apply **Rule (B)** to the second and third child of `tr`, i.e. `td[2]` and `td[3]`, for the same reason as the first child `td[1]`, and we obtain:
   `<xsl:apply-templates select="./team[1]"/>`
   `<xsl:apply-templates select="./team[2]"/>`
   (d) Generate the closing tag for `tr`: `</tr>`
4. Generate the closing tag for HTML node `table`: `</table>`
5. Close the template rule: `</xsl:template>`

Constructing a Template Rule for a One-Many Mapping in M_{OM}. This subsection describes the process for constructing a template rule for a 1-m mapping m in M_{OM}. Being a 1-m mapping, the HTML component $m.h$ must be a concatenation of several HTML nodes. Thus, each HTML node h_i in $m.h$ is processed sequentially to find out the sequence of XSLT instructions that fill

up the template rule. Given a 1-m mapping m, the rule for creating a template representing the transformation from $m.x$ to each h_i is presented in Table 6.

Table 6. Rules for creating the template for 1-m mappings

Rule	Condition	Template
A	$(m.x, h_i) \in M_{SM}$	Apply the algorithm for constructing a template rule for a structure mapping to $(m.x, h_i)$. Then, delete mappings $(m.x, h_i) \in M_{SM}$
B	$(m.x, h_i) \in M_E$	Apply the algorithm for constructing a template rule for an exact mapping to $(m.x, h_i)$. Then, delete mappings $(m.x, h_i) \in M_E$
C	$\exists e \in M_E$, $e.x$ is a descendant of $m.x$, $e.h = h_i$	<$h_i.name$><xsl:apply-templates select="{XPath describing $e.x$ in the context of $m.x$}"/></$h_i.name$>
D	$\exists e \in M_E$, $owner(e.x) = m.x$, $e.x$ is an ATTRIBUTE_NODE, $e.h = h_i$	Apply the algorithm for constructing a template rule for an exact mapping to e
E	$\exists e \in M_E$, $owner(e.x)$ is a descendant of $m.x$, $e.x$ is an ATTRIBUTE_NODE, $e.h = h_i$	<$h_i.name$><xsl:value-of select="{XPath describing $owner(e.x)$ in the context of $m.x$}/@$e.x.name$"/></$h_i.name$>
F	$\exists sr : (h_i, sr.rule) \in$ SUB_RULES	<$h_i.name$>$sr.rule$</$h_i.name$>
G	h_i is an *extra node* but not a *special* HTML element	Node h_i complete with its subtree

Example 5. Consider 1-m mapping (`match`, `h2 ++ table`) discovered in Sect. 4.4, and structure mapping (`match`, `table`) found in Sect. 4.3 for the Soccer example (Fig. 6). Suppose we have generated substring rule (`h2`, `<xsl:value-of select= "team[1]"/> vs <xsl:value-of select="team[2]"/>`). The steps involved in generating the template rule for the 1-m mapping is:

1. Create a template rule: `<xsl:template match="match">`
2. Apply **Rule (F)** since SUB_RULES contains a substring rule for `h2`:
```
<h2><xsl:value-of select="team[1]"/> vs
    <xsl:value-of select="team[2]"/></h2>
```
3. For HTML node `table`, apply **Rule (A)** since (`match`, `table`) $\in M_{SM}$. This is the same as the template body discovered in Example 4, which is:
```
<table><tr>
  <xsl:apply-templates select="./date"/>
  <xsl:apply-templates select="./team[1]"/>
  <xsl:apply-templates select="./team[2]"/>
</tr></table>
```
4. Close the template rule: `</xsl:template>`

4.6 Refining the XSLT Stylesheet

In some cases, the (new) HTML document obtained by applying the generated XSLT stylesheet to the XML document may not be accurate, due to the wrong ordering of instructions within a template. i.e. there exist differences between this (new) HTML document and the original (user-defined) HTML document. By examining such differences, we can improve the accuracy of the stylesheets generated by XSLTGen. This step is applicable in circumstances where we have a set of complete and accurate mappings between the XML and HTML documents, but generated erroneous XSLT code based on these mappings. If the discovered mappings themselves are incorrect or incomplete, then this refinement step will not be effective and it is better to address the problem by improving the matching techniques. An indicator that we have complete and accurate mappings is that each element in the new HTML document corresponds exactly to the element in the original HTML document at the same depth.

Refining can be effective in situations where the generated XSLT stylesheet can be fixed by applying local move operations within the template matches, i.e. the generated stylesheet is inaccurate due to the wrong ordering of XSLT instructions within the template rules. This situation typically occurs when we have two or more XML nodes with the same name and are located at the same depth in the XML DOM, but have different order or sequence of children. In this case, the mappings generated are complete and accurate, however our XSLT stylesheet constructor assumes that these XML nodes have the same order of children and hence, it follows the order of the first node (amongst these XML nodes) encountered in the pre-order traversal of the XML DOM. Therefore, the main objective of this refining step is to fix the order of XSLT instructions within the template matches of the generated stylesheet, so that the resulting HTML document is closer to or exactly the same as the original HTML document.

A naive approach to the above problem is to use brute force and attempt all possible orderings of instructions within templates until the correct one is found (i.e. until there exist no differences between the new and the original HTML documents). However, this approach is prohibitively costly. Therefore, we adopt a heuristic approach, which begins by examining the differences between the original and the new HTML documents. We employ a change-detection algorithm [6], that produces a sequence of edit operations needed to transform the original HTML document to the new HTML document. The types of edit operations returned are insert, delete, change, and move. Of course, other similar change detection algorithms such as [9, 18] could potentially be used instead.

To carry out the refinement, the edit operation that we focus on is the move operations, since we want to swap around the XSLT instructions in a template match to get the correct order. In order for this to work, we require that there are no missing XSLT instructions for any template match in the XSLT stylesheet. After examining all move operations, this procedure is started over using the fixed XSLT stylesheet. This repetition is stopped when no move operations are found in one iteration; or, the number of move operations found in one iteration is greater than those found in the previous iteration. The second condition is

required to prevent the possibility of fixing the stylesheet incorrectly. We want the number of move operations to decrease in each iteration until it reaches zero. A sketch of the refinement algorithm is presented in Fig. 10.

```
procedure ImproveXSLT(XML, HTML, XSLT, M_E, M_SM, M_OM)
begin
1.   M_prev = ∅
2.   Hist = ∅    /* List of XSLT instructions that have been moved */
3.   repeat
4.        Apply XSLT to XML to get HTML_new
5.        Find the set of edit operations, E, between HTML and HTML_new
6.        M = move operations in E
7.        if (|M| = 0 or |M| > |M_prev|)
8.             return
9.        M_prev = M
10.       NoFix = 0    /* Number of modifications made to XSLT in the current iteration */
11.       for each move m ∈ M
12.            Find the template match to be fixed, T, in XSLT
13.            Find the specific XSLT instruction to be moved, I, within T
14.            if I ∉ Hist
15.                 Move I into the correct place using the information from M
16.                 Add I to Hist
17.                 Increment NoFix
18.  until
19.       No new modifications made to XSLT, i.e. NoFix = 0
end
```

Fig. 10. The XSLT stylesheet refining algorithm (sketch)

5 Empirical Evaluation

We have conducted experiments to study and measure the performance of the XSLTGen system. Our goals were to evaluate the matching accuracy of XSLTGen, and verify that XSLTGen generates useful XSLT stylesheets with a variety of XML and HTML documents. The system is written in Java, and employs a library for HTML cleaning, *JTidy*[4].

5.1 Data Sets

It is difficult to test this system on a class of documents, due to the lack of other suitable automatic generation systems for comparison. However, to give the reader some idea on how XSLTGen performs, we evaluated XSLTGen on four examples taken from a popular XSLT book[5] and a real-life data taken from MSN Messenger[6] chat log.

These datasets were originally pairs of (XML document, XSLT stylesheet). To get the HTML document associated with each dataset, we apply the original

[4] http://sourceforge.net/projects/jtidy
[5] http://www.wrox.com/books/0764543814.shtml
[6] http://messenger.ninemsn.com/Default.aspx

XSLT stylesheet to the XML document using Xalan[7] XSLT processor. We choose the datasets such that they exhibit a wide variety of characteristics. This is useful for benchmarking the performance of the XSLTGen system with different types of data. The characteristics of the five datasets are shown in Table 7. The Books dataset has its HTML document in a tabular format and the structure of the XML is fairly similar. The Itinerary dataset contains no extra nodes in its HTML DOM, unlike Books. Also, its HTML DOM is roughly twice the size of its XML DOM. The Poem dataset is different from the previous two, containing a large number of special HTML elements. The Soccer dataset has many more HTML nodes than XML nodes and also a very large number of extra nodes. The Chat Log dataset is noteworthy in that it has a large maximum number of children per node, 20 in each DOM, which is significantly bigger than any of the other datasets.

Table 7. Datasets used in our experiments

Datasets x : xml h : html		# elem node	# non-leaf elem node	depth	# text node	# attr node	# extra HTML node	# special HTML element	max # children / node
Books	x	17	5	2	12	4	-	-	3
	h	28	7	4	21	2	5	0	5
Itinerary	x	10	1	1	9	9	-	-	9
	h	21	3	3	18	0	0	0	18
Poem	x	19	4	2	15	0	-	-	6
	h	20	6	3	15	1	1	12	5
Soccer	x	25	7	2	18	13	-	-	6
	h	99	44	5	55	18	54	0	13
Chat Log	x	121	61	3	60	161	-	-	20
	h	244	45	6	105	40	100	0	20

5.2 Manual Mappings

For evaluation purposes, we manually determined the correct mappings between the XML and HTML DOMs in each dataset. These mappings include exact mappings, substring mappings, structure mappings, and 1-m mappings. Table 8 shows the number of manual mappings found for each dataset.

5.3 Experiments

For each dataset, we applied XSLTGen to find the mappings between elements in the XML and HTML DOMs, and generate an XSLT stylesheet that transforms the XML document to the HTML document. We then measured two aspects:

[7] http://xml.apache.org/xalan-j/index.html

Table 8. Manual mappings determined in the datasets

Datasets	Exact	Substring	Structure	1-m
Books	22	0	5	0
Itinerary	9	9	1	9
Poem	1	12	5	0
Soccer	48	68	7	6
Chat Log	1440	0	51	0

1. *Matching accuracy*: the percentage of the manually determined mappings that XSLTGen discovered.
2. The quality of the XSLT stylesheet inferred by XSLTGen.

To evaluate the quality of the XSLT stylesheet generated by XSLTGen in each dataset, we applied the generated XSLT stylesheet back to the XML document using Xalan and then compared the resulting HTML with the original HTML document using HTMLDiff[8]. HTMLDiff is a tool for analysing changes made between two revisions of the same file. It is commonly used for analysing HTML and XML documents. The differences may be viewed visually in a browser, or analysed at the source level.

5.4 Matching Accuracy

Table 9 shows the matching accuracy on the different datasets for the subsystems of XSLTGen. As shown in the table, XSLTGen achieves high matching accuracy across all five datasets. Exact mappings reach 100% accuracy in four out of five datasets. In the dataset *Chat Log*, exact mappings reach 86% accuracy which is still deemed significantly accurate. This is because there are undiscovered mappings from XML ATTRIBUTE NODEs to HTML ATTRIBUTE NODEs, which violates our assumption in Sect. 4.2 that the value of an HTML ATTRIBUTE_NODE is usually specific to the display of the HTML document in the Web browsers and is not generated from a text within the XML document. Substring mappings achieve 100% accuracy in the datasets *Itinerary* and *Soccer*. In contrast, substring mappings achieve 0% accuracy in the dataset *Poem*. This poor performance is caused by incorrectly classifying the substring mappings as exact mappings during the text matching process. In the datasets *Books* and *Chat Log*, substring mappings do not exist. Structure mappings achieve perfect accuracy in all datasets except *Poem*. In the dataset *Poem*, the structure mappings achieve 80% accuracy because the XML node `author` is incorrectly matched with the HTML TEXT_NODE "`Rupert Brooke`" in text matching, while it should be matched with the HTML node `div` in structure matching. Following the success of the other mappings, 1-m mappings also achieve 100% accuracy in all applicable datasets, i.e. *Itinerary* and *Soccer*. In the datasets *Books*, *Poem* and *Chat Log*, there are no 1-m mappings.

[8] http://www.componentsoftware.com/products/HTMLDiff/

Table 9. Matching accuracy of XSLTGen (in %)

Datasets	Exact	Substring	Structure	1-m
Books	100.00		100.00	
Itinerary	100.00	100.00	100.00	100.00
Poem	100.00	0.00	80.00	
Soccer	100.00	100.00	100.00	100.00
Chat Log	86.11		100.00	

The results indicate that in most of these cases, the XSLTGen system is capable of discovering complete and accurate mappings.

5.5 Quality of Generated XSLT Stylesheets

Table 10 shows the result of comparing the original and the new HTML document, i.e. the one produced by applying the generated XSLT stylesheet to the XML document, for different datasets in terms of the percentage of correct nodes.

Table 10. Percentage of correct nodes in the new HTML document for each dataset

Datasets	Element Nodes	Text Nodes	Attributes Nodes
Books	100.00	85.71	100.00
Itinerary	100.00	100.00	100.00
Poem	100.00	14.29	80.00
Soccer	100.00	100.00	100.00
Chat Log	100.00	100.00	75.00

As shown in the table, the new HTML documents have a high percentage of correct nodes. Using HTMLDiff, the results of comparing the new HTML document with the original HTML document in each dataset is reflected in the table. The accuracy is very high. In the datasets *Itinerary* and *Soccer*, the HTML documents being compared are identical. This is shown by the achievement of 100% in all types of nodes. In the dataset *Poem*, the two HTML documents have exactly the same appearance in Web browsers, but according to HTMLDiff, there are some missing whitespaces in each line within the paragraphs of the new HTML document. That is the reason why the percentage of correct TEXT_NODEs in the *Poem* dataset is very low (14%). The only possible explanation in this case is that in the text matching subsystem, we remove the leading and trailing whitespaces of a string before the matching is done. The improvement stage also does not fix the stylesheet since there are no move operations. In the dataset *Books*, the difference occurs in the first column of the table. In the original HTML

document, the first column is a sequence of numbers 1, 2, 3, and 4; whereas in the new HTML document, the first column is a sequence of 1s. This is because the numbers 1, 2, 3, and 4 in the original HTML document are represented using four *extra node*s, and our template rule constructor in the XSLT stylesheet generator assumes that all *extra node*s that are cousins (their parent are siblings and have the same node name) have the same structure and values. Since in this dataset the four *extra node*s have different text values, the percentage of correct TEXT_NODEs in the new HTML document is slightly affected (86%). Lastly, the differences between the original and the new HTML documents in the dataset *Chat Log* are caused by the undiscovered mappings mentioned in the previous subsection. Because of these undiscovered mappings, it is not possible to fix the XSLT stylesheet in the improvement stage. These undiscovered mappings affect some ATTRIBUTE_NODEs in the new HTML document but the percentage of correct ATTRIBUTE_NODEs is still acceptable (75%).

5.6 Discussion and Future Work

We have also tested XSLTGen on many other examples. In general, it seems to perform most effectively in situations where the XSLT stylesheet that needs to be generated follows a 'fill-in-the-blanks' style design pattern [16]. In such situations, the structure of the required stylesheet is rather similar to the desired output, with variable data being retrieved from the XML and inserted at particular points. The stylesheets generated were also tested on extra structurally similar versions of the given examples, that were generated by insertions and deletions of subtrees. The precision was similar to that for the original examples.

However, there are some problems that prevent XSLTGen from obtaining higher matching accuracy. First, in a few cases, XSLTGen is not able to discover some mappings which deal with relationships between XML ATTRIBUTE_NODEs and HTML ATTRIBUTE_NODEs. The reason is that these mappings violate our assumption stated in Sect. 4.2. This problem can be alleviated by adding HTML ATTRIBUTE_NODEs in the matching process. Undiscovered mappings are also caused by incorrectly matching some nodes, which is the second problem faced in the matching process. Incorrect matchings typically occur when an XML or an HTML TEXT_NODE has some ELEMENT_NODE siblings. In some cases, these nodes should be matched during the text matching process, while in other cases they should be matched in structure matching. Here, the challenge will be in developing matching techniques that are able to determine whether a TEXT_NODE should be matched during text matching or structure matching. The third problem concerns with incorrectly classified mappings. This problem only occurs between a substring mapping and an exact mapping, when the compared strings have some leading and trailing whitespaces. Determining whether the whitespaces should be kept or removed is a difficult choice. In many cases, the whitespaces should be removed in order to correctly match some nodes. This is how whitespaces are treated in our text matching. However in some cases, the whitespaces should be kept since they are used to specify the formatting of the

HTML document. This problem can be mitigated by adding an option to keep or remove the whitespaces.

Besides this, as the theme of our text matching subsystem is text-based matching (matching two strings), the performance of the matching process decreases if the supplied documents contain mainly numerical data. In this case, the mappings discovered, especially substring mappings, are often inaccurate and conflicting, i.e. more than one HTML nodes match a single XML node. This is also true for cases where very complex restructurings of the data are need to be performed, such as unnesting and normalizing.

The mappings discovered certainly are an important basis for generating a good and accurate XSLT stylesheet. Undiscovered mappings and incorrect matchings cause the generated stylesheet to be erroneous, since the HTML nodes that are supposed to have matching XML nodes are treated as *extra node*s and thus, are directly copied to the corresponding template. Although in some cases the HTML document generated using this kind of stylesheet is identical to the original HTML document, this behaviour obviously reduces the reusability of the XSLT stylesheet, since it contains information specific to a particular XML document. On the other hand, incorrectly classified mappings do not cause serious problems in the XSLT stylesheet. They may or may not affect the appearance of the HTML document generated using this stylesheet in a browser.

However, having complete and accurate mappings does not guarantee that the generated XSLT stylesheet will be accurate and of high quality. Another critical factor that should be considered is the similarities and differences between the structure of the XML document and the structure of the HTML document.

Occasionally within those complete and accurate mappings, there is more than one mapping discovered for the same XML node. This case only happens when two or more strings in the HTML document are generated from a single string in the XML document. In this case, the generated XSLT stylesheet is inconsistent, since it contains conflicting template rules. Hence, an additional extension to XSLTGen is to make it be aware of such cases, and not generate conflicting template rules, but instead, integrate the template of each of these template rules to the appropriate `<xsl:apply-template>` instruction that calls it, taking into account the structures of the two documents.

We note that the current version of XSLTGen does not support the capability to automatically generate XSLT stylesheets with complex functions (e.g. sorting). This is a very challenging task and an interesting direction for future work. Another direction for future work would be to modify XSLTGen so that it uses a DTD, if it has been provided with the input XML document in the automatic generation process. Of course, if the desired output is HTML, then the standard DTD for the HTML language is unlikely to be useful.

Lastly, we observe that the focus of the XSLTGen has been on producing quality stylesheets, rather than minimising execution time. Improving the runtime efficiency and scalability of the system for very large documents are interesting ways to enhance the system. However, we expect the XSLTGen system to

be deployed in a static, rather than dynamic manner (i.e. run once-only for an input-output pair, rather than repeatedly) and so this is not a primary issue.

6 Related Work

Recently, there has been much work in the literature about XML document transformations, in which only a few address the problem of generating XSLT stylesheet automatically. To the best of our knowledge, this work mainly focuses on XML to XML transformations and the techniques involved are specific to the XML to XML transformations, such as element/tag names comparison, which is impossible in our case since XML and HTML have completely different tag names; and the use of XML Schemas. Although it may be possible to generate an XML Schema for an HTML document, it would not be particularly useful in our scenario.

[13] presents a system that captures the semantics of the XML schemas and using these semantics to automatically generate the necessary XSLTs. The system firstly defines a rich information model using ontology and then maps the schema's elements, complex types, and simple types to the information model, thereby formally capturing the schemas' semantics. While the creation of information model is partially automated by arbitrarily declaring an ontological concept per schema component; the mapping process requires a human intervention. In the next step, the active semantic hub is used to automatically generate the XSLT based on the element's meanings. The algorithm find elements of the source and target that mapped to the same ontological concepts, or to concepts that can be related to each other with encoded conversion rules.

A semi-automatic XSLT stylesheet generation is also invoked within the IDACT system [23], which is a tool for automating the compilation of heterogeneous scientific datasets. If a suitable transformation is not found in its database, IDACT attempts to create a new transformation. IDACT firstly represents the input and output XML documents as trees, and then determines the relationships between them, by considering the XML element names, or the XML element content formats. By searching a library of XSLT conversions or functions, and a database of element name relationships, IDACT can build an entire XSLT stylesheet from library components. However, if IDACT finds an element that does not have a relationship, the user is prompted to provide one. This new relationship and the new XSLT stylesheet will then be saved and made available for future use.

[11] describes an approach to an automatic XML to XML transformation generator that is based on a theory of information-preserving and approximating XML operations and their associated DTD transformations. The system strictly requires the presence of DTDs of both the source and target documents. The process starts with identifying an algebra of information-preserving and - approxi- mating XML transformations. This is done by defining corresponding relations on XML trees, which are induced by operations on XML values. The operations considered as information-preserving are *renaming of tags*, *regroup-*

ing, and *congruence*; while the one considered as information-approximating is *deletion*. The *renaming of tags* is comparable to our exact mapping. However, its procedure involves some measures of similarity between tag names, which is not applicable to XSLTGen since XML and HTML have completely different set of tags. The *congruence* operation is similar to our structure matching. The next step is to construct a search space of DTDs by applying algebra operations and find a path from a source DTD element to the required target DTD element. The path represents the sequence of operations that realise the sought transformation. Based on the presentation in the paper, the approach seems to be a theoretical one and does not appear to have been implemented in an actual system.

In [3], a conceptual modelling based approach is used for performing semantic matching. It introduces a new layered model for XML schemas, called LIMXS, which offers a semantic view for XML schemas through the specification of concepts and semantic relationships among them. This model initiates a dynamic and incremental algorithm for finding semantic mappings. The system initially performs semantic matching between a source and target semantic views. Once semantic mapping is generated and validated by user, the result is given to the logical layer, which performs logical matching. Finally, an XSLT code is automatically produced. In a way, XSLTGen also uses the dynamic and incremental approach with our text matching, structure matching, and sequence checking, but without the need for user interaction.

The only prior work about XML to HTML transformations of which we are aware of is XSLbyDemo [24], a system that generates an XSLT stylesheet by example. The process begins with transforming the XML document to an HTML page using an XSLT stylesheet that was manually created taking into account the DTD of the XML document. This HTML page is referred to as *initial HTML page*. The user then modifies the initial HTML page using a WYSIWYG editor and their actions are recorded in an operation history. Based on the user's operation history, an XSLT stylesheet is generated. Obviously, this system is not automatic, since the user directly involves at some stages of the XSLT generation process. Hence, it is not comparable to our fully automatic XSLTGen system. Specifically, our approach differs from XSLbyDemo in three key ways:

1. Our algorithm produces an XSLT stylesheet consisting of transformations from an XML document to an HTML document, while XSLbyDemo generates transformations from an initial HTML to its modified HTML document.
2. Following the first argument, our generated XSLT can be applied directly to other XML documents from the same document class, whereas using XSLbyDemo, the other XML documents have to be converted to their initial HTML pages before the generated XSLT can be applied.
3. Finally, our users do not have to be familiar with a WYSIWYG editor and the need of providing structural information through the editing actions. The only thing that they need to possess is knowledge of a basic HTML tool.

In the process of generating XSL Transformations, XSLTGen involves a step of matching the supplied XML and HTML documents, i.e. finding semantic

mappings between the XML and HTML tags. The rest of this section focuses on the related work in the area of tree matching and semantic mapping.

6.1 Tree Matching

There are a number of algorithms available for tree matching. Work done in [1, 28, 29, 32] on the tree distance problem or tree-to-tree correction problem and work done in [6, 20] known as the change-detection algorithm, compare and discover the sequence of edit operations needed to transform the source tree into the result tree given. These algorithms are mainly based on structure matching, and their input comprises of two labelled trees of the same type, i.e. two HTML trees or two XML trees. The text matching involved is very simple and limited since it compares only the labels of the trees. More recent algorithms on tree matching and main memory change detection for XML include the XyDiff system [9] and work in [18], which leverages relational database technology.

6.2 Semantic Mapping

In the field of semantic mapping, a significant amount of work has focused on schema matching (refer to [27] for survey). Schema matching is similar to our matching problem in the sense that two different schemas, with different sets of element names and data instances, are compared. However, the two schemas being compared are mostly from the same domain and therefore, their element names are different but comparable. Besides using structure matching, most of the schema mapping systems rely on element name matchers to match schemas. The TransSCM system [22] matches schema based on the structure and names of the SGML tags extracted from DTD files by using concept of labelled graphs. The Artemis system [2, 5] measures similarity of element names, data types and structure to match schemas. *In XSLTGen, it is impossible to compare the element names since XML and HTML have completely different tag names.*

XMapper [17] is a system built for finding semantic mappings between structured documents within a given domain, particularly XML sources. This system uses an inductive machine learning approach to improve accuracy of mappings for XML data sources, whose data types are either identical or very similar, and the tag names between these data sources are significantly different. In essence, this system is suitable for our matching process since the tag names of XML and HTML documents are absolutely different. However, this system is not automatic since it requires the user to select one matching tag between two documents.

The Clio system [15] is an interactive, semi-automated tool for computing schema matchings. It was introduced for the relational model in [21] and was based on *value correspondences* provided by the user, in order to create the corresponding data transformation/query. In [31], the system has been extended by using instances to refine schema matchings. Refinements are obtained by inferring schema matchings from operations applied to example data, which is done by the user who manipulates the data interactively. User interaction is also needed in [25] where a two-phase approach for schema matching is proposed. The second

phase, called *semantic translation*, generates transformations that preserve given constraints on the schema. However, if few or even no constraints are available, the approach does not work well. It is clear that the algorithm for finding schema matching used in the Clio system is not suitable for our work, since user interaction is required along the phases of finding schema matchings. In addition to this, the Clio system is applicable only to structured and semi-structured data that can be described by a schema (a relational schema, a nested XML Schema, or DTD). It is not applicable to the exchange of documents or unstructured data (e.g. HTML documents, multimedia, and unstructured text) [25].

Recent work in the area of ontology matching also focuses on the problem of finding semantic mappings between two ontologies. GLUE system [10] employs machine learning techniques to semi-automatically create such semantic mappings. Given two ontologies: for each node in one ontology, the purpose is to find the most similar node in the other ontology using the notions of *Similarity Measures* and *Relaxation Labelling*. Similar to our matching process, the basis used in the *similarity measure* and *relaxation labelling* are data values and the structure of the ontologies, respectively. However, GLUE is only capable of finding 1-1 mappings whereas our matching process is able to discover not only 1-1 mappings but also 1-m mappings and m-1 mappings (in substring mappings).

The main difference between mapping in XSLTGen and other mapping systems, is that in XSLTGen we believe that mappings exist between elements in the XML and HTML documents, since the HTML document is derived from the XML document by the user; whereas in other systems, the mappings may not exist. Moreover, the mappings generated by the matching process in XSLTGen are used to generate code (an XSLT stylesheet) and that is why the mappings found have to be accurate and complete, while in schema matching and ontology matching, the purpose is only to find the most similar nodes between the two sources, without further processing of the results. To accommodate the XSLT stylesheet generation, XSLTGen is capable of finding 1-1 mappings and 1-m mappings; whereas the other mapping systems focus exclusively on discovering 1-1 mappings. Besides this, the matching subsystem in XSLTGen has the advantage of having very similar and related data sources, since the HTML data is derived from the XML data. Hence, they can be used as the primary basis to find the mappings. In other systems, the data instances in the two sources are completely different, the only association that they have is that the sources come from the same domain. Following this argument, XSLTGen discovers the mappings between two different types of document, i.e. an XML and an HTML document, whereas the other systems compare two documents of the same type. Finally, another important aspect which differs XSLTGen from several other systems, is that the process of discovering the mappings which will then be used to generate XSLT stylesheet is completely automatic.

7 Conclusion

With the massive upsurge in the data exchange and publishing on the Web, simple conversion of data from its stored representation (XML) to its publishing format (HTML) is becoming increasingly important. XSLT plays a prominent role in transforming XML documents into HTML documents. However, XSLT is difficult for users to learn.

We have devised the XSLTGen system, a system for automatically generating an XSLT stylesheet, given a source XML document and a desired output HTML document. This is useful for helping users to learn XSLT. The main strong characteristics of the generated XSLT stylesheets are accuracy and reusability. We have described how the notions of text matching, structure matching and sequence checking, enable XSLTGen to discover not only 1-1 semantic mappings between the elements in the XML and HTML documents, but also 1-m mappings between the two documents. We have described a fully automatic XSLT generation system that generates XSLT rules based on the mappings found. Our experiments showed that XSLTGen can achieve high matching accuracy and produce high quality XSLT stylesheets.

References

[1] D.T. Barnard, N. Duncan, and G. Clarke. Tree-to-tree Correction for Document Trees. Technical Report 95–372, Department of Computing and Information Science, Queen's University, Kingston, 1995.

[2] S. Bergamaschi, S. Castano, S.D.C.D. Vimeracati, and M. Vincini. An Intelligent Approach to Information Integration. In *Proceedings of the 1st International Conference on Formal Ontology in Information Systems*, pages 253–267, Trento, Italy, June 1998.

[3] A. Boukottaya, C. Vanoirbeek, F. Paganelli, and O.A. Khaled. Automating XML Documents Transformations: A Conceptual Modelling Based Approach. In *Proceedings of the 1st Asia-Pacific Conference on Conceptual Modelling*, pages 81–90, Dunedin, New Zealand, January 2004.

[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation, October 2000. *http://www.w3.org/TR/REC − xml.*

[5] S. Castano and V.D. Antonellis. A Schema Analysis and Reconciliation Tool Environment for Heterogeneous Databases. In *Proceedings of the 1999 International Database Engineering and Applications Symposium*, pages 53–62, Montreal, Canada, August 1999.

[6] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change Detection in Hierarchically Structured Information. In *Proceedings of the 1996 International Conference on Management of Data*, pages 493–504, Montreal, Canada, June 1996.

[7] J. Clark. *XSL Transformation (XSLT) Version 1.0.* W3C Recommendation, November 1999. *http://www.w3.org/TR/xslt.*

[8] J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C Recommendation, November 1999. *http://www.w3.org/TR/xpath.*

[9] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *ICDE*, pages 41–52, 2002.
[10] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map between Ontologies on the Semantic Web. In *Proceedings of the 11th International Conference on World Wide Web*, pages 662–673, Honolulu, USA, May 2002.
[11] M. Erwig. Toward the Automatic Derivation of XML Transformations. In *Proceedings of the 1st International Workshop on XML Schema and Data Management*, pages 342–354, Chicago, USA, October 2003.
[12] A.L. Hors et.al. *Document Object Model (DOM) Level 2 Core Specification Version 1.0*. W3C Recommendation, November 2000. *http://www.w3.org/TR/DOM-Level-2-Core*.
[13] J. Fox. Generating XSLT with a Semantic Hub. In *Proceedings of the 2002 XML Conference*, Baltimore, USA, December 2002.
[14] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning Document Type Descriptors from XML Document Collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, January 2003.
[15] L.M. Haas, R.J. Miller, B. Niswonger, M.T. Roth, P.M. Schwarx, and E.L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyong Integration. *Bulleting of the IEEE Computer Society Technical Committee on Data Engineering*, 22(1):31–36, March 1999.
[16] M. Kay. *XSLT Programmer's Reference*. Wrox Press Ltd., 2000.
[17] L. Kurgan, W. Swiercz, and K.J. Cios. Semantic Mapping of XML Tags using Inductive Machine Learning. In *Proceedings of the 2002 International Conference on Machine Learning and Applications*, pages 99–109, Las Vegas, USA, June 2002.
[18] E. Leonardi, S. Bhowmick, T. Dharma, and Madria S. Detecting content changes on ordered xml documents using relational databases. In *DEXA*, pages 580–590, 2004.
[19] M. Leventhal. XSL Considered Harmful. *http://www.xml.com/pub/a/1999/05/xsl/xslconsidered_1.html*, 1999.
[20] S. Lim and Y. Ng. An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies. In *Proceedings of the 17th International Conference on Data Engineering*, pages 303–312, Heidelberg, Germany, April 2001.
[21] R.J. Miller, L.M. Haas, and M.A. Hernández. Schema Mapping as Query Discovery. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 77–88, Cairo, Egypt, September 2000.
[22] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 122–133, New York, USA, August 1998.
[23] K.L. Nance and B. Hay. IDACT: Automating Data Discovery and Compilation. In *Proceedings of the 2004 Nasa's Earth Science Technology Conference*, Palo Alto, USA, June 2003.
[24] K. Ono, T. Koyanagi, M. Abe, and M. Hori. XSLT Stylesheet Generation by Example with WYSIWYG Editing. In *Proceedings of the 2002 International Symposium on Applications and the Internet*, Nara, Japan, March 2002.
[25] L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernández, and R. Fagin. Translating Web Data. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 598–609, Hong Kong, China, August 2002.
[26] D. Raggett, A.L. Hors, and I. Jacobs. *Hypertext Markup Language (HTML) 4.01*. W3C Recommendation, December 1999. *http://www.w3.org/TR/html4*.

[27] E. Rahm and P.A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, December 2001.

[28] S.M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 6(6):184–186, December 1977.

[29] K.C. Tai. The Tree-to-Tree Correction Problem. *Journal of the ACM*, 26(3):422–433, July 1979.

[30] S. Waworuntu and J. Bailey. XSLTGen: A system for automatically generating XML transformations via semantic mappings. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)*, volume LNCS 3288, pages 479–492, November, 2004.

[31] L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Santa Barbara, USA, May 2001.

[32] K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal of Computing*, 18(6):1245–1262, December 1989.

An Ontology-Guided Approach to Change Detection of the Semantic Web Data*

Li Qin[1] and Vijayalakshmi Atluri[2]

[1] Department of Marketing and CIS, Western New England College
1215 Wilbraham Road, Springfield, MA 01119
lq274250@wnec.edu
[2] CIMIC and MSIS Department, Rutgers University
180 University Avenue, Newark, NJ 07102
atluri@cimic.rutgers.edu

Abstract. To achieve improved availability and performance, often, local copies of remote data from autonomous sources are maintained. Web search engines are the primary examples of such services. Increasingly, these services are utilizing the Semantic Web as it is often envisioned as a machine-interpretable web. In order to keep the local repositories current, it is essential to synchronize their content with that of their original sources. Change detection is the first step to accomplish this. It is essential to have efficient change detection mechanisms as the size of the local repositories is often very large.
In this paper, we present an approach that exploits the semantic relationships among the concepts in guiding the change detection process. Given changes to some seed instances, a reasoning engine fires a set of pre-defined rules to characterize the profile of the changed target instances. In addition to change detection, our proposed semantics-based approach of utilizing semantic associations can be utilized in other applications such as guiding information discovery for agents, consistency maintenance among distributed information sources, among others.

1 Introduction

To achieve improved availability and performance, often, local copies of remote data from autonomous sources are maintained. Web search engines are the primary examples of such services. Increasingly, these services are utilizing the Semantic Web as it is often envisioned as a machine-interpretable web. In order to keep the local repositories current, it is essential to synchronize their content with that of their original sources. Change detection is the first step to accomplish this, which essentially is to find whether and what changes have occurred to data of interest, especially those owned and updated by autonomous sources [23]. For example, a search engine has to detect changes to data published by autonomous sources in order to synchronize its local copies and its index with their sources.

* A preliminary version of this paper has appeared in "L. Qin and V. Atluri, Ontology-guided Change Detection to the Semantic Web Data, 23rd International Conference on Conceptual Modeling (ER 2004)."

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 130–157, 2006.

The major challenge confronting change detection lies in the conflict between limited availability of resources for change detection and the enormity of the data available. As an extension of WWW, the Semantic Web [2] will continue to be decentralized with its information space projected to increase at a much faster pace than resource availability. The Semantic Web will be ubiquitous since it is a machine-interpretable web targeted for automation, integration and reuse of data across different applications. As a result, it is essential to have efficient change detection mechanisms as the size of the local repositories is often very large. In addition to search engines, local repositories include data warehouses, cache maintenance and knowledge archival applications for the Semantic Web.

Earlier approaches to change detection to web pages rely on the link structure among web pages or statistics estimated offline (e.g., change frequencies) [6,16]. The Semantic Web will no longer be simply pages and links, but is a network of resources whose semantics and interrelationships are explicitly stated. As such, it enables machines to make inferences and deductions about these resources. Specifically, semantic Web technologies (e.g. RDF [19], RDF Schema [20] and OWL [15]) provide methods and standards to enrich data instances with metadata, which are defined as concepts and properties in ontologies. Each ontology is a formal, explicit specification of shared conceptualization of a given domain of discourse. As a result, machines can interpret the data instances by following the pointer embedded in their annotation to seek their meaning in the ontologies.

In this paper, we exploit the rich specifications of the Semantic Web in efficiently guiding the change detection process. In particular, we exploit the semantic relationships at ontological level as ontologies present richer and more complete semantics. We assume that, if a data instance changes, it is highly likely that all its semantically related instances also have changed. Therefore, our approach gives priority to those related instances while searching for changes. To this end, we identify certain inferences rules among concepts, properties, and instances of concepts as well as their changes. Given changes detected to seed instances, our reasoning engine uses these rules to generate a profile of target instances that are likely to have changed. This profile essentially contains information such as the concepts that target instances belong to, how they are semantically related to the seed instances and any properties that target instances should or should not instantiate. To our knowledge, we are the first to investigate change detection in the domain of the Semantic Web, and to utilize semantic relationships in this process.

Since the detected changes are semantically related, it may reveal something more interesting than what can be discovered by observing each change separately. For example, if consistent changes are witnessed to multiple independent sources, this may increase the trust-worthiness of the detected changes so that they can be trusted to identify more changes. By taking advantage of the Semantic Web infrastructure, change detection can ultimately become a well-controlled process. In other words, instead of visiting pages in a blind way, one may target more accurately the pages to visit. In addition, our proposed semantics-based approach of utilizing semantic associations can be used in other applications such as guiding information discovery for agents, consistency maintenance among distributed information sources, among others. Taking the consis-

tency maintenance as an example, independent sources may publish contents that are semantically related. Therefore, when the contents of one source change, other sources should have their contents updated accordingly.

This paper is organized as follows. In section 2, we provide an overview of our intelligent change detection approach guided by ontologies. Since our approach fully takes advantage of the Semantic Web infrastructure, section 3 presents preliminaries on ontologies, instances, and relationships accompanied by examples. We elaborate the types of changes to the Semantic Web data and profiles in section 4. Note that various inference rules are used by the reasoning engine to make smart decisions about the target instances. Therefore, section 5 presents concept inference rules, property inference rules, instance inference rules, change inference rules and profile inference rules. Section 6 presents the system architecture for the implementation of our intelligent change detection system and the details of the reasoning process performed within the reasoning engine. However, due to the limited number of the Semantic Web data, we are yet not in a position to experiment on the efficiency and scalability of our approach. A brief overview of the related work is provided in section 7. Conclusions and an insight into our future work are provided in section 8.

2 Overview

Our approach begins with identifying different types of inference rules based on ontologies. These rules are independent of any specific ontology to ensure their universal applicability, and they are exploited by the reasoning engine in guiding the change detection process. Our change detection approach is ontology-guided since all these rules act based on the ontologies involved. We identify the following five categories of inference rules:

1. **Change Inference Rules:** Given changes to seed instances, these rules imply changes to their semantically related instances.
   The set of change inference rules is extensible, meaning that similar rules to be identified in the future can be added to this set.
2. **Profile Inference Rules:** There is a profile inference rule corresponding to each change inference rule. Once any change inference rule is fired, the corresponding profile inference rule is used to derive a profile for characterizing these instances. This profile consists of descriptions of target concepts possibly in terms of $eq[c]$[3]and $sub[c]$, target instances in terms of $eq[i]$ and $da[i]$, and target properties in terms of $eq[p]$ and $sup[p]$.
   Given the profile for target instances, the following three rules are used to further reason about the specifics for these instances: The concept inference rules derive the concepts that these instances belong to; the instance inference rules derive how these instances are associated with other visited instances; the property inference rules derive whether these instances should or should not instantiate certain properties.
3. **Concept Inference Rules:** If the description of target concepts contains $eq[c]$ or $sub[c]$, concept inference rules are called to derive the specific constituent concepts.

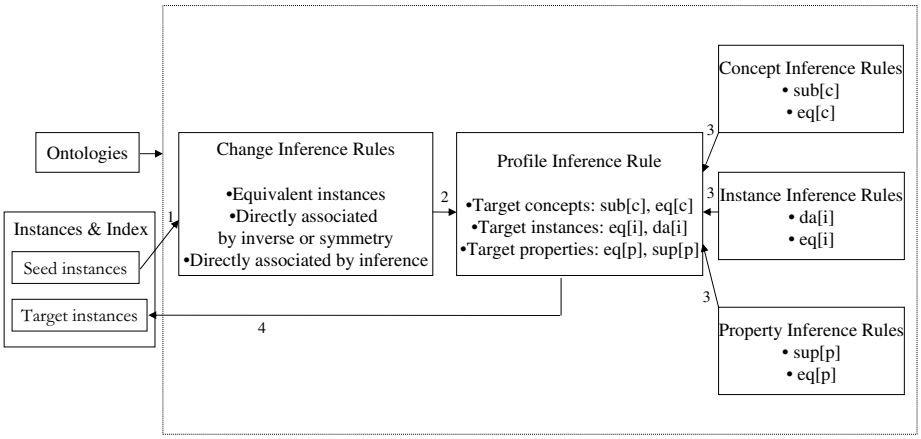


Fig. 1. Information and rules used by the reasoning engine

4. **Property Inference Rules:** If the description of target properties contains $eq[p]$ or $sup[p]$, property inference rules are called to derive the specific constituent properties.
5. **Instance Inference Rules:** If the description of target instances contains $eq[i]$ or $da[i]$, instance inference rules are called to derive how the specific target instances are semantically related to the seed instance.

Figure 1 shows the components involved in the reasoning, where the arrows and numbers between components show how and when each component is used by the reasoning engine. It starts with some seed instances, and finds changes to them. These changes become input to the change inference rules. Assume that the entry page for the MSIS Dept. of Rutgers, which is an instance of concept 'Department' has been visited, and the 'address' of the department has been updated, which happens when the department moves to a new building.

Given the change(s) detected to this seed instance, the reasoning engine checks the change inference rules and fires those relevant to the detected change(s) and the ontologies, shown as step 1 in Figure 1. Let us assume that one of the change inference rules is stated as follows: "if 'address' property can infer any property of the semantically associated concepts, then this inferred property value, which belongs to instances associated with this department instance, should have changed." Then, the reasoning engine visits the ontology that this department instance points to, and finds out that 'address' property is defined to the concept 'Academic Unit', of which concept 'Department' is defined as a specialization. Also, concept 'Employee' associates itself with concept

[3] $eq[c]$ and $sub[c]$ represent the set of equivalent and subclass concepts of concept c, respectively; $eq[p]$ and $sup[p]$ represent the set of equivalent properties and super-properties of property p, respectively; $eq[i]$ and $da[i]$ represent the set of equivalent and directly associated instances to instance i, respectively.

'Academic Unit' through object property 'worksFor', and it has been identified that the 'business address' property of concept 'Employee' can be inferred through the 'address' property of concept 'Academic Unit'. This means that if the 'address' property of an instance of 'Department' has changed, then the value to the 'Business Address' of all the instances of 'Employee' directly associated with this 'Department' instance should also have changed. As a result, the above change inference rule is fired.

Since the fired change inference rule implies possible changes to some semantically related instances, the corresponding profile inference rules are triggered (indicated by step 2 in Figure 1). In our example, the profile inference rule for semantically related instances by inference is triggered.

Shown as step 3 in Figure 1, the profile inference rules generate a profile consisting of target concepts, target instances and target properties with each set described by certain operators. In our example, if concept 'Employee' has subclass concepts 'Faculty', 'Staff' and 'Ph.D Student' with each having their own subclass concepts, then the target concepts will contain the operator for the subclass concepts of 'Employee' as sub[Employee]. To derive their specific constituent elements, the operators for concepts, instances and properties in the profile description will call the concept inference rules, instance inference rules and property inference rules, respectively. For our example, the concept inference rules will be called to derive the specific concepts constituting sub[Employee], which includes concepts 'Faculty', 'Staff', 'Ph.D Student' as well as their subclass concepts, if exist. After that, the reasoning engine finally derives a profile for the target instances to be the instances of concepts 'Employee', 'Faculty', 'Staff', 'Ph.D Student' as well as their subclass concepts or equivalent concepts. In addition, these instances are directly associated with the MSIS Department through 'worksFor' property, and have its 'business address' property instantiated.

The profile is used to locate the actual target instances if the profile is satisfiable by certain instances (shown as step 4 in Figure 1). The target data instances for our example may be located in the personal web pages of the department's faculty members, the faculty list page on the department's web site, and so on.

3 Preliminaries

In this section, we present the preliminaries on ontologies, instances and relationships among concepts and instances, and introduce denotations that will be used in the latter sections of this paper.

3.1 Ontologies

Ontologies are an essential component of the Semantic Web since ontologies provide interpretations to the contents of web data. A well-cited definition for an ontology is given by Gruber [9] as a "specification of a conceptualization". Generally, an ontology defined for a domain contains a description of important concepts in the domain, properties of each concept, relationships among these concepts as well as restrictions or axioms upon properties in terms of cardinality, property value type, domain and range of a relationship, and so on.

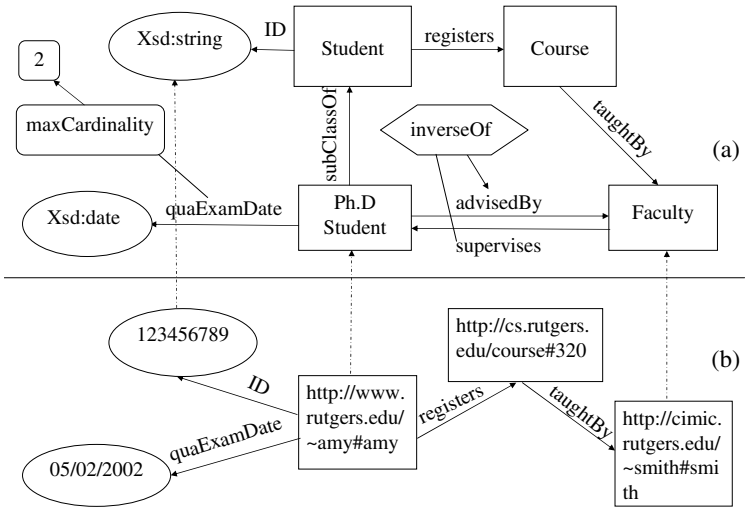


Fig. 2. Example for ontology and instances

Figure 2(a) is an example of an ontology, available at 'http://cimic.rutgers.edu/ontologies/university'. Here, rectangles represent concepts with their properties indicated as labels on the edges from concepts to their values, ovals represent datatype property values, rounded rectangles represent restrictions as well as their values and hexagons represent axioms of properties. Figure 2(b) contains some instances of this ontology, where instances are shown by rectangles and values to the datatype properties are shown in ovals. The dashed lines across these two parts indicate the correspondence between the instances and ontologies. We will use examples from this figure to complement our discussion in this section.

An ontology o_i consists of the following elements:

1. **concepts:** Each ontology o contains a set of concepts, $C[o] = \{c_1, c_2, \ldots, c_n\}$.

   For example, for the ontology in Figure 2, C['http://cimic.rutgers.edu/ontologies/university'] = {Student, Ph.D Student, Course, Faculty}.
2. **properties:** For each concept $c \in C[o]$, there exists a set of properties $P[c] = DP[c] \cup OP[c]$, where
   - $DP[c] = \{dp_1, dp_2, \ldots, dp_m\}$ are *datatype properties* of concept c, each taking a primitive data type as the value, and
   - $OP[c] = \{op_1, op_2, \ldots, op_n\}$ are *object properties*, each taking some concept(s) as the value.

   Note that properties are directional. We use $dp_i[c]$ to represent the datatype property dp_i of concept c, and $v[dp_i[c]]$ for the value taken by this property. Similarly, we use $op_i[c]$ to denote the object property op_i of concept c with $v[op_i[c]]$ for its value.

For example, for concept 'Ph.D Student' in Figure 2, P[Ph.D Student] = $\{subClassOf$, quaExamDate, advisedBy$\}$, with DP[Ph.D Student] = $\{$quaExamDate$\}$ where quaExamDate[Ph.D Student] = $xsd : date$ (quaExamDate represents the date of qualifier exam for Ph.D Students) and OP[Ph.D Student] = $\{subClassOf$, advisedBy$\}$.

Object properties of a concept c can be of two types:

- *domain-independent* ($OP_{DI}[c]$): Domain-independent object properties have pre-defined meaning that does not vary from one ontology to another.
- *domain-specific* ($OP_{DS}[c]$): The meaning of domain-specific object properties depends on the context in which it is defined.

Using the same example, OP_{DI}[Ph.D Student] = $\{subClassOf\}$ where $subClassOf$[Ph.D Student] = Student (which means 'Ph.D Student' is a subclass concept of 'Student') and OP_{DS}[Ph.D Student] = $\{$advisedBy$\}$ where advisedBy[Ph.D Student] = Faculty. The domain-specific property 'advisedBy', when considered in a medical domain, may represent the relationship between 'Patient' and 'Doctor', which is different from what we use here between 'Ph.D Student' and 'Faculty'. On the other hand, domain-independent relationships such as $subClassOf$ have the same predefined semantics irrespective of the domain in which it is used.

3. **restrictions:** For each property $p \in P[c]$ where $c \in C[o]$, there exists a set of restrictions on the value or cardinality of the property, denoted by $R[p] = \{r_1, \ldots, r_w\}$ where $r_k[p]$ is the restriction r_k of property p and $v[r_k[p]]$ is the value to the restriction.

   For example, R[quaExamDate] = $\{maxCardinality\}$ where $maxCardinality$[quaExamDate] = 2, which means each Ph.D student should have at most two date values for their qualifying exam (quaExamDate property).
4. **axioms:** For each property, $p \in P[c]$ where $c \in C[o]$, there exists a set of axioms with each defined by itself (unary) or in relation to another property (binary), denoted by $A[p] = \{a_1, a_2, \ldots, a_n\}$. $a_i[p]$ represents the axiom a_i of property p and if a_i is binary, $v[a_i[p]]$ denotes the property related to p through axiom a_i.

   Figure 2 shows that $inverseOf$[advisedBy] = supervises, where supervises $\in P$[Faculty] and supervises[Faculty] = Ph.D Student. Therefore, *inverseOf* is a binary axiom relating the properties 'advisedBy' and 'supervises'.

Different ontology languages may support different types of domain-independent object properties, restrictions and axioms upon properties. For instance, OWL Full [15] supports at least the following domain-independent object properties: *subClassOf, equivalentClass, intersectionOf, unionOf, complementOf.* It supports restrictions such as *cardinality, minCardinality, maxCardinality*, and axioms such as *subPropertyOf* (binary), *equivalentProperty* (binary), *TransitiveProperty* (unary), *SymmetricProperty* (unary), *FunctionalProperty* (unary), *InverseFunctionalProperty* (unary), *inverseOf* (binary), and so on.

Though our approach is not limited to any specific ontology language, to simplify our discussion, in this paper, we resort to the OWL Full vocabulary with their semantics. For completeness, we briefly explain some of them in the appendix.

3.2 Instances

We first discuss how instantiation should be done based on ontologies and then the components of an instance.

Ontologies Vs. Instances Ontologies define and relate concepts used to annotate the web data, which are instances to the concepts in ontologies. Different elements of ontologies serve different functions: concepts along with their datatype properties and domain-specific object properties are what can be instantiated by instances, restrictions upon properties specify the requirements that valid instantiation should satisfy, domain-independent object properties and the axioms of properties provide powerful mechanisms for enhanced reasoning about instances. A semantic document is an aggregation of instances of different concepts with some or all of the properties of each concept instantiated.

Note that in ontologies, a property is only specified to the most general concept to which it applies; the subclass concepts of this concept can inherit all of its properties. Therefore, the instances of these subclass concepts can instantiate these inherited properties as well as their own by assigning a (valid) value to it. For a concept c_i, we use $P'[c_i]$ to represent all the properties that instances of concept c_i can instantiate. Therefore, $P'[c_i]$ includes all the datatype properties and domain-specific object properties of c_i and its superclass concepts. Therefore, $P'[c_i] = DP[c_i] \cup OP_{DS}[c_i] \cup DP[c_j] \cup OP_{DS}[c_j]$, where $c_j = sup[c_i]$ or $c_j = eq[c_i]$. Similarly, $DP'[c_i]$ and $OP'_{DS}[c_i]$ represent the set of datatype properties and domain-specific object properties that instance i can instantiate where i is an instance of c_i, i.e. $i \in I[c_i]$. $R'[p]$ is the set of restrictions on property p. Based on Figure 2(a), since $subClassOf$[Ph.D Student] = Student, P'[Ph.D Student] = {quaExamDate,advisedBy,ID,registers}; DP'[Ph.D Student] = {ID,quaExamDate}; OP'_{DS}[Ph.D Student] = {registers, advisedBy} where registers[Ph.D Student] = Course and advisedBy[Ph.D Student] = Faculty; R'[advisedBy] = $\{maxCardinality\}$ where $maxCardinality$[advisedBy] = 2. Note that the instantiation of an object property of a concept taking another concept as its value represents a mapping (direct association) between the instances belonging to these two concepts.

If c_i is a subclass of c_j, then all the instances of c_i are also instances of c_j. To be clear, we use the following notation:

- $I[c]$ for the instances which are explicitly asserted to belong to concept c, and $i \in I[c]$ only if an instance i is asserted to belong to a concept c.
- $I'[c]$ to refer to all the instances of concept c by explicit assertion and by inheritance, in which case, $I'[c] = I[c] \cup I[c_i]$ for any c_i, where $c_i \in sub[c]$ or $c_i \in eq[c]$.

For instance, $subClassOf$[Ph.D Student] = Student and 'http://www.rutgers.edu/amy#amy' $\in$ I[Ph.D student]. Therefore, 'http://www.rutgers.edu/amy#amy' $\in$ I'[student]. For an instance i, we use $C[i]$ to represent the set of concepts, where $i \in I'[c]$. Therefore, C['http://www.rutgers.edu/amy#amy'] = {Student,Ph.D Student}.

Components of an Instance: Each concept c has a set of instances $I[c] = \{i_1, \ldots, i_n\}$ and every $i_k \in I[c]$ is a 4-tuple $i_j = \langle URI_k, c, DP'[c], OP'_{DS}[c] \rangle$, such that

1. URI_k is a Universal Resource Identifier by which i_k can be universally identified and other instances can refer to it.
2. c is the concept that i_k is asserted to instantiate.
3. $DP'[c]$ is a set of datatype property instantiations where each $dp_k \in DP'[c]$ takes a specific value v_k, denoted as $i_k : dp_k = v_k$, where v_k has a specified primitive data type as its domain.
4. $OP'_{DS}[c]$ is a set of object property instantiations, where each $op_k \in OP'_{DS}[c]$ and $op_k[c] = c_j$ takes an instance i_j as its value, denoted as $i_k : op_k = i_j$ where $i_j \in I'[c_j]$.

We use $P[i]$, $DP[i]$ and $OP[i]$ to denote the set of properties, datatype properties and object properties instantiated by instance i. If $i \in I[c]$, $p \in P[i]$, then $p \in P'[c]$. Also, if $i \in I[c]$, then $P[i] \subseteq P'[c]$, $DP[i] \subseteq DP'[c]$, $OP[i] \subseteq OP'[c]$.

Take the instance of concept 'Ph.D Student' in Figure 2 as an example. The URI for this instance is 'http://www.rutgers.edu/amy#amy' and the concept it belongs to is 'Ph.D Student'. P['http://www.rutgers.edu/amy#amy'] = {ID,quaExamDate,registers} with DP['http://www.rutgers.edu/amy#amy'] = {ID,quaExamDate}
and OP_{DS}['http://www.rutgers.edu/amy#amy'] = {registers}.
More specifically, 'http://www.rutgers.edu/amy#amy':ID = 123456789,
'http://www.rutgers.edu/amy#amy':quaExamDate = 05/02/2002 and
'http://www.rutgers.edu/amy#amy':registers = 'http://cs.rutgers.edu/course#320'.

3.3 Relationships

Given two concepts c_i and c_j, we say c_i and c_j are directly associated through op_i if $op_i[c_i] = c_j$ or $op_i[c_j] = c_i$. We use $da[c_i]$ to denote the set of concepts that are directly associated with concept c_i and $da[c_i : op_i]$ to denote the concept directly associated with concept c_i through object property op_i. For example, da[Ph.D Student] = {Student,Faculty}. In particular, da[Ph.D Student:$subClassOf$] = Student and da[Ph.D Student:advisedBy] = Faculty.

The relationship between two instances can be directly associated, indirectly associated or not associated. We will first discuss equivalence between instances as a special category, whose transitivity allows it to be either directly associated or indirectly associated. In this paper, we focus only on direct association between instances because our change and profile inference rules are mainly specified upon directly associated instances.

Note that the relationship between instances covers both explicit and implicit relationships. By explicit relationship, we mean the relationship between the instances is asserted explicitly by content creators. In other cases, the implicit relationship between two instances can be derived through reasoning. The equivalence between instances we discuss below is implicit, if it is evaluated based on the value of its identification property.

Equivalent instances Each instance should be given a URI by its author for identifying the instance and for other instances to refer to it. URIs are decentralized and "Two URIs are different unless they are the same character for character." [1] However, different URIs may be equivalent if they refer to the same real world object. For example,

information on a faculty member who works for a university may be published as an instance in the university's faculty directory, the department's faculty page, some research project's participating faculty page, the faculty member's personal web page, and so on. In other words, considering the 'Faculty' instance example in Figure 2, this faculty member on his personal web page has a URI of 'http://cimic.rutgers.edu/smith#smith' whereas the same faculty instance published in the faculty list of his department may have a URI of 'http://cs.rutgers.edu/Faculty#smith'. These instances refer to the same faculty member, thus are equivalent with each other though each of these web sites gives it a different URI and may or may not assert it as equivalent to the other.

We notice that equivalent instances published in different sources may be matched based on the value of some identification or quasi-identification property (or a combination of multiple properties), similar to the primary key in a relational database table. Instances of different concepts may be equivalent. For instance, an international Ph.D student may be instantiated as an instance of concepts such as 'Student', 'International Student', 'Graduate Student' or 'Ph.D Student'. In particular, if concepts share a property defined to be *owl:InverseFunctionalProperty* (defined in the appendix), then instances that have the same value for this property are equivalent. In other words, a property, whose axiom defines it to be an *owl:InverseFunctionalProperty*, is actually an identification property. Besides, equivalence between instances can also be explicitly indicated using *owl:sameIndividualAs*.

Equivalence between instances: Two instances are equivalent if they refer to the same entity in the real world. We use $eq[i_m]$ for the set of equivalent instances of i_m with i_m itself included. If instances $i_m \in I[c_i]$ and $i_n \in I[c_j]$ are equivalent, then

- c_i and c_j share an identification datatype property dp_k such that $i_m : dp_k = i_n : dp_k$; or c_i and c_j share an identification object property op_k such that $i_m : op_k = i_n : op_k$ (or $i_m : op_k \in eq[i_n : op_k]$);
- c_i and c_j refer to the same concept, or c_i is an equivalent concept of c_j, or c_i and c_j are subclass concepts of c_k where dp_k(or op_k) $\in P[c_k]$.

This definition will be the foundation for the instance inference rule on equivalent instances, discussed in section 5.

Directly associated instances: For instances i_i and i_j where $i_i \in I'[c_i]$, $i_j \in I'[c_j]$, if there exists $op_i \in OP'[c_i]$ such that $op_i[c_i] = c_k$, $c_k \in C[i_j]$ and $i_i : op_i = i_j$, then we say i_i and i_j are directly associated instances through op_i.

For instance i, we use $da[i]$ for the set of instances directly associated with i, and $da[i : op]$ represents the set of instances directly associated with i through op. Take an instance from Figure 2 as an example, da['http://www.rutgers.edu/amy#amy':registers] = 'http://cs.rutgers.edu/course#320'.

4 Changes to the Semantic Web Data and Profiles

Based on the changes to the Semantic Web data and ontologies, the reasoning engine fires the appropriate change inference rules and profile inference rules to generate a profile for characterizing the target instances. While we outline, in section 4.1, the different types of changes to the Semantic Web data, in section 4.2, we describe the components of a profile.

4.1 Changes to the Semantic Web Data

Detecting changes to the Semantic Web data requires one to first identify whether or not a change has occurred to the instantiation of concepts along with their properties. If a change has occurred, then the actual changes to the instances and their properties need to be identified. A change to the Semantic Web data instance is denoted by Δ. Three types of changes are significant to our change detection approach:

- Addition(Deletion) of property instantiation. $\Delta^A(\Delta^D) = \langle i, c, p, v\rangle$, where i is the instance involved in the addition (deletion), c the concept to which i belongs, i.e. $i \in I[c]$; p the property added or deleted in the change, and v the value of p involved in the addition (deletion);
- Update to property instantiation value. $\Delta^U = \langle i, c, p, ov, nv\rangle$, where i is the instance being updated, c the concept to which i belongs, i.e. $i \in I[c]$; p the property being updated, ov the old value of p before the update, and nv the new value of p after the update.

In case of change to concept membership, we may treat it as a combination of deletion of all the properties of an instance followed by subsequent additions.

Each change to the Semantic Web data above constitutes a 'complete delta' [14] in the sense that it provides sufficient information for transforming the old version to the new version, and vice versa.

Given a specific change Δ , we use $i[\Delta]$, $c[\Delta]$, $p[\Delta]$, $dp[\Delta]$, $op[\Delta]$ to denote the instance, the concept to which the instance belongs, the property, the datatype property and object property involved in the change, respectively. We use $v[\Delta^A]$ to denote the value to $p[\Delta^A]$ added in the change, $v[\Delta^D]$ to denote the value to $p[\Delta^D]$ deleted by the change, $ov[\Delta^U]$ to denote the old value of $p[\Delta^U]$ before the change and $nv[\Delta^U]$ to denote the new value of $p[\Delta^U]$ after the change.

4.2 Profiles

Profile. A profile generated by the reasoning engine consists of the following descriptions to characterize the target instances:

- **Target concepts (TC):** This is the set of concept(s) that target instances belong to.
- **Target instances (TI):** TI may consist of instances identified explicitly by its URI or instances whose properties take specific values.
- **Target Properties (TP):** This is the set of properties that target instances may or may not instantiate, based on the type of the change. Specifically, TP are instantiated if the change is Δ^U or Δ^D, but they are not instantiated if the change is Δ^A. We use EXISTS and NOT EXISTS to indicate whether or not TP are instantiated.

The profile generated for the target instances describes their common characteristics, which is further used to locate these instances.

5 Inference Rules

Based on some detected changes and ontologies, the reasoning engine makes smart decisions about what instances should have changed and describes them through a profile in terms of target concepts, instances and properties. The reasoning process relies on different types of inference rules, which are triggered in a specific sequence: It first fires the change inference rules based on the changes to seed instances and ontologies, then the profile inference rules to generate the profile, in which concept inference rules, property inference rules and instance inference rules may be triggered. Below, we present concept inference rules, property inference rules, instance inference rules, change inference rules and profile inference rules in sections 5.1, 5.2, 5.3, 5.4 and 5.5, respectively.

5.1 Concept Inference Rules

The components of an ontology described in section 3 are based on what is explicitly stated in the ontology. The significance of ontologies to the Semantic Web definitely goes beyond its components by allowing for reasoning through domain-independent object properties and axioms. Domain-independent object properties such as $subClassOf$ and $equivalentClass$ have their own axioms such as transitivity. Operators including $eq[c_i]$ and $sub[c_i]$ are used in the profile inference rules to represent the target concepts for a profile. The following inferences rules specify how the set of equivalent and subclass concepts can be generated for a given concept based on the ontology.

Concept Inference Rules: We identify the following inference rules for identifying $sub[c]$ and $eq[c]$ for a given concept c. Let c_i, c_j and c_k be three concepts.

1. $(equivalentClass[c_k] = c_i \vee equivalentClass[c_k] = c_j$ where $c_j \in eq[c_i]) \Rightarrow (c_k \in eq[c_i])$. This means, if c_k is asserted as $equivalentClass$ of c_i or $equivalentClass$ of c_j such that c_j belongs to the set of concepts equivalent to c_i, then c_k belongs to the set of concepts equivalent to c_i.
2. $(c_k \in eq[c_i]) \Rightarrow (c_i \in eq[c_k])$. If c_k belongs to the set of concepts equivalent to c_i, then c_i also belongs to the set of concepts equivalent to c_k.
3. $(subClassOf[c_k] = c_i \vee subClassOf[c_k] = c_j$ where $c_j \in sub[c_i]) \Rightarrow (c_k \in sub[c_i])$; and $(c_i \in intersectionOf[c_k] \vee c_k \in unionOf[c_i]) \Rightarrow (subClassOf[c_k] = c_i)$. This means, if c_k is asserted as ***subClassOf*** c_i or ***subClassOf*** c_j such that c_j belongs to the set of subclass concepts of c_i, then c_k belongs to the set of subclass concepts of c_i. Moreover, c_k is a subclass of c_i can also be implied if c_k is the intersection of c_i and some other concepts, or if c_i is the union of c_k and some other concepts.
4. $(c_k \in eq[c_j] \wedge c_k \in sub[c_i]) \Rightarrow (c_j \in sub[c_i])$. This means, if c_k belongs to the set of concepts equivalent to c_j and the set of subclass concepts of c_i, then c_j also belongs to the set of subclass concepts of c_i.
5. $(c_k \in eq[c_j] \wedge c_i \in sub[c_k]) \Rightarrow (c_i \in sub[c_j])$. This mean, if c_i belongs to the set of subclass concepts of c_k and c_k belongs to the set of equivalent concepts of c_j, then c_i also belongs to the set of subclass concepts of c_j.

In the following, we will explain with an example, how target concepts can be generated using the concept inference rules stated above. Figure 3 shows concepts c_1,...,c_8 with the domain-independent object properties specified among them. Let us assume that the profile inference rules generate a profile with target concepts = $\{c_1, sub[c_1], eq[c_1]\}$. According to this figure, $equivalentClass[c_2] = c_1$ and $equivalentClass[c_2] = c_3$. From Rule 1, we infer that $c_2 \in eq[c_1]$ and $c_2 \in eq[c_3]$. Then, based on Rule 2, one may infer $c_3 \in eq[c_2]$. Finally, based on $c_2 \in eq[c_1]$ and $c_3 \in eq[c_2]$, one can infer $c_3 \in eq[c_1]$ by using Rule 1 again. After considering how c_1, c_2, c_3 are related to other concepts, we conclude $eq[c_1] = \{c_2, c_3\}$. Now, let us examine what concepts constitute $sub[c_1]$. First, based on Rule 1, $c_4 \in sub[c_1]$ and $c_7 \in sub[c_1]$ can be inferred. Based on $equivalentClass[c_4] = c_5$ (therefore $c_4 \in eq[c_5]$) and $c_4 \in sub[c_1]$, one may infer $c_5 \in sub[c_1]$ by using Rule 4. Also, $c_6 \in sub[c_1]$ can be inferred based on $subClassOf[c_6] = c_2$ and $c_2 \in eq[c_1]$ using Rule 5. Finally, $c_8 \in sub[c_1]$ can be inferred based on $c_6 \in sub[c_1]$ and $c_8 \in eq[c_6]$ further using Rule 4. As a result, $sub[c_1] = \{c_4, c_5, c_6, c_7, c_8\}$. By substituting the $eq[c_1]$ and $sub[c_1]$ in the target concepts, we get target concepts = $\{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$.

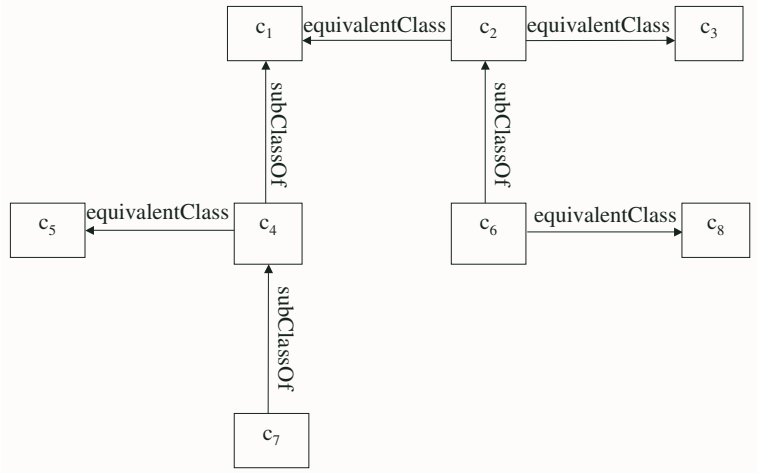


Fig. 3. Example for $sub[c]$ and $eq[c]$

For concept 'Student' in Figure 2, the following concept inference rule is fired for identifying sub[student]: $subClassOf$[Ph.D Student] = Student $\Rightarrow$ Ph.D Student $\in$ sub[Student].

5.2 Property Inference Rules

Similar rules for $sup[p]$ and $eq[p]$ can be built based on $subPropertyOf$ and $equivalentProperty$. For a property p, we use $sup[p]$ to represent the super-properties

of p, i.e. the properties to which p is a sub-property, and $eq[p]$ to represent all the properties equivalent to p.

Property Inference Rules: We identify the following inference rules for identifying $sup[p]$ and $eq[p]$ for a given property p. Let p_i, p_j and p_k be three properties.

1. $(equivalentProperty[p_k] = p_i \vee equivalentProperty[p_k] = p_j$ where $p_j \in eq[p_i]) \Rightarrow (p_k \in eq[p_i])$. This means, if p_k is asserted as $equivalentProperty$ of p_i, or that of p_j such that p_j belongs to the set of properties equivalent to p_i, then p_k belongs to the set of properties equivalent to p_i.
2. $(p_k \in eq[p_i]) \Rightarrow (p_i \in eq[p_k])$. This means, if p_k belongs to the set of properties equivalent to p_i, then p_i also belongs to the set of properties equivalent to p_k.
3. $(subPropertyOf[p_k] = p_i \vee subPropertyOf[p_k] = p_j$ where $p_i \in sup[p_j]) \Rightarrow (p_i \in sup[p_k])$. This means, if p_k is asserted as $subPropertyOf\ p_i$ or that of p_j such that p_i belongs to the set of super-properties of p_j, then p_i belongs to the set of super-properties of p_k.
4. $(p_k \in eq[p_j] \wedge p_k \in sup[p_i]) \Rightarrow (p_j \in sup[p_i])$. This means, if p_k belongs to the set of properties equivalent to p_j and the set of super-properties of p_i, then p_j also belongs to the set of super-properties of p_i.
5. $(p_k \in eq[p_j] \wedge p_i \in sup[p_k]) \Rightarrow (p_i \in sup[p_j])$. This means, if p_k belongs to the set of properties equivalent to p_j and p_i belongs to the set of super-properties of p_k, then p_i also belongs to the set of super-properties of p_j.

These property inference rules would help us identify the target properties from the set of properties where initial changes are detected. Let's say, the 'ID' property of 'Student' has an equivalent property 'SSN', then the following property inference rule is fired for identifying eq[ID] where ID $\in DP$[Student]: $equivalentProperty$[ID] = SSN $\Rightarrow$ SSN $\in eq$[ID].

5.3 Instance Inference Rules

Instance inference rules used by the reasoning engine can identify $eq[i]$ and $da[i]$ for a given instance i.

The complexity of identifying equivalent instances results from the distributed nature of the web, where authors of different information sources may instantiate equivalent instances by resorting to different concepts and properties for their own application, convenience or preference. Now, we focus on how equivalent instances can be generated for a given instance.

Equivalent Instance Inference Rule: We identify the following inference rule for identifying $eq[i_m]$ for a given instance.

1. Given $i_m \in I[c_m]$, $eq[i_m] = \{i | (i \in I[c_j]$ such that $c_j = c_k$, $c_j \in eq[c_k]$, or $c_j \in sub[c_k]) \wedge (i : p_k = i_m : p_k \vee i : p_k = eq[i_m : p_k])$ where $InverseFunctionalProperty \in A[p_k]$ and $p_k \in P[c_k]\}$.

This rule states that given i_m is an instance of concept c_m, the equivalent instances of i_m will be those that are instances of c_m or an equivalent concept of c_m or a subclass concept of the concept to which the identification property is defined, and take the same value for its identification property as that for i_m.

Take concept 'Student' in Figure 2 as an example, let's say, property 'ID' of concept 'Student' is an $InverseFunctionalProperty$, then for a specific instance of 'Student', 'http://www.rutgers.edu/student#123', whose 'ID' takes the value of 999999999, the following instance inference rule is fired for identifying its equivalent instances: 'http://www.rutgers.edu/student#123' $\in I$[Student] and ID $\in P$[Student] and $InverseFunctionalProperty \in A$[Student] and 'http://www.rutgers.edu/student#123' :ID = 99999999 $\Rightarrow eq$['http://www.rutgers.edu/student#123'] = $\{i|i \in I[c]$ such as c = Student or $c \in eq$[Student] or $c \in sub$[Student] and i:ID = 999999999$\}$.

Directly Associated Instance Inference Rules: We identify the following inference rules for identifying $da[i]$ for a given instance i.

1. Given $i_m \in I'[c_m]$, if $op_m[c_k] = c_j$ (or $op_m[c_j] = c_k$) where $c_k \in C[i_m]$ and $SymmetricProperty \notin A[op_m]$, where $A[p]$ as the set of axioms over property p, then $da[i_m : op_m] = \{i|i \in I'[c_j] \wedge i_n : op_m = i_j$ where $i_n \in eq[i_m]$ and $i_j \in eq[i]$ (or $i_j : op_m = i_n$ where $i_n \in eq[i_m]$ and $i_j \in eq[i]$) $\}$.
   This rule states that, i_m is an instance (by assertion or inference) of concept c_m, and c_m is directly associated with concept c_j through object property op_m which is not a symmetric property ($op_m[c_m] = c_j$). Then the directly associated instances of i_m through op_m belong to concept c_j (by assertion or inference), and these instances or their equivalent instances are taken as the value to object property op_m of i_m or its equivalent instances. It also implies that, if c_j is directly associated with c_m through object property op_m ($op_m[c_j] = c_m$), then the directly associated instances of i_m through op_m belong to concept c_j (by assertion or inference), and these instances or their equivalent instances take i_m or its equivalence as the value to object property op_m.
2. Given $i_m \in I'[c_m]$, if $op_m[c_k] = c_j$ (or $op_m[c_j] = c_k$) where $c_k \in C[i_m]$ and $SymmetricProperty \in A[op_m]$, then $da[i_m : op_m] = \{i|i \in I'[c_j] \wedge (i_n : op_m = i_j \vee i_j : op_m = i_n$ where $i_n \in eq[i_m]$ and $i_j \in eq[i])\}$.
   This states that, i_m is an instance of concept c_m, c_m is directly associated with concept c_j through object property op_m, $op_m[c_m] = c_j$, (or c_j is directly associated with c_m through object property op_m, $op_m[c_j] = c_m$), and op_m is a symmetric property. Then the directly associated instances of i_m through op_m belong to concept c_j, these instances or their equivalence either are taken as the value to object property op_m of i_m or its equivalent instances, or take i_m or its equivalence as the value to object property op_m. In the case that c_m and c_j are directly associated through a symmetric property, $c_m = c_j$ or $c_m \in eq[c_j]$.
3. Given $i_m \in I'[c_m]$, if $op_m[c_k] = c_j$ where $c_k \in C[i_m]$ and $inverseOf[op_m] = op_j$ (or $inverseOf[op_j] = op_m$), then $da[i_m : op_m] = \{i|i \in I'[c_j] \wedge (i_n : op_m = i_j$ where $i_n \in eq[i_m]$ and $i_j \in eq[i])\}$; $da[i_m : op_j] = \{i|i \in I'[c_j] \wedge (i_j : op_j = i_n$ where $i_n \in eq[i_m]$ and $i_j \in eq[i])\}$; $da[i_m : op_m, op_j] = \{i|i \in I'[c_j] \wedge (i_n : op_m = i_j \vee i_j : op_j = i_n$ where $i_n \in eq[i_m]$ and $i_j \in eq[i])\}$.

This states that, i_m is an instance of concept c_m, c_m is directly associated with concept c_j through object property op_m, $op_m[c_m] = c_j$, and property op_m is $inverseOf\ op_j$, $op_j[c_j] = c_m$. Then the directly associated instances of i_m through op_m and op_j belong to concept c_j, these instances or their equivalent instances either are taken as the value to object property op_m of i_m or its equivalent instances, or take i_m or its equivalent instances as the value to object property op_j.

For example, in Figure 2, concept 'Ph.D Student' is directly associated with concept 'Faculty' through object properties 'advisedBy' and 'supervises'. Then for a specific instance of concept 'Ph.D Student', 'http://www.rutgers.edu/phdstudent#222', the following instance inference rule is fired to identify its directly associated instances through 'advisedBy' and 'supervises': 'http://www.rutgers.edu/phdstudent#222' $\in I$[Ph.D Student], advisedBy[Ph.D Student] = Faculty and $inverseOf$[advisedBy] = supervises $\Rightarrow$ da['http://www.rutgers.edu/phdstudent#222':advisedBy, supervises] = $\{i | i \in I'$[Faculty] $\wedge$ (i_m:advisedBy = $i_j \vee i_j$:supervises = i_m where $i_m \in eq$['http://www.rutgers.edu/phdstudent#222'] and $i_j \in eq[i]$)$\}$.

The profile generated by the profile inference rules may call the rules defined above to get the target instances.

5.4 Change Inference Rules

Our previous inference rules are defined through reasoning based on the ontologies whereas the change inference rules we define below are based on the heuristics that replicated or semantically related information are expected to change in a consistent way, as the result of the efforts by the same or even different information sources to maintain freshness and consistency of their data.

1. $\forall \Delta_i$, if $i_j \in eq[i[\Delta_i]]$, then there exists Δ_j where $i[\Delta_j] = i_j$.
2. $\forall \Delta_i$, $i[\Delta_i] : op[\Delta_i] = i_j$, there exists Δ_j where $i[\Delta_j] = i_j$ or $i[\Delta_j] = i_k$ where $i_k \in eq[i[\Delta_j]]$; If $SymmetricProperty \in A[op[\Delta_i]]$, then $op[\Delta_j] = op[\Delta_i]$, else $inverseOf[op[\Delta_i]] = op[\Delta_j]$.
3. For $p_i \in P[c_i]$, $p_j \in P[c_j]$ and $op_i[c_i] = c_j$, if p_j can be inferred from p_i, $\forall \Delta_i$ where $op[\Delta_i] = p_i$, $i[\Delta_i] = i_i$, there exists Δ_j where $op[\Delta_j] = p_j$, $i[\Delta_j] \in da[i_i : op_i]$ or $i[\Delta_j] \in da[i_i : op_i, op_j]$ if $inverseOf[op_i] = op_j$.

The first rule states that, if an instance has changed and it has equivalent instances, it implies that these equivalent instances also have changed. So, the question now is how to find out the profile for the equivalent instances of a given instance especially when their equivalence is not explicitly specified.

The second rule states that, if a change involves an object property of an instance in which this object property has an inverse property defined in the ontology, or this object property itself is symmetric, then it implies that the directly associated instances as well as their equivalent ones have also changed.

The third rule states that, if the value to a property has been changed and another property of the directly associated concept can be inferred from the changed property it implies the value to the other property involved should also have changed. The property inference relationship we proposed in this rule can be identified by domain experts.

For example, assume an instance of 'Ph.D Student' in Figure 2 has its 'advisedBy' property changed, since this property has an inverse property 'supervises' which belongs to concept 'Faculty', then the second change inference rule above is fired: Δ_i and $i[\Delta_i]$:advisedBy = i_j and $inverseOf$[advisedBy] = supervises $\Rightarrow$ Δ_j where $i[\Delta_j] = i_j$ or $i[\Delta_j] = i_k$ where $i_k \in eq[i[\Delta_j]]$.

Though the second and the third rules both involve directly associated instances, they are significantly different: the third rule involves all the directly associated instances through the object property between the concepts; the second rule involves only some of the directly associated instances when the cardinality of the relationship between the concepts is not one-to-one.

5.5 Profile Inference Rules

Profile for Equivalent Instances Based on the first change inference rule, our focus here is to profile the equivalent instances of a given instance, to which a change has been detected. Changes are expected to the corresponding property of the equivalent instances.

In the following, we will explain each step involved in the reasoning process that helps to identify the profile of target instances. Assuming a change Δ_m has been detected to an instance i_m, where $I[\Delta_m] = i_m$ and $p[\Delta_m] = p_m$, where p_m is some property of i_m.

The reasoning engine needs to find out to which concept c_m the property is defined in the ontology, where $p[\Delta_m] \in P[c_m]$. Note that c_m is not always necessarily equal to $c[\Delta_m]$, but it could be any other concept in $sub[c_m]$ or $eq[c_m]$ since $p[\Delta_m]$ can be instantiated by any instance of these concepts.

Identifying the target instances (TI) will call the instance inference rules for $eq[i[\Delta_m]]$. As indicated in the instance inference rules, there should exist $p_I \in P'[c[\Delta_m]]$, where p_I is either designated as an identification property of c_m or is defined to be an $InverseFunctionalProperty$. For any $i_n \in eq[i_m]$, $i_n : p_I = i_m : p_I$.

Then the reasoning engine needs to find out whether p_m is the sub-property of some other property p_j where $subPropertyOf[p_m] = p_j$. The equivalent instances of $i[\Delta_m]$ may have changes to p_m or p_j. Note that a change to a property always triggers a change to its super-property, but may not necessarily change its sub-properties. The reasoning engine also checks whether p_i has any equivalent properties.

Profile for Equivalent Instances. Given a change Δ, the profile for the equivalent instances is:

TI = $\{eq[i[\Delta]]\}$,
TC = $\{c_i, sub[c_i], eq[c_i]\}$ where $p[\Delta] \in P[c_i]$, and
TP = $\{p[\Delta], eq[p[\Delta]], sup[p[\Delta]]\}$ EXISTS for Δ^D and Δ^U, NOT EXISTS for Δ^A.

Profile for Associated Instances by Inverse or Symmetry For any change involving an object property, at least two instances directly related by the property are involved: the instance to which a change is detected to its object property and the instance taken as the value to this object property. If the inverse relationship of $p[\Delta]$ is

defined in the ontology, which means there exists another object property op_i where $inverseOf[op_i] = op[\Delta]$ (or $inverseOf[op[\Delta]] = op_i$) or property $p[\Delta]$ is symmetric indicated by $SymmetricProperty \in A[p[\Delta]]$, then the change may also be reflected in the instance which is taken by the property $p[\Delta]$. The target instances may associate themselves with $i[\Delta]$ through the inverse property op_i or the symmetric property $p[\Delta]$ itself. Here comes the profile for the directly associated instances by inverse or symmetry.

Profile for Directly Associated Instances by Inverse. Given a change Δ involving $op[\Delta]$ with $inverseOf[op[\Delta]]$ defined in the ontology, the profile for the directly associated instances by inverse is:

For Δ^A:
  TI = $\{v[\Delta^A], eq[v[\Delta^A]]\}$;
For Δ^D:
  if the cardinality of $op[\Delta^D]$ is one-to-one
    TI = $\{da[i[\Delta^D] : op[\Delta^D], inverseOf[op[\Delta^D]]]\}$;
  else if the cardinality of $op[\Delta^D]$ is many-to-one
    TI = $\{da[i[\Delta^D] : op[\Delta^D]]\}$;
  else if the cardinality of $op[\Delta^D]$ is one-to-many
    TI = $\{da[i[\Delta^D] : inverseOf[op[\Delta^D]]]\}$;
  else
    TI = $\{v[\Delta^D], eq[v[\Delta^D]]\}$.
For Δ^U:
  if the cardinality of $op[\Delta^U]$ is one-to-one
    TI = $\{da[i[\Delta^U] : op[\Delta^U], inverseOf[op[\Delta^U]]], ov[\Delta^U], eq[ov[\Delta^U]], nv[\Delta^U]$, $eq[nv[\Delta^U]]\}$;
  else if the cardinality of $op[\Delta^U]$ is many-to-one
    TI = $\{da[i[\Delta^U] : op[\Delta^U]], ov[\Delta^U], eq[ov[\Delta^U]], nv[\Delta^U]$, $eq[nv[\Delta^U]]\}$;
  else if the cardinality of $op[\Delta^U]$ is one-to-many
    TI = $\{da[i[\Delta^U] : inverseOf[op[\Delta^U]]], ov[\Delta^U], eq[ov[\Delta^U]], nv[\Delta^U]$, $eq[nv[\Delta^U]]\}$;
  else
    TI = $\{ov[\Delta^U], nv[\Delta^U], eq[ov[\Delta^U]], eq[nv[\Delta^U]]\}$.
TC = $\{c_i, sub[c_i], eq[c_i]\}$ if $inverseOf[op[\Delta]] \in P[c_i]$;
TP = $\{inverseOf[op[\Delta]], eq[inverseOf[op[\Delta]]], sup[inverseOf[op[\Delta]]]\}$ EXISTS for Δ^D and Δ^U, NOT EXISTS for Δ^A.

Profile for Directly Associated Instances by Symmetry. Given a change Δ involving a symmetric property $op[\Delta]$, the profile for the directly associated instances by symmetry is:

For Δ^A:
  TI = $\{v[\Delta^A], eq[v[\Delta^A]]\}$;

For Δ^D:
  if the cardinality of $op[\Delta^D]$ is one-to-one
    TI = $\{da[i[\Delta^D] : op[\Delta^D]]\}$;
  else
    TI = $\{v[\Delta^D], eq[v[\Delta^D]]\}$.
For Δ^U:
  if the cardinality of $op[\Delta^U]$ is one-to-one
    TI = $\{da[i[\Delta^U] : op[\Delta^U]], ov[\Delta^U], eq[ov[\Delta^U]], nv[\Delta^U], eq[nv[\Delta^U]]\}$;
  else
    TI = $\{ov[\Delta^U], nv[\Delta^U], eq[ov[\Delta^U]], eq[nv[\Delta^U]]\}$.
TC = $\{c_j, sub[c_j], eq[c_j]\}$ where $op[\Delta] \in P[c_j]$;
TP = $\{op[\Delta], eq[op[\Delta]], sup[op[\Delta]]\}$ EXISTS for Δ^D and Δ^U, NOT EXISTS for Δ^A.

Profile for Directly Associated Instances by Inference. Given a change Δ and $p[\Delta]$ can infer p_i where $p_i \in P[c_j]$, the profile for directly associated instances by inference is:

TI = $\{da[i[\Delta] : op_j]\}$, if $op_j[c[\Delta]] = c_j$ or $op_j[c_j] = c[\Delta]$;
TI = $\{da[i[\Delta] : op_j, op_i]\}$, if $inverseOf[op_i] = op_j$ or $inverseOf[op_j] = op_i$
TC = $\{c_j, eq[c_j], sub[c_j]\}$;
TP = $\{p_i, eq[p_i], sup[p_i]\}$ EXISTS for Δ^D and Δ^U, NOT EXISTS for Δ^A

As an example, if the instance of 'Ph.D Student', 'http://www.rutgers.edu/phdstudent#222', which takes 'http://www.rutgers.edu/faculty#111' as the value to its 'advisedBy' property, has this property deleted. After the second change inference rule is fired, the reasoning engine will fire the profile inference rule for directly associated instances by inverse as follows: Δ^D = ⟨ 'http://www.rutgers.edu/phdstudent#222', Ph.D Student, advisedBy,'http://www.rutgers.edu/faculty#111'⟩ and cardinality of 'advisedBy' is many-to-one ⇒ The profile for the target instances is:
TC = {Faculty, eq[Faculty], sub[Faculty]},
TP = {supervises, eq[supervises], sup[supervises]},
TI = {da['http://www.rutgers.edu/phdstudent#222':advisedBy]}.

6 Ontology-Guided Change Detection

In this section, we present the system architecture for intelligent change detection in section 6.1, followed by an example in section 6.2. We particularly focus our discussion on the reasoning process performed within the reasoning engine in section 6.3.

6.1 System Architecture

Figure 4 shows the architecture of intelligent crawling with our change detection technique incorporated. Similar to a data-flow diagram, we use circles to represent processes, arrows to represent flows of data, and open-ended rectangles to represent data repositories. Note that inference and ontologies are shown as separate data stores in the architecture and they can be combined if inference can be specified within ontologies.

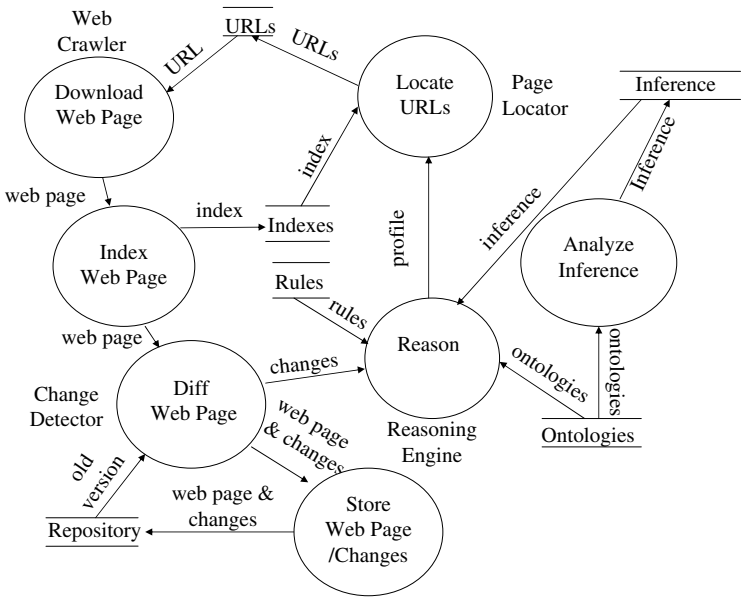


Fig. 4. Architecture for intelligent change detection

Algorithm 1 describes how the intelligent change detection is done. This algorithm works as follows: The web crawler starts downloading web pages by retrieving a URL from a given list and the full-text of the page is indexed. A diff algorithm is run to find out whether and what changes have occurred to this page by comparing it with its earlier version retrieved from the local repository. The detected changes are saved to the repository along with the new version of the page. Meanwhile, the detected changes are sent to the reasoning engine, which use the inference rules we defined in section 5 to generate a profile for the target instances to be visited next. The profile is used to locate the URLs of the instances satisfying the profile by querying the index of data instances and these URLs are appended to the front of the URL list so that the crawler can visit them next. As can be seen, the step of generating the profile for target instances is the key to our approach.

Note that this algorithm starts with (seed) instances, with inference rules fired based on ontologies and back to (target) instances. It is easy to go from instances to ontologies since each data instance contains a pointer to the ontologies where concepts and properties used for annotating the data instance are defined. But not vice versa since each ontology has no knowledge about by what data instances it is used. After the reasoning engine derives the profile describing the target instances, the URLs for these target instances satisfying the profile have to be determined. The search engine or local repository usually maintains an index of data instances. For instance, indexing data instances of a semantic document can be done based on keywords, concepts and relationships

Algorithm 1 Intelligent Change Detection

Require: URL list for seed instances, index of web pages
  while the URL list for seed instances is not empty **do**
    The web crawler retrieves a URL from the front of the list
    The web crawler downloads the web page
    The indexer indexes the web page
    The change detector diffs the web page against its previous version from the repository
    The change detector saves the web page and the changes to the repository, and sends changes to the reasoning engine
    The reasoning engine determines the profile for the target web pages to be visited
    The page locator finds out the URLs based on the profile by querying the index
    The page locator appends the URLs to the URL list
  end while

[21]. The profile is translated into queries, which are evaluated against the index of data instances, as a result, URLs for the target instances are returned.

Regarding seed instances, we suggest that important web pages be selected. Changes to important web pages imply that many other pages should also be updated. This will provide a better warranty that our algorithm will work recursively and improve the data quality of the local repository.

6.2 An Example

In this section, we present an example to show how our change detection approach works.

We start with a list of seed instances. For instance, assume the crawler visits 'http://cimic.rutgers.edu/smith' and downloads the following data instance:

<rdf:RDF xmlns:univ ="http://cimic.rutgers.edu/ontologies/university#">
<univ:Faculty rdf:ID ="smith">
  <univ:name>John Smith</univ:name>
  <univ:officephone>9739995555</univ:officephone>
  <univ:email>jsmith@cimic.rutgers.edu</univ:email>
  <univ:teaches rdf:resource = "http://cs.rutgers.edu/course#320" />
</univ:Faculty>

This page is indexed under concept 'Faculty' and relationship 'teaches' as follows:

univ:Faculty
  ('http://cimic.rutgers.edu/smith#smith':name,John Smith)
  ('http://cimic.rutgers.edu/smith#smith':officephone,9739995555)
univ:teaches(Faculty,Course)
  ('http://cimic.rutgers.edu/smith#smith','http://cs.rutgers.edu/course#320')

Then the change detector retrieves the previous version of the page. For instance, the previous version of this page is shown as follows:

```
<rdf:RDF xmlns:univ = "http://cimic.rutgers.edu/ontologies/university#">
<univ:Faculty rdf:ID ="smith">
    <univ:name>John Smith</univ:name>
    <univ:officephone>9739995555</univ:officephone>
    <univ:email>jsmith@cimic.rutgers.edu</univ:email>
</univ:Faculty>
```

The change detector diffs the new version with the previous version and finds the 'teaches' property instantiation has been added ($\Delta^A = \langle$ 'http://cimic.rutgers.edu/smith #smith',univ:Faculty,teaches,'http://cs.rutgers.edu/course#320' $\rangle$).

This change is fed into our reasoning engine and the reasoning engine finds from the instance that the ontology used by this instance is 'http://cimic.rutgers.edu/ontologies/un iversity'. Let's say, part of this ontology about 'Faculty' is as follows:

```
<!DOCTYPE owl[
<!ENTITY univ "http://cimic.rutgers.edu/ontology/university#"> ]>
<rdf:RDF xmlns = "&univ;">
<owl:Class rdf:ID = "Faculty">
    <rdfs:subClassOf rdf:resource ="#Academic Staff" />
</owl:Class>
<owl:Class rdf:ID ="Course" />
<owl:DatatypeProperty rdf:ID ="courseID">
    <rdfs:domain rdf:resource ="#Course" />
    <rdf:type rdf:resource ="&owl;InverseFunctionalProperty" />
</owl:DatatypeProperty>
<owl:Class rdf:ID ="UndergradCourse">
    <rdfs:subClassOf rdf:resource="#Course" />
</owl:Class>
<owl:Class rdf:ID ="GradCourse">
    <rdfs:subClassOf rdf:resource ="#Course" />
</owl:Class>
<owl:ObjectProperty rdf:ID ="teaches">
    <rdfs:domain rdf:resource ="#Faculty" />
    <rdfs:range rdf:resource ="#course" />
    <owl:inverseOf rdf:resource ="#taughtBy" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID ="name">
    <rdfs:domain rdf:resource ="#Faculty" />
    <rdfs:range rdf:resource ="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID ="officephone">
    <rdfs:domain rdf:resource ="#Faculty" />
```

<rdfs:range rdf:resource =“&xsd;string” />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID =“email”>
<rdfs:domain rdf:resource =“#Faculty” />
<rdfs:range rdf:resource =“&xsd;string” />
</owl:DatatypeProperty>

Given the change, the reasoning engine checks the change inference rules. Since the property involved in the change is an object property of a ‘Faculty’ instance and there exists another property as an inverse of it, the change inference rule for directly associated instances by inverse is fired: $\Delta^A = \langle$ ‘http://cimic.rutgers.edu/smith#smith’,univ:Faculty,teaches,‘http://cs.rutgers.edu/course#320’ $\rangle$ and ‘http://cimic.rutgers.edu/smith#smith’:teaches = ‘http://cs.rutgers.edu/course#320’ and $inverseOf$[teaches] = taughtBy $\Rightarrow \Delta_j$ where $i[\Delta_j]$ = ‘http://cs.rutgers.edu/course#320’ or $i[\Delta_j] = i_k$ where $i_k \in eq$[‘http://cs.rutgers.edu/course#320’].

As a result, the profile inference rule for directly associated instances by inverse is also fired: $\Delta^A = \langle$ ‘http://cimic.rutgers.edu/smith#smith’,univ:Faculty,teaches, ‘http://cs.rutgers.edu/course#320’$\rangle \Rightarrow$ The profile for the target instances is: TC = {Course, eq[Course], sub[Course]}, TP = {taughtBy, eq[taughtBy], sup[taughtBy]}, TI = {‘http://cs.rutgers.edu/course#320’, eq[‘http://cs.rutgers.edu/course#320’]}.

The reasoning engine fires concept inference rules $subClassOf$[UndergradCourse] = Course $\Rightarrow$ UndergradCourse $\in$ sub[Course] and $subClassOf$[GradCourse] = Course $\Rightarrow$ GradCourse $\in$ sub[Course] to derive that sub[Course] = {UndergradCourse, GradCourse}. As a result, TC = {Course, UndergradCourse, GradCourse}. Similarly, if no equivalent property or super-property is defined to ‘taughtBy’, then TP = {taughtBy}. The reasoning engine thens fires the instance inference rule for the equivalent instances of ‘Course’ instance ‘http://cs.rutgers.edu/course#320’, eq[‘http://cs.rutgers.edu/course#320’]. The ontology shows that concept ‘Course’ has an *InverseFunctionalProperty* ‘courseID’. The specific instance inference rule is as follows: ‘http://cs.rutgers.edu/course#320’ $\in$ I[Course], courseID $\in$ P[course] and *InverseFunctionalProperty* $\in$ A[courseID] $\Rightarrow$ eq[‘http://cs.rutgers.edu/course#320’] = $\{i | (i \in I[c_j]$ such that c_j = Course or $c_j \in sub$[Course] or $c_j \in eq$[Course]) $\wedge$ (i:courseID = ‘http://cs.rutgers.edu/course#320’: courseID$\}$. After calling the concept inference rules, the target instances include data instances of concept ‘Course’, ‘UndergradCourse’, or ‘GradCourse’, having the same value for ‘courseID’ property of ‘http://cs.rutgers.edu/course#320’, and with property ‘taughtBy’ uninstantiated. This profile is translated into a query into the index of data instance. Let’s say, the index for ‘univ:Course’, ‘UndergradCourse’, ‘GradCourse’ and ‘taughtBy’ is as follows:

univ:Course
(‘http://cs.rutgers.edu/schedule#320’:courseID,320)
(‘http://cs.rutgers.edu/course#340’:courseID,340)
(‘http://cs.rutgers.edu/course#320’:courseID,320)
univ:GradCourse
(‘http://cs.rutgers.edu/grad/course#320’:courseID,320)

univ:UndergradCourse
('http://cs.rutgers.edu/undergrad/course#220':courseID,220)
univ:taughtBy(Course,Faculty)
('http://cs.rutgers.edu/course#340','http://www.rutgers.edu/mwhite')

The target instances retrieved from querying this index with the profile will be data instances 'http://cs.rutgers.edu/schedule#320' and 'http://cs.rutgers.edu/grad/course#320', which are equivalent instances of 'Course' instance 'http://cs.rutgers.edu/course#320'. These three URLs are appended to the front of the URL list and they will be visited next.

6.3 Reasoning Engine

Our proposed inference rules are based on OWL Full, which is the most expressive sublanguage provided by the OWL language. Since no reasoning software can support every feature of OWL Full [15], in this section, we would like to further present the reasoning process performed within the reasoning engine. Note that none of the existing reasoning systems (e.g., RacerPro [18]) can support the features of OWL Full [15] used in our rules. One may use the Instance Store [11], which uses a database to support reasoning over instances.

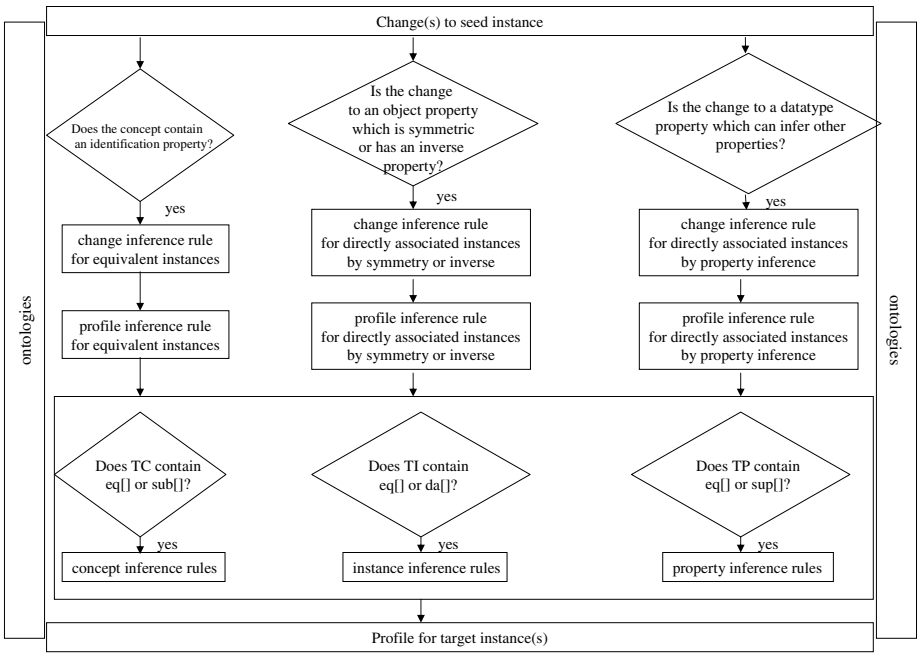


Fig. 5. Reasoning engine

In Figure 5, we present the reasoning process, starting with the changes to seed instances until the profile for the target instances is generated. This reasoning process needs to be repeated recursively. The target instances are to become the seed instances, which are further used to identify target instances. However, special care needs to be taken to prevent the seed instances in the prior processes from being identified as the target instances shortly after they are visited. Compared with determining whether a particular target instance has been visited recently after the reasoning process is done, a more efficient solution is possible by trying to avoid it before the rules are invoked. For example, if instance i_n is identified as an equivalent instance of seed instance i_m and then i_n becomes a seed instance, it is not necessary to invoke the change inference rule for equivalent instances upon i_n since it will retrieve the same set of instances as those upon i_m. However, if other new changes are detected to i_n, these new changes may trigger other change inference rules.

7 Related Work

Research has been done to study the change dynamics of web pages [3,5]. Especially, experiments [6] show that the change of web pages follows a Poisson process and the change frequency of each web page can be estimated offline based on its change history. Cho et al. further proposed synchronization strategies to improve the average freshness of the local repository by utilizing the change frequencies of web pages [7]. Qin et al. proposed object freshness as another freshness metric and discussed the operation of a tree structure, which can be used to schedule accesses to data objects at different frequencies in [16].

A large number of diff algorithms have been developed to accommodate different yet increasingly more popular data formats such as HTML and XML [4,8,22]. These diff algorithms focus on detecting the changes to a document by comparing the new version with its old version. Efficiency and optimality have been the focus of the competition among these diff algorithms.

For reasoning over the Semantic Web, the RacerPro system [18] is a commercial knowledge representation system with support for OWL DL. It can deal with reasoning over both ontologies and instances. Horrocks et al. proposed the Instance Store by using database to support reasoning over instances, providing sound and complete answers to instance retrieval queries [11]. The limitation of the Instance Store is that it does not support relationships between instances. Research attention has also been given to studying semantic relationships [21], querying the Semantic Web [10,13], changes to ontologies [12] and so on.

As for our semantics-based change detection approach under this new context, it is oriented for the Semantic Web by making use of its infrastructure and characteristics. Therefore, it works intelligently by exploiting instances and the correlation in their change behaviors under the guidance of ontologies.

8 Conclusions and Future Work

In this paper, we have presented a semantics-based change detection approach guided by ontologies for the Semantic Web data. Given changes to some instances, the reasoning engine works by firing pre-defined rules applicable to the changes, referring to the ontologies that instances point to and generating a profile for the target instances to be visited next. Due to the limited number of the Semantic Web data, we are yet not in a position to experiment on the efficiency and scalability of our approach. We are planning to implement our reasoning engine and test it over synthesized data and changes.

The target instances may be determined with more change information. For instance, a sample can be taken from the instances of a concept before reasoning is done. Or, certain instances of multiple related concepts can be visited and their changes may be propagated through the object properties to concepts of a wider range. As a result, the target instances may belong to some indirectly associated concepts. As part of our future work, we intend to study what role the semantic locality plays in change detection.

References

1. Tim Berners-Lee. Universal Resource Identifiers-Axioms of Web Architecture. Available at http://www.w3.org/DesignIssues/Axioms.html
2. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. Scientific American, May 2001.
3. Brian E. Brewington and George Cybenko. How Dynamic is the Web? 9th World Wide Web Conference (WWW9), 2000.
4. Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change Detection in Hierarchically Structured Information. ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, June 1996, pages 493-504.
5. Junghoo Cho and Hector Garcia-Molina. Estimating Frequency of Change. ACM Transactions on Internet Technology, 3(3): August 2003.
6. Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler, 26th International Conference on Very Large Databases (VLDB), September 2000, pages 200-209.
7. Junghoo Cho and Hector Garcia-Molina. Synchronizing a Database to Improve Freshness. ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, May 14-19, 2000, pages 117-128.
8. Gregory Cobena, Serge Abiteboul, and Amelie Marian. Detecting Changes in XML Documents. International Conference on Data Engineering (ICDE), 2002.
9. T. R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2), pages 199-220, 1993.
10. Ian Horrocks and Sergio Tessaris. Querying the Semantic Web: A Formal Approach. International Semantic Web Conference 2002, LNCS 2342, pp. 177-191, 2002.
11. Ian Horrocks, Lei Li, Daniele Turi and Sean Bechhofer. The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In Proc. of the 2004 Description Logic Workshop (DL 2004), pages 31-40, 2004.

12. Michel Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Finding and Characterizing Changes in Ontologies, In Proceedings of the 21st International Conference on Conceptual Modeling (ER2002), Tampere, Finland, October 7-11, 2002.
13. Aimilia Magkanaraki, Grigoris Karvounarakis, Ta Tuan Anh, Vassilis Christophides, Dimitris Plexousakis. Ontology Storage and Querying. Technical Report No 308, April 2002. Available at http://139.91.183.30:9090/RDF/publications/tr308.pdf
14. Amelie Marian, Serge Abiteboul, Grgory Cobena, and Laurent Mignet. Change-Centric Management of Versions in an XML Warehouse. 27th VLDB Conference, Rome, Italy, 2001, pages 581-590.
15. OWL Web Ontology Language Guide. Available at http://www.w3.org/TR/owl-guide/
16. Li Qin and Vijayalakshmi Atluri. An Access Scheduling Tree to Achieve Optimal Freshness in Local Repositories. Electronic Commerce and Web Technologies (EC-Web) 2003, pages 227-236.
17. Li Qin and Vijayalakshmi Atluri. Ontology-Guided Change Detection to the Semantic Web Data. 23rd International Conference on Conceptual Modeling (ER 2004).
18. RacerPro. http://www.racer-systems.com/products/racerpro/index.phtml
19. Resource Description Framework (RDF). Available at http://www.w3.org/RDF/
20. RDF Vocabulary Description Language 1.0: RDF Schema. Available at http://www.w3.org/TR/rdf-schema/
21. Amit Sheth. Relationships at the Heart of Semantic Web: Modeling, Discovering, Validating and Exploiting Complex Semantic Relationship, Keynote Address, SOFSEM 2002 (29th Annual Conference on Current Trends in Theory and Practice of Informatics), Milovy, Czech Replublic, November 2002.
22. Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: A Fast Change Detection Algorithm for XML Documents. Proceedings of International Conference on Data Engineering, 2003.
23. Jennifer Widom. Research Problems in Data Warehousing. 4th Int'l Conference on Information and Knowledge Management (CIKM), Baltimore, Maryland, Nov. 1995, pages 25-30.

9 Appendix

The following is the OWL vocabulary with their semantics. Readers may refer to the OWL language guide for more detailed information.

- *subClassOf:* If a concept c_1 is tagged as *owl:subClassOf* c_2, which we denote as *subClassOf*$[c_1] = c_2$, then c_1 can inherit all the properties of c_2, and all the instances of c_1 are also instances of c_2. e.g. 'Ph.D Student' is a subclass of 'Student'.
- *equivalentClass:* If a concept c_1 is tagged as *owl:equivalentClass* of c_1, which we denote as *equivalentClass*$[c_1] = c_2$, then the instances of c_1 are also instances of c_2, and vice versa. This can be used to tie up concepts from two different ontologies.
- *intersectionOf:* If a concept c_1 is tagged as *owl:intersectionOf* c_2 and c_3, which we denote as *intersectionOf*$[c_1] = \{c_2, c_3\}$, then all the instances of c_1 are also instances of c_2 and c_3, and all the common instances of c_2 and c_3 are also instances of c_1.
- *unionOf:* If a concept is c_1 tagged as the *owl:unionOf* c_2 and c_3, which we denote as *unionOf*$[c_1] = \{c_2, c_3\}$, then all the instances of c_1 should belong to at least one of c_2 and c_3, and all the instances of c_2 and c_3 are also instances of c_1.
- *complementOf:* If a concept c_1 is tagged as *owl:complementOf* c_2, which we denote as *complementOf*$[c_1] = c_2$, then they share no common instances, and c_1 and c_2 constitute the universe of discourse.

- *subPropertyOf:* If a property p_1 is tagged as *owl:subPropertyOf* p_2, which we denote as *subPropertyOf*$[p_1] = p_2$, then p_1 is a specialization of p_2.
- *equivalentProperty:* If a property p_1 is tagged as *owl:equivalentProperty* of p_2, which we denote as *equivalentProperty*$[p_1] = p_2$, then p_1 and p_2 should belong to the same or equivalent concepts.
- *TransitiveProperty:* If a property p is tagged as *owl:TransitiveProperty*, then for any i_1, i_2, and i_3, $i_1 : p = i_2 \wedge i_2 : p = i_3 \Rightarrow i_1 : p = i_3$. (Note: $i_1 : p = i_2$ means the value to the property p of instance i_1 is i_2.)
- *SymmetricProperty:* If a property p is tagged as *owl:SymmetricProperty*, then for any i_1 and i_2, $i_1 : p = i_2 \Leftrightarrow i_2 : p = i_1$ e.g. 'collaborateWith' between 'Person' and 'Person'.
- *FunctionalProperty:* If a property p is tagged as *owl:FunctionalProperty*, then for all i_1, i_2, and i_3, $i_1 : p = i_2 \wedge i_1 : p = i_3 \Rightarrow i_2 = i_3$. This means each instance should have one unique value for this property.
- *inverseOf:* If a property p_1 is tagged as *owl:inverseOf* p_2, then for all i_1 and i_2, $i_1 : p_1 = i_2 \Leftrightarrow i_2 : p_2 = i_1$. e.g. 'supervises' from concept 'Faculty' to concept 'Ph.D Student' vs. 'advisedBy' from concept 'Ph.D Student' to concept 'Faculty'.
- *InverseFunctionalProperty:* If a property p is tagged as *owl:InverseFunctionalProperty*, then for all i_1 and i_2, $i_1 : p = i_2 : p \Rightarrow i_1 = i_2$. e.g. the 'SSN' property of concept 'Person'.

Conceptual Modelling Patterns for Roles

Jordi Cabot[1,2] and Ruth Raventós[2]

[1]Estudis d'Informàtica i Multimèdia, Universitat Oberta de Catalunya
Av. Tibidabo, 39-43, E08035 Barcelona
jcabot@uoc.edu
[2] Departament Llenguatges i Sistemas Informàtics, Universitat Politècnica de Catalunya
Campus Nord, Edif. Omega, Jordi Girona 1-3, E08034 Barcelona
raventos@lsi.upc.edu

Abstract. Roles are meant to capture dynamic and temporal aspects of real-world objects. The role concept has been used with many semantic meanings: *dynamic class*, *aspect*, *perspective*, *interface* or *mode*. This paper identifies common semantics of different role models found in the literature. Moreover, it presents a set of conceptual modelling patterns for the role concept that include both the static and dynamic aspects of roles. In particular, we propose the *Role as Entity Types* conceptual modelling pattern to deal with the full role semantics. A conceptual modelling pattern is aimed at representing a specific structure of knowledge that appears in different domains. The use of these patterns eases the definition of roles in conceptual schemas. In addition, we describe the design of schemas defined by using the patterns in order to implement them in any object-oriented language.

1 Introduction

An accurate and complete conceptual modelling is essential for a correct development of information systems. Reusable conceptual schemas facilitate this difficult and time-consuming activity. The use of patterns is essential to increase the reusability in all stages of software development.

A pattern identifies a problem and provides the specification of a generic solution to that problem. The definition of patterns in conceptual modelling may be regarded in two different ways: conceptual modelling patterns and analysis patterns.

In this paper, we distinguish between a conceptual modelling pattern that is aimed at representing a specific structure of knowledge encountered in different domains (for instance the *MemberOf* relationship) and an analysis pattern that specifies a generic and domain-dependent knowledge required to develop an application for specific users (for instance a pattern for electronic marketplaces). Authors do not always make this distinction. For example, to Fowler, in [13], patterns correspond to our conceptual modelling patterns while to Fernandez and Yuan, in [12], patterns correspond to our definition of analysis patterns. For a further discussion on analysis patterns see Teniente in [43].

The goal of this paper is to propose a set of conceptual modelling patterns to facilitate the representation of roles in conceptual schemas.

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 158-184, 2006.

Although definitions of the role concept abound in the literature of conceptual modelling [3, 58, 13, 18, 35,37] a uniform and globally accepted definition is not given. To sum up, we could say that roles are meant to capture the dynamic and temporal aspects of real-world objects.

Roles appear very frequently in conceptual schemas. They are useful to model some dynamic situations from the real world that are not well represented just with the basic modelling language constructs, such as entity types that present different properties or behaviour depending on the context where they are used. For instance, the properties of a person when playing the role of student are different from those when playing the role of Employee. Moreover, when asking for his email address the answer depends on the role/s he is playing, it may be his personal email address, his student email address or his work email address.

Despite its importance, the possibilities that conceptual modelling languages offer to deal with roles are very limited (see, for example, what UML supports in [9] and [39]). In short, they consider roles just as the name of a participant in a relationship type.

We identify common and different semantics of role models found in the literature and characterize the patterns in terms of the features they cover. This allows the designer to choose the pattern best suitable for his needs. We also discuss the design and implementation of conceptual schemas that use these patterns to facilitate their implementation in object-oriented languages.

Of particular importance is our *Role as Entity Types* pattern, useful to represent roles when the full expressiveness of the role concept is needed. The pattern covers the most well-known role semantics. In contrast with previous approaches, one of its advantages is its simplicity, since roles and their evolution are represented with already existing constructs (entity types and constraints). Therefore, roles are easily integrated in conceptual schemas.

The rest of this paper is organized as follows: the next section presents a set of basic patterns. Section 3 proposes the *Roles as Entity Types* pattern. Section 4 comments related work and compares it with our proposals. Finally, conclusions and further work are presented.

2 Conceptual Modeling Patterns for Roles

The aim of this section is to define and compare a set of patterns to specify roles in conceptual schemas (CSs) by using just the standard constructs offered by conceptual modelling languages. Mosse in [24] and Fowler in [13] also propose a series of patterns for role representation. However, both of them discuss the patterns very informally and consider only a small subset of the role features we take into account.

One of the advantages of patterns explained in this section is that they are quite simple but, as a trade-off, their expressiveness is rather limited.

In order to describe the role patterns we adopt the template proposed by Geyer-Schulz and Hahsler in [16] to describe conceptual modelling patterns (called by the authors *analysis patterns*). They adopt a uniform and consistent format, in contrast to Fowler in [13] who uses a very free format for pattern writing. Geyer-Schulz and Hahsler stress that adhering to a structure for writing patterns is essential since

patterns are easier to teach, learn, compare with, write and use once the structure has been understood.

Their template preserves the typical context/problem/forces/solution structure of design patterns but adapted for the description of conceptual modelling patterns. The template includes the following sections: (1) Pattern Name. (2) Intent: what the pattern does and the problems it addresses. (3) Motivation: a scenario that illustrates the problem and how the pattern contributes to the solution in the specific scenario. (4) Forces and Context that should be resolved by the pattern. (5) Solution: description of all relevant structural and behavioural aspects of the pattern. (6) Consequences: how the pattern achieves its objectives and the existing trade-off. (7) Design and implementation: how the pattern can be realized in the design stage. (8) Known uses: examples of the pattern.

Obviously sections about intent, motivation and forces and context are common to all patterns for role representation. We first address these sections. Then, we define the solution that each pattern proposes, its consequences, its design and the known uses.

2.1 Intent

The intent is the representation of roles that entities play through their life span and the control of their evolution.

2.2 Motivation

The role concept appears frequently in many different domains of the real world, since we can find entity types in each domain that present some properties that evolve over time (see Papazoglou et al. in [32] and Jodlowski et al. in [20] for some example applications where roles are specially useful).

There is not a uniform and globally accepted definition of roles. The first commonly credited definition of roles in a data model goes back to the 70s when Bachman and Dayas proposed the *role data model* [3]. They defined a role as "a defined behaviour pattern that may be assumed by entities of different kinds". Since then, many other definitions and additional semantics have been proposed.

For instance, to Dahchour et al. in [8], the concept of role is "a generic relationship for conceptual modeling that relates a class of objects (e.g., people) and classes of roles (e.g., students, employees) for those objects. The relationship is meant to capture temporal aspects of real-world objects". To Papazoglou and Krämer in [31] a role "ascribes properties that possibly vary over time. The purpose of a role is to model different representation alternatives for the same object in terms of both structure and behavior".

To sum up, we could state that roles are useful to model the properties and behaviour of entities that evolve over time. The entity type *Person* is an illustrative example. During his or her life, a person may play different roles, for example he or she may become a student, an employee, a project manager, and so forth. Besides this, a person may have different properties and behaviours depending on the role or roles he/she is playing at a certain time.

For instance, let us consider the following scenario, which will serve as a recurrent example in the following pages: let Maria be a person (with a name, phone number, birthdate, country, age, sex and full address), who starts a degree (Maria plays the role of student). After some years of study, she registers to a second university program (Maria plays twice the role of student) and starts to work in a company (Maria plays the role of employee). In that company she may become a project manager (now, Maria through her employee role, plays the role of project manager). If in the future, Maria became a department manager, now Maria through her employee role would play a new role, department manager.

For each role we are interested in recording a set of properties specific for that role. As employee we are interested in: her employee number, category, company phone number, working status and the expiration date of her contract. As a project manager we are interested in information about the project she manages (the project name, the start date and the tasks it involves). Moreover, roles share properties with the main entity type. For instance, when considering Maria as employee we also want to know her name, even though the name is not an explicit property of employee.

Figure 1 shows the different relationships that are involved in the scenario introduced above. Note that, in this situation, if we ask for the value of a property of Maria the answer is not trivial because it depends on the role or roles she is playing. For instance, if we ask for her telephone number, the answer may be her personal number (since Maria is a person) or her company phone number (since Maria is an employee).

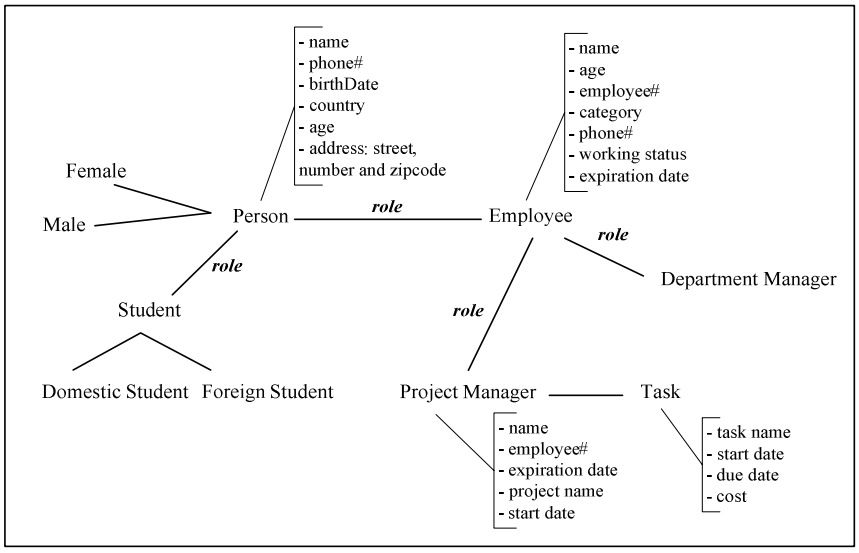


Fig. 1. Relationships involved in the example application

Despite its importance, the possibilities that conceptual modelling languages offer to deal with roles are very limited and cover only a very small part of their features. For instance, the ER model [6] considers roles as named places in a relationship; UML [29] considers that a role is an association end; in Description Logics [2] roles

only denote binary relationships between individuals; in Nijssen's information analysis method (NIAM) [25] and in its descendants as Object-Role Modeling (ORM) [19] each fact type (relationship) involves a number of roles, hence, roles are named placed in the relationships.

Taking into account the complexity of the notion of role and the lack of support for roles in present conceptual modelling languages, it is clear that patterns to define such a common construct are needed in conceptual modelling.

2.3 Forces and Context

To account for the complexity of the notion of roles and variety of semantics found in them, we describe below the set of features that roles should meet, most of which have been identified by Steinmann [38]. In our case, these features are the forces that influence and should be resolved by the pattern.

We describe them by using some examples related to the scenario introduced above:

1. Ownership. A role comes with its own properties and behaviour, i.e., an instance of *Employee* has its own properties which may be different from the ones of the entity type that plays such a role.
2. Dependency. An instance of a role is related to a unique instance of its entity type and its existence depends on the entity type to which it is associated to, i.e., it is not possible to have an instance of *Student* not related to an instance of *Person.* Although a fundamental characteristic, there exist proposals considering that a role should remain unconnected to any entity type, for instance, to model the salary of a vacant position for department manager [36]. This work does not address such possibility.
3. Diversity. An entity may play different roles simultaneously, i.e., an instance of *Person* may play simultaneously the role of *Student* and *Employee*.
4. Multiplicity. An instance of an entity type may play several instances of the same role type at the same time. For instance, a person that registers to more than one university has multiple instances of *Student* related to it.
5. Dynamicity. An entity may acquire and relinquish roles dynamically, i.e., a person may become a student, after some years become an employee, finish his/her studies, become a project manager, start another program and so forth.
6. Control. The sequence in which roles may be acquired and relinquished can be subject to restrictions, i.e., a person may not become an employee after he/she has retired or when he/she is also studying two degrees. Note that this does not prevents an employee from studying two degrees in the future. The restriction needs to be true only when hiring the employee.
7. Roles can play roles. This mirrors that an instance of *Person* can play the role of *Employee* and an instance of *Employee* can also play de role of *ProjectManager*.
8. Role identity. Each instance of a role has its own role identifier, which is different from that of all other instances of the entity to which is associated with. This solves the so-called *counting problem* introduced by Wieringa et al in [44]. It refers to the fact that we need to distinguish the instances of the roles from the instances of the entity types that play them. For example, if we want to count the number of people that are students in a university (i.e., every person who is registered to at least a

program in such university), the total number is less than the number of registered students in such university (in this case a person is counted twice if he or she is registered at two programs).

9. Adoption. Roles do not inherit properties from their entity types. Instead, instances of roles have access to some properties of their corresponding entities i.e., *Student* may adopt *name* and *address* properties of *Person* but neither *religion* nor *marital status* properties. Therefore, the *Student* role cannot use the last two referred properties.
10. Relationship independency. A role is meaningful even out of the context of a relationship. E.g., a person may play the role of student or employee without necessarily being tied to a university or a company respectively.
11. Common role for unrelated types. A set of unrelated types may play the same role [3]. For instance, a project manager may be the role of both employee and external service provider.
12. Sharing structure and behavior. Roles may have some common structure and behavior. For instance, the constraint that Maria may not become an employee before she is 16 years old should apply also to project manager.

2.4 *Roles as Participant Names* Pattern

2.4.1 Solution

A role is merely represented as a name assigned to the participation of an entity type in a relationship type. Although a rather limited representation, it is what conceptual modelling languages usually consider.

Figure 2 models the running example when considering roles as the participant names of a relationship type. Note that *ProjectManager* and *DepartmentManager* do not appear in the conceptual schema, since, in this approach, a role cannot play other roles. Students can neither be classified in domestic or foreign students.

2.4.2 Consequences

Only a small subset of the previous features is covered by the pattern. We justify each of them as follows:

2. Dependency: a relationship in a relationship type is always related to an instance of the participant entity types. For example, a student may only be defined if an instance of the relationship type relating *Person* and *University* exists.

3. Diversity: an entity type can participate in many different relationship types.

4. Multiplicity: this feature is partially covered since an entity type can play several times the same role (i.e., it may participate in the same relationship type) by providing the value of the multiplicy of the participant greater than one. However, two relationships of a relationship type may not exist between the same participants (i.e a person cannot study twice in the same University).

5. Dynamicity: relationships can be added and deleted at any time.

Due to the limited expressiveness of participant names, the pattern does not support the following features:

1. Ownership: the participation of an entity type in a relationship type does not have properties nor behaviour

6. Control: we cannot attach constraints to the participation apart from multiplicity constraints.

7. Roles can play roles: only entity types can participate in other relationship types. Therefore, we cannot define that the role *Employee* has the role of *ProjectManager*.

8. Role identity: a relationship type has no identity, it is identified through its participants.

9. Adoption: by navigating through the relationship type where the role is defined we can access to all properties of the entity type. We cannot restrict the access to the personal number of person when navigating from employee.

10. Relationship independency: obviously, roles represented as participant names only make sense in the context of a relationship type.

11. Common role for unrelated types: the same role cannot be used in different relationship types.

12. Different roles may share structure and behaviour: since roles as participant names have no structure or behaviour they cannot share it.

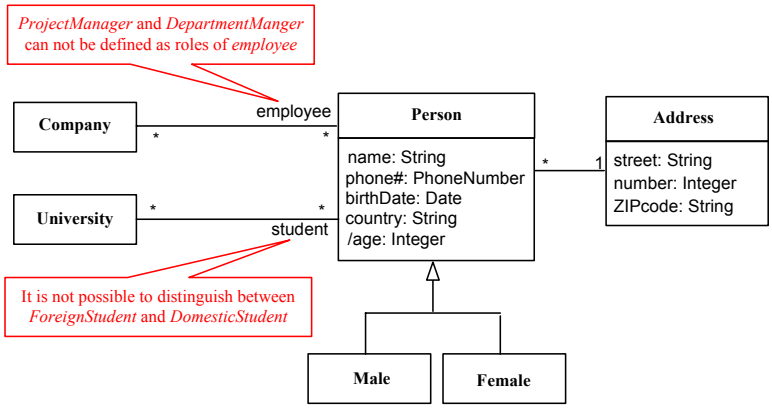


Fig. 2. Example of the *Roles as Participant Names* Pattern

2.4.3 Design and Implementation

The design and implementation of roles defined following this pattern in object oriented languages is straightforward. Roles are simply transformed into attributes of the entity type. The multiplicity and type of these attributes is obtained from the definition of the relationship type including the role.

2.4.4 Known Uses

This pattern is useful when we want to qualify the participation of an entity type in a relationship type but we are not interested in defining properties or behaviour of that participation. For instance, if we are only interested on a person becoming a student without worrying about his or her properties as student, like the degree he/she is studying.

2.5 *Roles as Subtypes* Pattern

2.5.1 Solution

Role entity types are represented as subtypes of the entity type playing them. For instance, *Student* and *Employee* would appear as subtypes of *Person*. Quite obviously, such a solution requires dynamic and multiple classification, since a person can change his/her role and play several roles simultaneously.

As an example Figure 3 shows the running example when considering roles as subtypes.

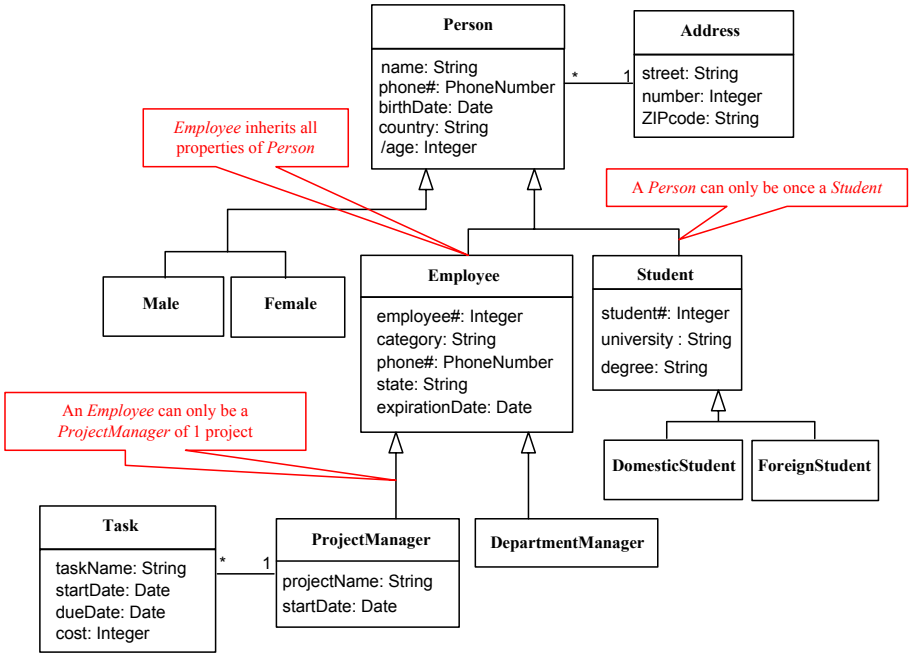


Fig. 3. Example of the *Roles as Subtypes* Pattern

2.5.2 Consequences

The following list of role features is supported for this pattern:

1. Ownership: as a subtype, the role can define its own properties and behaviour.
2. Dependency: all subtype instances are instances of their supertypes.
3. Diversity: by assuming multiple classification, an entity may appear in several role subtypes (i.e., a person may be an employee and a student at the same time).
5. Dynamicity: by assuming dynamic classification role instances can be added and removed from the subyptes at any time.
7. Roles can play roles: this feature is simulated by defining a role type as a subtype of another role type. For instance, *ProjectManager* is defined as a subtype of *Employee*.
10. Relationship independency: as entity types, roles have their own existence independent from any relationship type.

12. Different roles may share structure and behaviour: by assuming multiple inheritance, two roles may share structure and behaviour if they inherit them from a common supertype.

On the other hand some important features are not covered by the pattern:

4. Multiplicity: since Employee is a subtype of *Person*, we cannot define that a person plays simultaneously twice the role of *Employee*.

6. Control: as entity types, we can define constraints on role types. However, the constraints are static and thus must be satisfied at any time. We cannot define constraints that only apply when the role is inserted or deleted.

8. Role identity: as a subtype, the identifier of the role instance is the same as the identifier of the entity type instance. Therefore, the counting problem mentioned before is not solved. In fact, the counting problem is not solved either because multiplicity cannot be addressed with this pattern.

9. Adoption: with specialization we cannot restrict which attributes are adopted by the roles because they inherit all the attributes and relationships of their supertype (i.e. *Employee* inherits all the *Person*'s attributes).

11. Common role for unrelated types: this could be simulated using multiple inheritance. However, in such a case the role would inherit the properties of all its supertypes which is not what we meant. For instance, if we need *ProjectManager* to be the role of both *Employee* and *ExternalServiceProvider*, we could define *ProjectManager* as a subtype of both *Employee* and *ExternalServiceProvider* but then *ProjectManager* would inherit the properties of *Employee* as well as the properties of *ExternalServiceProvider*.

2.5.3 Design

The design and implementation of roles defined following this pattern in object oriented languages is not straightforward, since, in general, they do not support multiple nor dynamic classification. Fowler [13] and Pelechano et al. [34] comment some design patterns to cope with these issues.

2.5.4 Known Uses

Probably, this is the most intuitive way to represent roles. Provided that the previous limitations are not a problem in the specific application, this pattern represents a good combination between simplicity and expressiveness. However, we should also take into account that the design of this pattern is not trivial.

2.6 *Roles as Interfaces* Pattern

2.6.1 Solution

Roles are represented as interfaces. An interface represents a declaration of a set of coherent public features and obligations [29]. To define that an entity type plays a certain role we must specify that the type realizes (i.e., implements) the interface corresponding to that role.

The implementation relationship signifies that the entity type conforms to the contract specified by the interface (i.e., the entity type provides an operation for every operation defined in the interface and a property for each feature).

Figure 4 shows the running example when considering roles as interfaces. For instance, to specify that *Person* plays the role of *Employee*, *Employee* is defined as an interface that is implemented by *Person*. Note that this obliges *Person* to contain all the attributes defined in *Employee*, since an entity needs to contain all attributes of all its interfaces. Students are not classified in domestic or foreign since the notion of subtypes cannot be applied to interfaces. We could define them as interfaces extending the *Student* interface but the resulting semantics would be different (for example, the state of a person being a student would be completely independent from the state of a person being a domestic student, since interfaces inherit only the specification of extended interfaces but not its state).

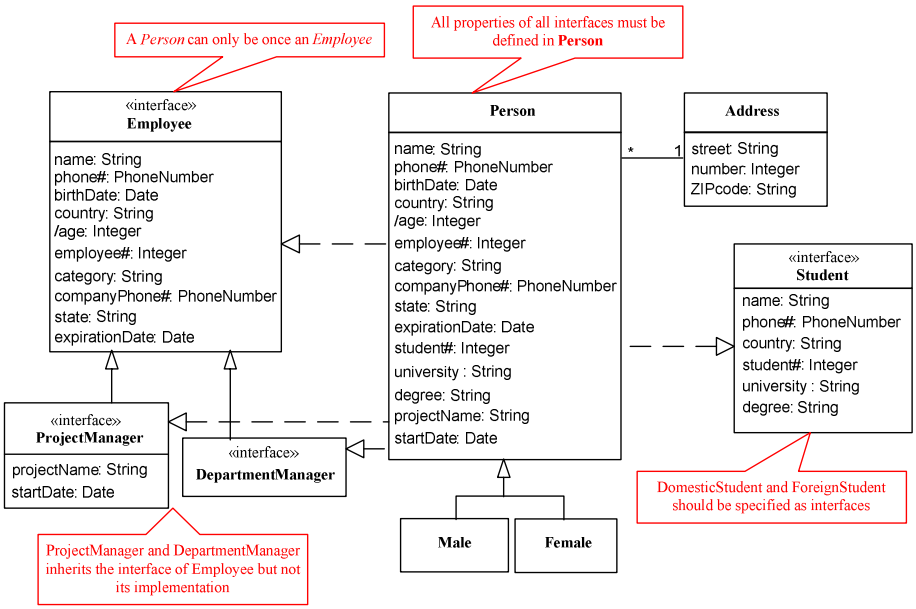


Fig. 4. Example of the *Roles as Interfaces* Pattern

2.6.2 Consequences

The features covered for this pattern are:

2. Dependency: the role does not exist as a separate instance from the instance of the entity type.
3. Diversity: an entity type can implement several interfaces.
10. Relationship independency: as interfaces, roles do not depend on any relationship of the entity type.
11. Common role for unrelated types: interfaces can be implemeted by many unrelated types.
12. Different roles may share structure and behaviour: this can be simulated by defining inheritance relationships between interfaces.

The features not covered for this pattern are:
1. Ownership: even though an interface defines its own properties, it does not have an independent state from the state of the instances of the entity type implementing them.
4. Multiplicity: an entity type cannot implement twice an interface, and thus, a entity type can only play once the same role.
5. Dynamicity: every instance of the entity type plays all roles corresponding to the interfaces implemented by the type during its whole life. All interfaces are acquired when the instance is created.
6. Control: since all roles are acquired at the beginning, this property does not make sense.
7. Roles can play roles: interfaces cannot implement other interfaces. For instance, *Employee* cannot implement the *ProjectManager* interface. When defining *ProjectManager* we can extend the specification of *Employee* but the semantics are the same as if two independent interfaces were defined since it is *Person* and not *Employee* who implements the interface *ProjectManager*.
8. Role identity: interfaces do not have instances.
9. Adoption: interfaces do not adopt any of the properties of the entity type. On the contrary, each interface defines its independent set of properties that must be implemented by the entity type.

2.6.3 Design and Implementation

One of the advantages of this pattern is that the design and implementation of roles as interfaces is a direct translation from the conceptual schema since most object oriented languages perfectly support the notion of interfaces.

2.6.4 Known Uses

Despite its major drawbacks, the main usage of this pattern is the definition of conceptual schemas where roles must be played by unrelated entity types. Also, as Fowler in [13] proposes, the pattern can be used to integrate seamlessly the specification of CS with its implementation when the implementation language does not support multiple and dynamic classification.

2.7 *Roles as Reified Entity Types* Pattern

2.7.1 Solution

Roles are represented as reified entity types of a relationship type between the entity type playing the role and another entity type (called the *companion* entity type). For instance, the student role can be defined as the reified entity type of the relationship between *Person* and *University* (although a more appropriate name for the reified type could be enrolment). Note that it is not clear whether *Student* is a role of *Person* or *University*.

Choosing the right companion entity type is not easy. In fact, sometimes it does not exist and must be created specifically to be able to define the role. For instance, to define the student role we have to include in the CS the *University* type. Note that, depending on the purpose of the CS, we may not be interested in recording data about the universities where people study. In such a case, a plain string recording the name

of the university as an attribute of the student could be enough (as in the roles as subtypes approach).

On the other hand, the election may impose some constraints to the role since there could not exist two relationships of a relationship type between the same participants. In the previous example, choosing *University* as companion entity type prevents a person to study twice in the same university.

Figure 5 shows the running example when considering roles as reified entity types.

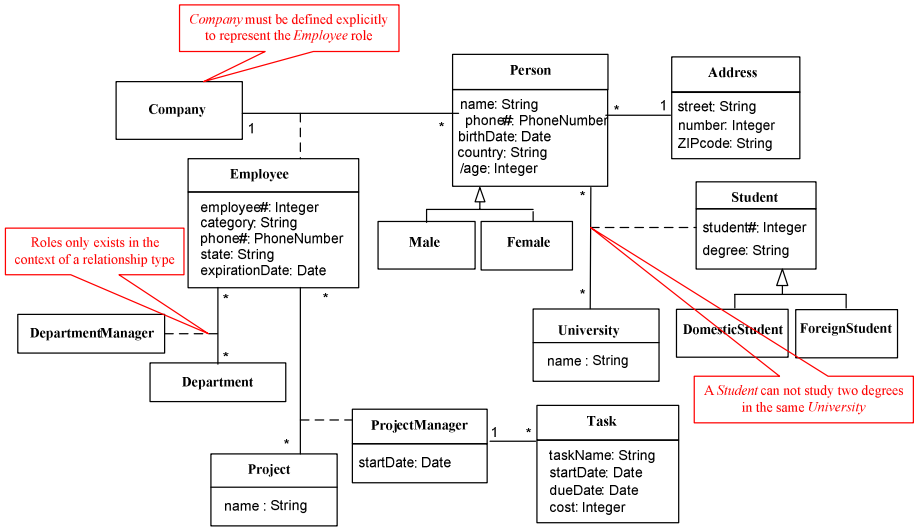


Fig. 5. Example of the *Roles as Reified Entity Types* Pattern.

2.7.2 Consequences

The features covered for this pattern are:

1. Ownership: as an entity type, the role can define its own properties and behaviour.
2. Dependency: as a reified entity type, an instance of the role type can only exist when the participants of the relationship also exist.
3. Diversity: an entity type may participate in several relationships.
5. Dynamicity: instances of reified entity types can be inserted and deleted at any time.
7. Roles can play roles: a reified entity type can be a participant in other relationship types. For instance, the reified entity type Employee is one of the participants of the relationship defined to specify the role DepartmentManager
8. Role identity: the role has its own identity as a reified entity type, different from the identity of the entity type.
12. Different roles may share structure and behaviour: this can be simulated by means of generalization and specialization relationships between the reified types.

On the other hand, the roles do not cover:

4. Multiplicity: it is restricted by the choosen companion class.

6. Control: as entity types, we can define constraints on role types. However, we cannot define constraints that only apply when the role is inserted or deleted.
9. Adoption: we cannot restrict which attributes are adopted by the roles since, by default, all properties can be acceded.
10. Relationship independency: as reified entity types, roles can only exist in the context of a relationship type.
11. Common role for unrelated types: two different relationship types cannot share the same reified entity type.

2.7.3 Design and Implementation

The reified entity types are designed and implemented as classes with a set of single-valued additional attributes, which refer to each of the participants of the relationship type.

2.7.4 Known Uses

This pattern can be regarded as an extension of the previous *roles as participant names* pattern. Therefore, this pattern is useful to qualify the participation of an entity type in a relationship type. Moreover, by using the pattern we can define properties and behaviour of that participation.

As a disadvantage, apart from the features not covered by the pattern, the use of this pattern may increase the complexity of the resulting CS since for each role we must create a relationship type, a reified relationship type and, when not present in the CS, the companion entity type.

3 *Roles as EntityTypes* Pattern

Previous patterns cover only a subset of the required role features. We propose to use our *Role as Entity types* pattern when the full expressivity of the role concept is needed. This pattern is an updated and extended version of the one presented in [5].

The basic aim of the pattern is to represent roles as an entity type related to the entity type playing the role by means of a special kind of relationship type, the *RoleOf* relationship type.

Next sections discuss in depth the pattern solution, its consequences, design and known uses.

3.1 Solution

We divide the solution of our role pattern in two subsections. The first one deals with the structural aspects of roles while the second one deals with their evolution.

3.1.1 Structural Aspects of Roles

We believe there is not a fundamental difference between roles and entity types since roles have their own properties and identity. Therefore, when using this pattern we represent roles as entity types with their own attributes, relationships and

generalisation/specialisation hierarchies. For practical reasons, we call *role entity* types (or simply role if the context is clear) the entity types that represent roles and *natural entity types*[1] (or simply entity types) the entity types that may play those roles.

We define the relationship between a role entity type and its natural entity type by means of a new generic relationship type, the *RoleOf* relationship. A generic relationship type is a relationship type that may have several realizations in a domain [27]. Each realization of this generic relationship type is a specific relationship type relating a natural entity type to a role entity type to indicate that the natural entity type may play the role represented by the role entity type.

In the relationship type we also specify the properties (attributes and associations) of the natural entity type that are adopted by the role entity type.

Note that, since roles may play other roles, the same entity type may appear as a role entity type in a *RoleOf* relationship and as a natural entity type in a different *RoleOf* relationship.

Although this representation may be expressed in many conceptual modelling languages, in this work, we only adapt it to UML [29] and OCL [28]. See Olivé in [27] for a general discussion about the implementation of generic relationship types in conceptual schemas.

To represent the *RoleOf* relationship we use the standard extension mechanisms provided by UML, such as stereotypes, tags and constraints. Stereotypes allow us to define (virtual) new subclasses of metaclasses by adding some additional semantics and properties (tags) to its base entity type. A stereotype may also define additional constraints. It is worth to notice that merely using these lightweight[2] extension mechanisms ensures that the pattern can be easily integrated in UML conceptual schemas.

We represent the *RoleOf* generic relationship type by means of the «RoleOf» stereotype. The base class of the stereotype is the *Association* metaclass, which represents association relationships among classes. Each specific relationship type is labelled with this stereotype. The stereotype also permits the definition of the properties[3] the role adopts from the natural entity type. They are represented with a multivalued attribute, called *adoptedProperties*. Figure 6 shows the definition of the «RoleOf» stereotype.

Fig. 6. Definition of the RoleOf stereotype

As an example, figure 7 shows the running example introduced in section 2.2 using this pattern. Note that all roles are represented as entity types with a «RoleOf»

[1] The *natural entity type* of a role relationship has sometimes been called *object class* [8, 44] *ObjectWithRoles* [17], *natural type* [18, 38], *base class* [7, 31], *entity type* [1], *entity class* [3], *base role* [33], or *core object* [4].

[2] In contrast with heavyweight mechanisms that involve the creation of new metaentity types.

[3] A property in UML 2.0 [29] represents both the attributes and associations of an entity type.

relationship type relating the role with is entity type. For instance, the role *Student* is represented as an entity type related to *Person* through a «RoleOf» relationship type. In the relationship type it is also indicated that student adopts the properties: *name*, *address*, *phone#* and *country* from *Person* (its natural entity type). *Employee* participates in three «RoleOf» relationship types, one as a role of *Person* and the other ones as a natural entity type playing the role of *ProjectManager* and *DepartmentManager*.

The stereotyped operations, also shown in the figure, will be further described in the following section.

To complete the definition of the static aspects of roles we must attach some constraints to the «RoleOf» stereotype in order to control the correctness of its use. There already exist proposals to automatically ensure the consistency of a conceptual schema according to the conceptual modelling language metamodel extended with stereotypes [41].

The constraints are the following:

- A stereotyped «RoleOf» association is a binary association with multiplicity '1' and settability *readOnly* in an association end.
- Each value of the *adoptedProperty* tag must coincide with the name of a property of the natural entity type.
- A role entity type can only be related throughout a *RoleOf* relationship to at most a natural entity type.
- No cycles of roles are permitted. A role entity type may not be related throughout a direct or indirect *RoleOf* relationship to itself.

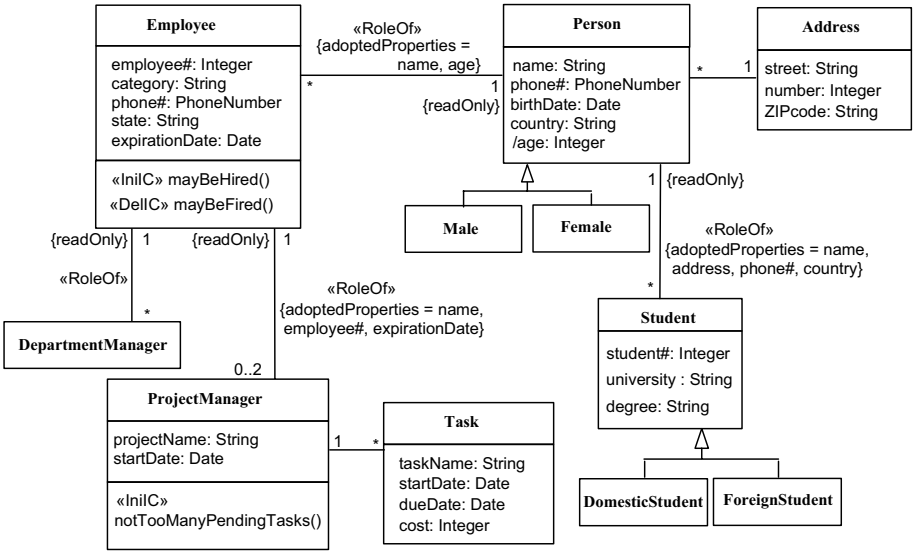


Fig. 7. Example of *RoleOf* relationships in the UML

Properties adopted by the role from its natural entity type may be considered as implicit properties of the role entity type. Nevertheless, in order to facilitate the use of these adopted properties (for instance, when writing OCL expressions) we may need

to include them explicitly in the role entity type. In this case, we add an extra property in the role entity type for each adopted property. These extra properties are labeled with the «adopted» stereotype to distinguish them from the own properties of the role entity type. In addition, they are derived. Their derivation rule always follows the general form:

```
context RoleEntityType::adoptedPropertyX: Type
derive: naturalEntityType.propertyX
```

Note that, to facilitate the work of designers, these added properties can be automatically generated.

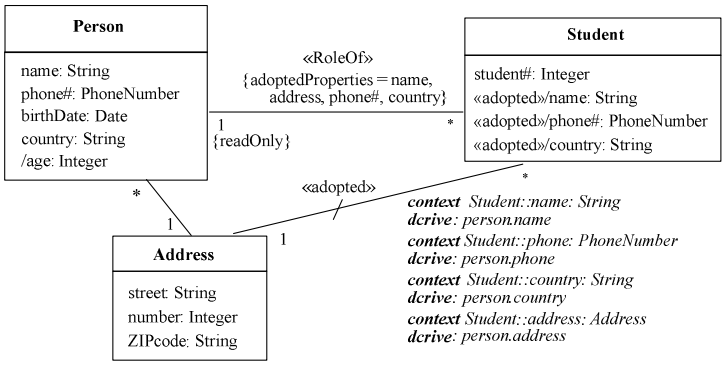


Fig. 8. Example of the *Student* role entity type

Figure 8 extends a subset of the previous example illustrating the *Student* role entity type including its adopted properties.

Likewise, when the CS includes operations role entity types can also adopt said operations. For instance, if we express the *age* derived property of the *Person* entity type as a query operation (Figure 9) we may be interested in defining that the *age* operation can also be executed over employees (indicated in the *adoptedOperation* tag). Operations are adopted following a delegation mechanism, i.e., the body of the adopted operation delegates the execution to the original operation.

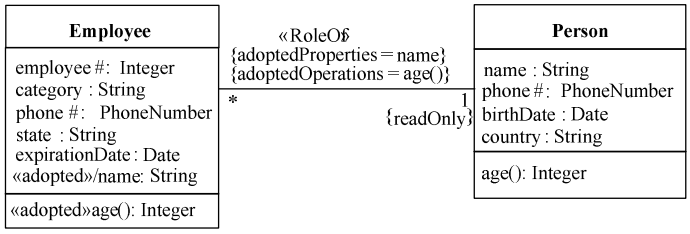


Fig. 9. Adoption of the *age* operation

For instance, the adopted *age* operation in the *Employee* role entity type would be defined as:

```
context Employee::Age():Natural
body: person.age()
```

3.1.2 Role Acquisition and Relinquishment

So far, we have introduced a representation of the static part of the *Roles as Entity Types* Pattern. Nevertheless, this is not enough since role instances may be added or removed dynamically from an entity during its lifecycle and this addition or removal may be subjected to user-defined restrictions.

Since roles are represented as entity types we may define constraints on roles in the same way we define constraints on entity types. Some of the constraints are inherent to our role representation (for example, that a person must play the role of *Employee* to play the role of *ProjectManager*, is already enforced by the schema). Other restrictions may be expressed by means of the predefined constraints of the UML. For example, to restrict that an *Employee* cannot play more than twice the *ProjectManager* role simultaneously, it is enough to define a cardinality constraint in the relationship type. The definition of the rest of constraints requires the use of a general-purpose language, commonly OCL in the case of UML. For instance, we could specify OCL constraints to control that:

- A *Person* can only play the role of *Employee* if he/she is between 18 and 65 years old:

```
context Employee
inv: self.age>=18 and self.age<=65
```

- Any task of a ProjectManager must finish before his contract expires

```
context Task
inv:self.dueDate<self.projectManager.expirationDate
```

These OCL constraints are static, and thus, the role instances must satisfy them at any time. However, many of the restrictions that may be involved in the evolution of roles only apply at particular times, particularly they only need to be satisfied when the role is acquired or when it is relinquished. To specify such constraints we use the notion of creation-time constraints defined by Olivé in [26] and, in a similar way, we define the deletion-time constraints.

Creation-time constraints must hold when the instances of some entity type are created (in our case when the role is created). Deletion-time constraints must hold when the instances of some entity type are deleted (in our case when the role is deleted). These constraints are represented as operations, also called *constraint* operations, attached to the entity types and identified by a special stereotype. The creation-time constraint operations are marked with the stereotype «IniIC». We define the stereotype «DelIC» for the deletion-time constraint operations.

These operations return a boolean that must be true to indicate that the constraint is satisfied. If the operation returns false (i.e., the constraint is not satisfied) then the creation or deletion event of the role is not accomplished. When appropriate, the operations are automatically executed by the information system.

As an example, the constraints in Figure 7 can be defined as follows:

- A person cannot become an employee if he/she is studying two university programs simultaneously. Note that this does not imply that a person that is already an employee may not apply for two degrees.

```
context Employee :: mayBeHired () : Boolean
  body:  self.person.student->size()<2
```

- An employee may not be fired if he or she is in maternity leave.

```
context Employee :: mayBeFired () : Boolean
  body: self.workingStatus<>'MaternityLeave'
```

- An employee may not become a new project manager if he/she still holds more than ten pending tasks.

```
context ProjectManager::notTooManyPendingTasks():
                                                Boolean
 body : self.employee.projectManager.tasks ->
            select(dueDate>Today)->size()<=10
```

3.2 Consequences

The pattern achieves most of the role features outlined before:

1. Ownership. As roles are represented as entity types, they may have their own properties.

2. Dependency. The cardinality '1' with the tag {readOnly} ensures that all role instances depend on a unique instance of the natural entity type.

3. Diversity. Entity types may have many *RoleOf* relationships.

4. Multiplicity. This is obtained by defining a cardinality greater than one in the *RoleOf* relationship.

5. Dinamicity. Entities are related to their roles through an association. Thus, an entity may acquire or retract instances of a role at any time.

6. Control. The sequence in which roles may be acquired and relinquished can be subjected to restrictions, including creation-time and deletion-time constraints.

7. Roles can play roles. Roles are represented by ordinary entity types. So, they can be participants of a *RoleOf* relationship.

8. Role identity. As roles are represented as entity types, their instances have their own identifier.

9. Adoption. The *adoptedProperty* tag of the *RoleOf* relationship allows the definition of the adopted properties.

10. Relationship independency. As entity types, roles are independent from relationship types.

12. Different roles may share structure and behavior. As entity types we can define generalization relationships between roles.

As a trade-off, our pattern does not directly supports the remaining feature (11. Common role for unrelated types). However it can be easily represented. For instance, if we need *ProjectManager* to be the role of both *Employee* and

ExternalServiceProvider, we could define a common supertype for *Employee* (understood as *InternalServiceProvider*) and *ExternalServiceProvider*, called *ServiceProvider*, which plays the role of *ProjectManager*.

An alternative is to define two different RoleOf relationship types, one between *ProjectManager* and *Employee* and another one between *ProjectManager* and *ExternalServiceProvider*. Both relationship types are specified with a *xor* constraint to prevent a project manager being an employee and an external service provider at the same time. *ServiceProvider* is not needed. On the other hand, the management of the adopted properties is more complex.

Figure 10 shows the example using both alternatives.

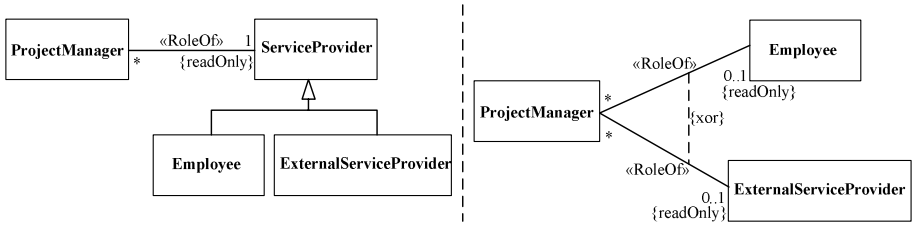


Fig. 10. Representing *common role for unrelated types* feature

3.3 Design and Implementation

There are some design patterns useful for designing and implementing roles in object oriented languages [13]. However, most of them are unable to deal with our proposed role semantics completely. A well-known pattern close to our role defined semantics is the *Role Object Pattern* [4]. This pattern is especially well suited for role implementation when roles are deemed as a specialization (or a kind of specialization) of its entity type (see Pelechano et al. in [33] as an example).

Nevertheless, this pattern is not entirely appropriate for designing our conceptual modelling pattern. We encounter two main problems in the *Role Object Pattern*. First, it uses a common superclass for all the roles of the entity type. In our approach, roles are independent entity types with not necessarily any common properties that justify this superclass. Second, all the roles are forced to have the same inherited properties; it is not possible to define different adopted properties for each role.

This is the reason why we advocate here for an adapted version of this pattern that takes into account our complete role semantics, including the adoption mechanism and the creation-time and deletion-time constraints.

Given a natural entity type and the set of its roles, we create a class for the natural entity type and a class for each role. We create a different relationship between the natural entity type and each of its roles. This relationship will be used to navigate from the natural entity type to its roles and vice versa. We add to the natural entity type two new operations *addRole* and *deleteRole* in charge of adding (deleting) roles to the natural entity after checking the creation-time (deletion-time) constraints. We could also add other useful operations when dealing with roles, such as *hasRole* (to check whether an entity plays a role) or *getRole* (to obtain a role played by the entity).

The problem of the design of the adopted properties may be regarded as the same problem as designing derived information. In general, from a design and/or implementation point of view, there are two different approaches to deal with derived information. The attributes may be computed if they are calculated by means of an operation or may be materialized if they are explicitly stored in the class. In this case, for each adopted property we add an extra operation to the role class that returns the value of the property of the natural entity type. The operation accesses the property of the natural entity type navigating through the relationship.

Figure 11 summarizes our proposal.

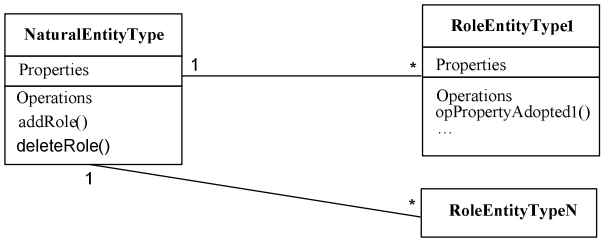


Fig. 11. Summarized class diagram of the design

In figure 12, we apply the proposed design pattern to a part of the conceptual schema of figure 7. Note that *Employee* is both a role for the *Person* entity type and a natural entity type for the *ProjectManager* role, and thus, it presents both a reference to *Person* (as a role entity type) and the operations *addRole* and *deleteRole* (as a natural entity type). Additionally, *Employee* includes also the *name* and *age* operations to get this information from *Person*.

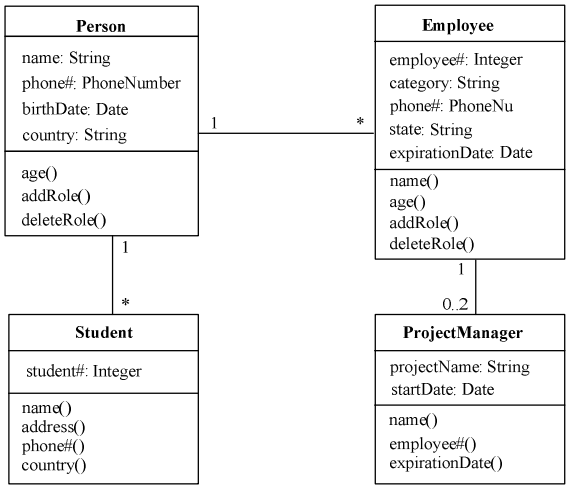


Fig. 12. Example of an application of the design

This structure can be directly implemented in any common object-oriented language. An example of the implementation in the Java Language can be found in Appendix A.

3.4 Known Uses

This pattern should be used to represent the full expressiveness of roles in conceptual schemas.

In contrast to other approaches where the complexity of the CS is really increased when using roles due to the special construct needed to represent them, our pattern allows a plain integration of roles in CS. Therefore, there are no trade-offs that prevent from applying the pattern whenever it may be useful.

4 Related Work

The role concept has been widely addressed in the literature. Although all approaches present their own characteristics, they can be grouped in four basic approaches to represent roles: 1 - roles as the name of a participant in a relationship type [6, 10, 19, 29]; 2 - roles as a sort of subtypes or supertypes of the natural entity types [1, 31, 37]; 3 - roles as interfaces [21, 23, 40]; and 4 - roles as a distinct element from an entity type but coupled to it [3, 7, 8, 11, 17, 20, 22, 35, 38, 42 , 44, 45].

The first three families are similar to our *Roles as Participant Names*, *Roles as Subtypes* and *Roles as Interfaces* patterns, respectively. Therefore, the major advantages and drawbacks of these three groups are mainly the same commented for the corresponding patterns in Section 2.

In this section we focus on the comparison between our *Roles as Entity types* pattern and the other approaches also considering a role as a distinct element from an entity type but coupled to it.

Table 1 compares the most representative approaches in terms of the role features they can handle. Most of these approaches use different semantics from the ones presented in this paper or are unable to handle the full role semantics.

All approaches shown in the table fulfil the ownership, dependency, diversity and dynamicity features.

However, few approaches consider roles with their other identity (thus, solving the counting problem). Some of them propose alternative techniques to distinguish between different role instances of the same natural instance. For instance, Gottlob et al [17] mixes the identifier of the natural instance with the value of a special attribute, called qualifier and Wong et al. [45] uses the state of the role instance.

Even more critical is the support of the control and adoption features.

Most of them do not handle the control feature. Some allow the definition of static constraints. Additionally, Pernici [35] and Wieringa [44] take into account the sequence in which roles are acquired and relinquinshed, but do not consider the definition of additional restrictions over the sequence (as our creation-time and deletion-time constraints).

Table 1. Comparison of role representation approaches

Features \ Approaches	Bachman and Daya [3]	Chu and Zhang [7]	Dahchour, Priotte and Zimányi [8]	Fan, Barker, Porter and Clark [11]	Gottlob, Schrefl and Röck [17]	Jodlowski, Habela, Podzien and Subieta [20]	Kristensen [22]	Pernici [35]	Steimann (DKE) [38]	Thalheim [42]	Wieringa, de Jonge and Spruit [44]	Wong, Chau and Lochovsky [45]
Ownership	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Dependency	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Diversity	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Multiplicity	✘	✘	✔	✘	✔	✔	✔	✔	✔	✔	✔	✔
Dynamicity	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Control	✘	✘	✔	✘	✘	✘	✘	~	✘	✘	~	✘
Roles can play roles	✘	✔	✔	✘	✔	✔	✔	✘	✘	✔	✔	✔
Identity	✘	✔	✔	✘	~	✔	✘	✘	✘	✔	✔	~
Adoption	✔	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Relantionship Independency	✔	✘	✔	✘	✔	✔	✔	✔	✘	✔	✔	✔
Common role for unrelated types	✔	✘	✘	✔	✘	✘	✔	✘	✔	✘	✘	✔
Sharing structure and behavior	✘	✔	✔	✔	✔	✔	✔	✘	✔	✘	✔	✔

Adoption is neither supported. Most approaches define that roles can (or cannot) access all the properties of the natural entity type but they do not provide a mechanism to indicate which properties may be adopted.

Our alternative suggesting roles as separated entity types fulfils the role semantics.

We believe one of the main advantages of our *Roles as Entity Types* pattern over previous approaches is that we handle the complexity of role semantics in a very simple manner since we represent roles and their evolution with already existing elements (entity types and constraints) without adding completely new language constructs (as done by several of the previous approaches). Therefore, the designer can easily use the patterns to specify roles in conceptual schemas. In addition, our pattern describes a representation of roles in the standard UML, and thus, the pattern can be directly incorporated into current UML CASE tools.

We would also like to remark that our pattern is complete and feasible in the sense that it includes the design and the implementation of the pattern, in contrast to most of previous approaches that do not state how this could be achieved.

5 Conclusions

This paper identifies the most important features of roles and presents a set of conceptual modelling patterns to facilitate the representation of roles in conceptual

schemas. Each pattern is characterized in terms of the features it covers. We also review their design and implementation.

Roles as Entity Types pattern is of special importance. We propose using this pattern when we need to represent the full expressivity of roles in CSs. We have adapted the pattern to the UML conceptual schemas. To our knowledge, ours is the first UML standard extension that defines roles in conceptual schemas specified with this language. Because of its simplicity, the pattern can be easily implemented in any CASE tool in order to allow designers the use of the role concept.

The pattern includes the static aspects of roles as well as their evolution. We define roles as entity types (role entity types) related to natural entity types by means of a generic *RoleOf* relationship type that includes the adoption of properties from the natural entity types by the role entity types. We have extended UML by means of the *«RoleOf»* stereotype to be able to represent such kind of relationships. To specify the role evolution, we use two special kinds of constraints: creation-time constraints and deletion-time constraints.

It would be interesting to semi-automate the selection and application of these patterns in CS. Given the set of role features the designer needs to take into account, the CS and a set of roles, we could integrate the roles in the CS by using the simplest pattern covering the required role features. Additionally, given the CS with the roles included, we would like to automate its design and implementation by means of an application that, given a conceptual schema (for instance, represented in XMI [30]), generates automatically the corresponding classes in the target object oriented language. These are directions in which we plan to continue our work.

Acknowledgements

We would like to thank Jordi Conesa, Dolors Costal, Xavier de Palol, Cristina Gómez, Antoni Olivé, Anna Queralt, Maria Ribera Sancho, Ernest Teniente for their many useful comments in the preparation of this paper. This work has been partially supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

1. A. Albano, R. Bergamini, G. Ghelli, R. Orsini, “An Object Data Model with Roles”, Proceedings of the 19th Very Large Data Bases (VLDB) Conference. Morgan Kaufmann, 1993, pp. 39-51.
2. F. Baader, W. Nutt, “Basic Description Logics”, In: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications. CambridgeUniversity Press, 2003
3. C.W. Bachman, M. Daya. “The Role Concept in Data Models”, Proceedings of the 3rd Very Large Data Bases (VLDB) Conference, 1977, pp. 464-476.
4. D. Bäumer, D. Riehle, W. Wiberski, M. Wulf. “The Role Object Pattern”, Proceedings of Pattern Languages of Programming (PLoP) Conference 1997. Technical Report WUCS-97-34. Washington University Dept.

5. J. Cabot, R. Raventos, "Roles as Entity Types: A Conceptual Modelling Pattern", Proceedings of the 23rd International Conferece on Conceptual Modeling (ER'04), LNCS 3288, Springer, pp. 69-82
6. P.P. Chen. "The entity-relationship model: Towards a unified view of data, ACM Transactions on DatabaseSystems 1 (1), 1976, pp. 9-36.
7. W.W. Chu, G. Zhang, "Associations and Roles in Object-oriented Modeling", Proceedings of the 16th International Conferece on Conceptual Modeling (ER'97), LNCS 1331, Springer, pp. 257-270.
8. M. Dahchour, A. Pirotte, E. Zimányi, "A role model and its metaclass implementation", Information Systems, 29 (2004) pp. 235-270.
9. R.Depke, G.Engels, J.M. Küster, "On the Integration of Roles in the UML", Technical Report No. 214, University of Paderborn, August 2000.
10. E. Falkenberg, "Concepts for modelling information", Proceedings of the IFIP Conference on Modelling in Data Base Management Systems, North-Holland, Amsterdam; 1976, pp. 95-109.
11. J. Fan, K. Barker, B.W. Porter, P. Clark, "Representing roles and purpose", Proceedings of the First International Conference on Knowledge Capture (K-CAP 2001), pp. 38-43.
12. E. B. Fernandez; X. Yuan. "Semantic Analisis Patterns", Proceedings of the 19th Int. Conference on Conceptual Modeling (ER'00), LNCS 1920, Springer 2000, pp. 183-195.
13. M. Fowler, "Dealing with Roles", Pattern Languages of Programming (PLoP '97) and EuroPLoP '97 Conference, Technical Report #wucs-97-34, Dept. of Computer Science, Washington University, 1997.
14. M. Fowler, "Analysis Patterns: Reusable Object Models", Addison-Wesley, 1997.
15. E.Gamma, R.Helm, R.Johnson, J. Vlissides, "Design Patterns – Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.
16. A. Geyer-Schulz, M. Hahsler, "Software Reuse with Analysis Patterns", Proceedings of the 8th Americas Conference on Information Systems (AMCIS 2002), August 2002, pp. 1156-1165.
17. G. Gottlob, M. Schrefl, B. Röck, "Extending Object-oriented Systems with Roles", ACM Transactions on Information Systems 14 (3), 1996, pp. 268-296.
18. N. Guarino, "Concepts, Attributes and Arbitrary Relations", Data & Knowledge Engineering 8, 1992, pp. 249-261.
19. T. Halpin, "Conceptual Schema & Relational Database Design", Second Edition, Prentice-Hall of Australia Pty Ltd: Sydnes, 1995
20. A. Jodłowski, P. Habela, J. Płodzien, C. Subieta, "Extending OO Metamodels towards Dynamic Object Roles", R. Meersman et al. (Eds.): On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, LNCS 2888 Springer 2003, pp. 1032–1047.
21. E.A. Kendall, Role Modeling for Agent System Analysis, Design, and Implementation, IEEE Concurrency, vol. 8 no. 2, 2000, pp. 34-41.
22. B.B. Kristensen, Object Oriented Modeling with Roles, Proceedings of the 2nd International Conference on Object-Oriented Information Systems (OOIS'95), 1995
23. D. Lea, J. Marlowe, "Interface-Based Protocol Specification of Open Systems using PSL", 9th European Conference ECOOP'95 - Object-Oriented Programming, LNCS 952 Springer 1995, pp. 374-398.
24. F.G. Mossé, "Modeling Roles - A Practical Series of Analysis Patterns", Journal of Object Technology (JOT) , vol.1 no.4, 2002, pp.27-37.
25. G.M. Nijssen and T.A. Halpin. "Conceptual Schema and Relational Database Design: a fact oriented approach". Prentice-Hall, Sydney, Australia, 1989.

26. A. Olivé, "Integrity Constraints Definition in Object–Oriented Conceptual Modeling Languages", Proceedings of the 22th International Conference on Conceptual Modeling (ER'03), LNCS 2813, 2003, pp.349-362.
27. A. Olivé, "Representation of Generic Relationship Types in Conceptual Modeling·, Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), LNCS 2348, pp. 675-691
28. OMG, "UML 2.0 OCL Specification", Adopted Specification (ptc/03-10-14), 2003
29. OMG, "UML 2.0 Superstructure Specification", Adopted Specification (ptc/03-08-02), 2003
30. OMG, "OMG XML Metadata Interchange Specification", v.1.2, January 2002.
31. M.P. Papazoglou, B.J. Krämer, "A database model for object dynamics", The Very Large Databases (VLDB) Journal (6), January 1997, pp. 73-96.
32. M. P. Papazoglou, "Modeling Object Dynamics", in. M.P. Papazoglou, S. Spaccapietra, Z.Tari (Eds.), Advances in Object-Oriented Data Modeling. MIT Press 2000, pp. 195-217.
33. V. Pelechano, M. Albert, E. Campos, O. Pastor, "Automating the Code Generation of Role Classes in OO Conceptual Schemas", Proceedings of the 4st International Conference on Enterprise Information Systems (ICEIS 2002), 2002, pp. 656-686.
34. V. Pelechano, O. Pastor, E. Insfrán, "Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques", Data & Knowledge Engineering 40, 2002, pp. 315-353
35. B. Pernici, "Objects with Roles", Proceedings of the Conference on Office Information Systems, SIGOIS Bulletin, vol. 11, no. 2/3, ACM Press, New York, 1990, pp. 205-215.
36. T. Reenskaug, P.Wold, O.A. Lehne, Working with Objects: The OOram Software Engineering Method, Prentice-Hall, Englewood Cliffs, NJ, 1995.
37. J. Sowa, "Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley Publishing Company, New York, 1984.
38. F. Steimann, "On the Representation of Roles in Object-oriented and Conceptual Modelling", Data & Knowledge Engineering 35, 2000, pp. 83-106.
39. F. Steimann, "A Radical Revision of UML's Role Concept", UML 2000: The Unified Modelling Language, LNCS 1939, Springer, pp. 194-209.
40. F. Steimann, "Role=Interface", Journal of Object-Oriented Programming, October/November 2001, Vol. 14, Num. 14, pp. 23-32.
41. J.G. Süß, A. Leicher, F. Chabarek, "Software Model Engineering and Reuse with the Evolution and Validation Environment", N.Guelfi, E. Astesiano, G. Reggio (Eds.): Scientific Engineering of Distributed Java Applications, Third International Workshop, FIDJI 2003, November 27-28, 2003, Revised Papers, LNCS 2952, Springer 2004, pp. 196-105.
42. B. Thalheim, "Entity-Relationship Modeling: Foundations of Database Technology", Springer-Verlag, 2000.
43. E. Teniente, "Analysis Pattern Definition in the UML", Proceedings Information Resources Management Association (IRMA) 2003, Idea Group Pub., pp. 774–777.
44. R.Wieringa, W. de Jorge, P.Spruit, "Using Dynamic Classes and Role Classes to Model Object Migration", Theory and Practice of Object Systems, 1(1), 1995,pp. 61-83.
45. R. K. Wong, H. L. Chau, F. H. Lochovsky, "A Data Model and Semantics of Objects with Dynamic Roles", 13th International Conference on Data Engineering, IEEE Computer Society, pp. 402-411.

Appendix A

```
public class Person
{
  public String name;
  public PhoneNumber phone;
  public Date birthDate;
  public Address address;
  Vector rols=new Vector()[4];

  public double age()   { //Age calculation}


  public void addRole(Object o) //Adding a new role
  {
    if (o instanceof Employee)
    { //Checking mayBeHired constraint
      int i=0; int numSt=0; Object o2;
      while (i<rols.size() && numSt<2)
      {
        o2=rols.get(i);
        if(o2 instanceof Student) numSt++;
        i++;
      }
      if(numSt<2) {rols.add(o);
           ((Employee)o).naturalEntityType=this;}
      else System.out.println("Error");
    }
    . . .
  }

  public void deleteRole(Object o)
  {
    if(o instanceof Employee)//Checking mayBeFired
                                          constraint
    {
     if(!((Employee) o).
                workingStatus.equals("MaternityLeave"))
     { rols.removeElement(o);
           ((Employee) o).naturalEntityType=null;}
    }
```

[4] Note that Person has a single multivalued attribute to store all the roles of that person, instead of having a different multivalued attribute for each of its roles (an attribute for the student instances, another for the employee instances…). We can use a single attribute since all the classes in Java are implicit subclasses of the class Object. When dealing with the attribute we make the appropriate castings to the specific role class.

```
    // ...
  }
}

public class Employee
{
  public int emp;
  public String category;
  public Object naturalEntityType;
     . . .
  //Adopted properties
  public String name()
     { return ((Person) naturalEntityType).name; }
  public double age()
     { return ((Person) naturalEntityType).age; }
}
```

Heuristic Strategies for the Discovery of Inclusion Dependencies and Other Patterns*

Andreas Koeller[1]** and Elke A. Rundensteiner[2]

[1] Oracle Corporation, NEDC
Nashua, NH 03062, USA
koeller@acm.org
[2] Department of Computer Science, Worcester Polytechnic Institute,
100 Institute Road, Worcester MA 01609, USA
rundenst@cs.wpi.edu

Abstract. Inclusion dependencies (INDs) between databases are assertions of subset-relationships between sets of attributes (dimensions) in two relations. Such dependencies are useful for a number of purposes related to information integration, such as database similarity discovery and foreign key discovery.
An exhaustive approach at discovering INDs between two relations suffers from the dimensionality curse, since the number of potential mappings between the attributes of two relations is exponential in the number of attributes. For this reason, levelwise (Apriori-like) approaches at discovery do not scale beyond relations with 8 to 10 attributes. Approaches modeling the similarity space as graphs or hypergraphs are promising, but also do not scale very well.
This paper discusses approaches to scale discovery algorithms for INDs and some other similarity patterns in databases. The major obstacle to scalability is the exponentially growing size of the data structure representing potential INDs. Therefore, the focus of our solution is on heuristic techniques that reduce the number of IND candidates considered by the algorithm. Despite the use of heuristics, the accuracy of the results is good for real-world data.
Experiments are presented assessing the quality of the discovery results versus the runtime savings. We conclude that the heuristic approach is useful and improves scalability significantly. It is particularly applicable for relations that have attributes with few distinct values.

1 Introduction

In database research, and in particular in database design, modeling, and optimization, much emphasis has been placed on dependencies in databases. A vast field of research deals with functional dependencies (FDs), and many other dependencies between attributes of the same relation have been studied.

However, one type of dependency, called Inclusion Dependency (INDs) [1], is defined across *two* relations. The problem of IND *discovery*, which is addressed in this

* This work was supported in part by NSF grant #IIS9988776.
** This work was performed while the author was affiliated with Montclair State University, Montclair, NJ, USA

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 185–210, 2006.

paper, involves finding the minimal set of maximal inclusion dependencies between two relations from the data in the relations, rather than from implied or explicit knowledge about schemas (e.g., based on attribute names, features, or ontologies). IND discovery is also different from IND *inference* [1,2], which is the problem of discovering new INDs from known INDs, by using inference mechanisms such as transitivity, projection and permutation. IND discovery proceeds by querying or otherwise examining data across two relations without prior knowledge about those relations, in order to find inclusion patterns between them.

Solving IND discovery problems is interesting for a number of applications. INDs describe subset-relationships between projections (sets of attributes) of two relations, and can be thought of as related to the "specialization" relationship of object-oriented systems. For example, *foreign key constraints* are nothing but true (valid) INDs between a foreign key in one table and the associated key in another. Foreign key and functional dependency discovery [3] can be used to reorganize legacy database systems. In query rewriting, algorithms that answer queries over information spaces with partially redundant tables benefit from knowledge of INDs [4]. Examples can be found in the literature, e.g., query folding [5,6,7]. In the context of schema integration and matching [8], knowledge of redundancies across sources is essential. INDs represent such redundancies.

The problem of IND discovery is NP-hard [2], and enumeration algorithms are prohibitively slow, even for small real-world problems [9,10]. Since the problem is related to the discovery of functional dependencies [3] and association rule mining [11], proposals exist to adapt successful algorithms from those domains to the IND discovery problem [10]. In particular, those algorithms use a *levelwise strategy* [12], discovering single-attribute INDs first, then two-attribute (binary) INDs, then higher-order INDs. However, this approach does not scale beyond very modestly sized problems, as demonstrated in [9] and [10].

In previous work [13], the authors have proposed a scalable algorithm called FIND_2 that *discovers* INDs between unknown relations. Another similar algorithm, called *Zigzag*, has been independently proposed by de Marchi *et al.* [14]. The FIND_2 algorithm and the *Zigzag* algorithm approach the IND discovery problem from similar directions. They both observe that the solution to an IND discovery problem can be mapped to a *hypergraph*. Thus they can map the problem of IND discovery to a problem of discovering a hypergraph from limited knowledge of the hypergraph's nodes and edges. The algorithms employed in both approaches (hyperclique finding in FIND_2 and minimal traversal in *Zigzag*) are polynomial in the number of edges, and therefore exponential in the number of nodes in the hypergraph (since the number of edges in a general hypergraph of k nodes is bounded by 2^k). In the problem mapping applied in those algorithms, discovery problems over relations with 50 attributes (a common size) can easily lead to hypergraphs with hundreds of nodes, which for an algorithm running in exponential time in the number of nodes poses a serious problem [9,14]. Depending on the data in the source relations, even relations with 10–20 attributes can lead to unacceptably high runtimes or memory problems.

This paper deals with heuristic strategies to scale hypergraph-based IND-discovery algorithms beyond the sizes manageable in the basic hypergraph approach. The heuris-

tics reduce the size of hypergraph data structures involved in the discovery process by exploiting easily computable database statistics. While the non-heuristic FIND_2 and *Zigzag* algorithms find the exact problem solution, some of the strategies proposed here reduce the completeness of the solution. That is, the heuristics will sometimes prevent the finding of *all* INDs, but all INDs that *are* discovered will be correct, and often at least the *largest* IND in a given problem will be found.

It should be noted here that our work is orthogonal to manual or semi-automatic discovery of database relationships, as suggested by many research works [15,16] and implemented in many industrial software solutions. Our algorithms do not make use of domain knowledge such as ontologies, expert-supplied attribute relationship information, or use other schema-driven techniques. They exclusively use the *data* in the information sources to suggest relationships between databases.

The contributions of this paper are as follows: We identify and define "spurious" inclusion dependencies (INDs) as a major reason for performance problems in IND discovery. Then, we give a model of detecting such INDs. We also show how to derive heuristics based on this model, give additional suggestions as to the improvement of IND discovery, and present an experimental study of the advantages of our heuristic algorithm.

A preliminary version of this work was presented at the ODBASE 2004 conference [17]. In addition to the work published there, this paper contains: a discussion of the quality of results in heuristic IND discovery, a discussion of the use of the algorithm presented for discovery of patterns *other* than INDs, as well as four new experiments comparing our algorithm with an alternative technique, studying the effects of heuristics on quality, evaluating the DV heuristic specifically, and assessing the usefulness of the algorithm for non-IND pattern discovery.

The remainder of this paper is organized as follows: Section 2 reviews INDs and a hypergraph-based discovery algorithms for them. Section 3 introduces spurious INDs and motivates the concept. Section 4 introduces heuristics based on that notion and their application to IND discovery. Section 5 discusses how to assess the quality of the heuristic algorithm's results. Section 6 suggests the application of this algorithm to detect *approximate* relationships between tables, rather than INDs, which are based on exact subsets. Section 7 discusses experimental data to support our theoretical results and assess the algorithms. Sections 8 and Section 9 present related work and conclusions, respectively.

2 Background

2.1 Problem Definition

Our goal is to solve the problem of deducing all inclusion dependencies between two given relations solely from the data in the relations. Inclusion dependencies are defined as below.

Definition 1 (IND). *Let $R[a_1, a_2, \ldots, a_n]$ and $S[b_1, b_2, \ldots, b_m]$ be (projections on) two relations. Let X be a sequence of k distinct attribute names from R and Y a sequence of k distinct attribute names from S, with $1 \leq k \leq \min(n, m)$. Then an **inclusion dependency (IND)** σ is an assertion of the form $\sigma = R[X] \subseteq S[Y]$. k is called*

*the **arity** of σ and denoted by $|\sigma|$. An IND $\sigma = (R[a_1, \ldots, a_k] \subseteq S[b_1, \ldots, b_k])$ is **valid** between two relations R and S if the sets of tuples in R and S satisfy the assertion given by σ.*

Due to its unclear semantics, we do not consider duplication of attributes on either side of the IND (i.e., we require sequences X and Y to be composed of distinct attributes). However, allowing duplicate attributes is possible and would not significantly increase the runtime of the algorithms presented here.

Casanova et al. [1] give a complete set of inference rules for INDs, observing that INDs are *reflexive*, *transitive* and invariant under *projection and permutation*. Permutation here refers to the reordering of attributes on *both* sides of the IND. For example, $R[AB] \subseteq S[KL] \equiv R[BA] \subseteq S[LK] \not\equiv R[BA] \subseteq S[KL]$. Transitivity is defined as usual, $R[X] \subseteq S[Y] \wedge S[Y] \subseteq T[Z] \Rightarrow R[X] \subseteq T[Z]$.

Projection invariance of INDs is the key to discovery algorithms. By projection, a valid k-ary IND with $k > 1$ **implies** sets of m-ary valid INDs, with $1 \leq m \leq k$. Specifically, for a given valid IND $\sigma = R[X] \subseteq S[Y]$, the IND $\sigma' = R[X'] \subseteq S[Y']$ will be valid for any subsequence $X' \subseteq X$ and its corresponding subsequence $Y' \subseteq Y$. Such a set of m-ary INDs implied by a k-ary IND has a cardinality of $\binom{k}{m}$ and is denoted by Σ_m^k. A very important observation is that the validity of all implied m-ary INDs of a given IND σ is a *necessary but not sufficient* condition for the validity of σ. For example, $(R[A_1] \subseteq S[B_1]) \wedge (R[A_2] \subseteq S[B_2]) \wedge (R[A_3] \subseteq S[B_3])$ does not imply $R[A_1, A_2, A_3] \subseteq S[B_1, B_2, B_3]$, as can easily be seen through an example (Fig. 1).

R

A_1	A_2	A_3
1	4	7
2	5	8
3	6	9

S

B_1	B_2	B_3
1	4	7
2	5	8
3	6	-1
-1	6	9
3	-1	9

$R[A_1, A_2] \subseteq S[B_1, B_2]$ is valid.
$R[A_2, A_3] \subseteq S[B_2, B_3]$ is valid.
$R[A_1, A_3] \subseteq S[B_1, B_3]$ is valid.
$R[A_1, A_2, A_3] \subseteq S[B_1, B_2, B_3]$ is not valid.

Fig. 1. Validity of all implied INDs is not a sufficient validity test.

Due to the projection invariance, a set Σ of INDs between two relations can be described by a **cover of INDs**, denoted by $\mathcal{G}(\Sigma)$. Intuitively, this is a minimal set of INDs from which all INDs in Σ can be derived by projection, permutation, and transitivity. Naturally, $\mathcal{G}(\Sigma) \subseteq \Sigma$. With these observations, the IND discovery problem reduces to the problem of finding a *cover of INDs* for a given pair of relations.

2.2 IND-discovery Algorithms

Exhaustive Discovery Since $|\Sigma_m^k| = \binom{k}{m}$, the number of valid INDs implied by a single k-ary IND σ is exponential in k: $\sum_{m=1}^{k-1} \binom{k}{m} = 2^k - 2$. Furthermore, INDs are *not*

invariant under permutation of the attributes of just *one* side, but only if the attribute lists on both sides are permutated synchronously. That means for example that, when *discovering* INDs between two relations with k attributes, one has to test $k!$ potential INDs just for the hypothesis that the one relation is completely included in the other. Consequently, exhaustive enumeration algorithms are exponential and not feasible for IND discovery.

Since Apriori-like algorithm are the standard solution for many discovery problems (e.g., for association rules), the question arises whether such an algorithm might be appropriate for our problem. In fact, a levelwise algorithm [12] akin to the Apriori algorithms in association rule mining [11] has been proposed as a solution to this problem [10]. It discovers unary INDs first and then forms binary IND candidates from the valid unary INDs. Those INDs then have to be validated against the database. From the validated binary INDs, 3-ary INDs are formed, then tested, and so on. In the presence of a single sufficiently large valid IND σ, such an algorithm will have to discover $2^{|\sigma|} - 2$ implied INDs before even *considering* σ. This is clearly not a feasible approach. Experiments conducted by the authors (see Sec. 7) and de Marchi [10] both suggest that levelwise algorithms do not scale beyond a maximal IND size of 8–10.

Hypergraph-Based Discovery In general, the worst-case complexity of the problem is determined by the number of possible distinct INDs between two relations. However, in real-world problems, one expects to find a very low number of large distinct INDs (in fact, often just one), and possibly several small INDs (see also Sec. 5). Therefore, it is meaningful to find a *minimal cover* of valid INDs without even enumerating all valid INDs, reducing the complexity significantly.

For this purpose, the problem is mapped into a graph problem. We use a family of **k-uniform hypergraphs** which are graphs in which each edge is incident to exactly k nodes. Standard undirected graphs can be considered "2-uniform hypergraphs". Furthermore, we extend the concept of *clique* (maximal connected subgraph) to hypergraphs.

Definition 2 (hyperclique). *Let $G = (V, E)$ be a k-uniform hypergraph. A **hyperclique** is a set $C \subseteq V$ such that for each k-subset S of distinct nodes from C, the edge corresponding to S exists in E.*

In analogy to above, a clique is a hyperclique in a 2-hypergraph.

To map our problem, we now map the set of valid INDs to a family of hypergraphs G_m $(2 \leq m < k)$, by making all k-ary valid INDs hyperedges in a k-uniform hypergraph. The nodes of all hypergraphs (for any k) are formed by the unary INDs. For example, the first hypergraph for $k = 2$ has as its nodes all valid *unary* INDs, and as its edges all valid *binary* INDs.

We then use the fact that, for $m = 2 \ldots k - 1$, any set Σ_m^k of INDs implied by a valid σ_k maps to a hyperclique in the corresponding hypergraph G_m. In other words, after an initial step of discovering low-arity INDs ($k = 1 \ldots 2$), we can form candidates for valid high-arity INDs by considering only those potential INDs that correspond to cliques in k-uniform hypergraphs for small k.

Algorithm FIND$_2$ Algorithm FIND$_2$ (Fig. 2) applies hyperclique-finding techniques to find inclusion dependencies (INDs). It was published as part of a dissertation [9] and also appears in [13]. Full details and derivations can be found in [18]. FIND$_2$ takes as input two relations R and S, with k_R and k_S attributes, respectively and returns a cover $\mathcal{G}(\Sigma)$ of INDs between R and S. The algorithm proceeds in stages enumerated by a parameter $k = 2, 3, \ldots$. It begins by exhaustively validating unary and binary INDs, forming a (2-uniform) hypergraph using unary INDs as nodes and binary INDs as edges (Step 1, $k = 2$). A clique-finding algorithm then determines all higher-arity INDs candidates (Step 2, candidates c_1 and c_2 in the figure). Since the clique property is necessary but not sufficient for the validity of a higher-arity IND (Sec. 2.1), each IND candidate thus discovered must also be checked for validity. Each IND that tests invalid (but corresponds to a clique in the 2-hypergraph) is broken down into its implied 3-ary INDs. They then form the edges of a 3-hypergraph (Step 3, $k = 3$). Edges corresponding to invalid INDs are removed from the 3-hypergraph.

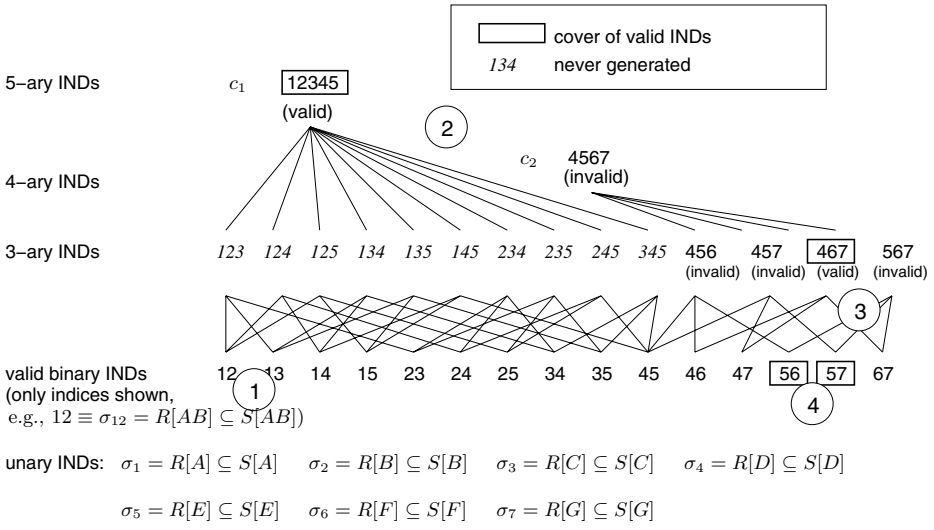


Fig. 2. Overview of the complete algorithm FIND$_2$.

Then, hypercliques are found in the 3-uniform hypergraph formed with unary INDs as nodes and 3-ary INDs as edges. Hypercliques found are new IND candidates. Invalidated IND candidates found in this step are broken down into 4-ary subsets ($k = 4$). The process is repeated for increasing k until no new cliques are found. At each phase, some small elements of the cover $\mathcal{G}(\Sigma)$ might be missed and are discovered by a cleanup process (Step 4, see also [18]). In all of our experiments using real data sets, the algorithm terminated for $k \leq 6$ (in Fig. 2, the algorithm terminates for $k = 3$).

Since the publication of FIND$_2$, de Marchi *et al.* have independently proposed a similar algorithm called *Zigzag* [14], which uses the same basic model as ours, but em-

ploys *minimal hypergraph traversals* [3,19] instead of clique-finding in order to generate large IND candidates. Furthermore, they introduce an optimization to the treatment of invalidated large IND candidates (e.g., c_2 in Fig. 2), in that they also attempt to validate such a failed IND candidate by projecting out single attributes from it, rather than restarting the discovery process for $k+1$. They make a decision as to which strategy to apply based on the number of tuples in relations R and S that violate the hypothesized IND.

3 The Semantics of Inclusion Dependencies

Attribute sets that stand in an IND to each other are not necessarily matches for the purpose of data integration. INDs can occur between attributes "by accident", especially if attributes have few distinct values and have similar or equal domains. Therefore, an algorithm enumerating all inclusion dependencies across two database tables is likely to produce some results that are not interesting for the purpose of data integration or schema matching.

3.1 Why Improve IND Discovery?

Algorithms FIND$_2$ and *Zigzag* as described so far find the complete and correct solution to the IND-finding problem for two given relations. In principle, both algorithms first discover unary and binary INDs by enumeration and testing (called *pessimistic strategy* in [14]), and then form an *optimistic hypothesis* about the IND space by assuming that all high-arity INDs that *could* be valid based on the validated unary and binary INDs *are* in fact valid. That assumption makes both algorithms extremely sensitive to an overestimation of valid unary and binary INDs. A high number of such small INDs would cause many invalid larger IND candidates to be generated and tested against the database.

Also, several of the algorithms involved, in particular the hypergraph-based pattern discovery (hyperclique-finding in FIND$_2$, min-hypergraph traversal in *Zigzag*), have high complexity [19,18], and are fast only for sparse hypergraphs.

Therefore, the overall discovery strategy is sensitive to such overestimations and it is important to prune unnecessary INDs from the search space.

3.2 Spurious INDs

We will now motivate the concept of "overestimating" INDs. For this purpose, we define a notion of "accidental" or "spurious" INDs which are valid in the database but do not contribute significantly to finding a solution to our problem.

Definition 3 (Spurious IND). *An inclusion dependency $\sigma = R[A] \subseteq S[B]$ is called spurious iff (1) it is valid in the database and (2) does not reflect a* semantic relationship *between attribute sets A and B (i.e., A and B do not represent the same real-world dimensions).*

The exact meaning of "semantic relationship" depends somewhat on the context in which the IND discovery is used. For example, in schema matching, semantically related attributes would be mapped into the same attribute in the integrated schema. In query rewriting, a semantic relationship between two attributes would represent a redundancy between those attributes.

Often, spurious INDs occur when the domains of attributes are small (i.e., if attributes have many duplicate values), as the following example illustrates.

Example 1. Consider Fig. 3 for an example. The domains of three columns in table *Member* and two columns in table *Former* are "year", which is a domain with few values. The figure shows the cover $\mathcal{G}(\Sigma)$ of INDs for this problem.

Member

Name	Birthyear	MemberSince	MemberUntil
Jones	1940	1969	1989
Miller	1945	1960	1988
Myers	1960	1980	1988
Shultz	1969	1988	1989
Becker	1961	1989	

Former

Member	YOB	LeftIn
Myers	1960	1988
Shultz	1969	1989

Former[Member,YOB,LeftIn] ⊆ Member[Name,Birthyear,MemberUntil],
Former[**YOB,LeftIn**] ⊆ **Member**[**MemberSince,MemberUntil**]
Former[**LeftIn**] ⊆ **Member**[**MemberSince**]

Fig. 3. Accidental INDs introduced by small domains

Two low-arity INDs are part of the cover of INDs between **Former** and **Member**, shown in bold font in Fig. 3. However, in some sense, these INDs are intuitively "wrong". Note that they are not implied by any INDs with arity larger than 2. Therefore, the discovery algorithm will not need these INDs for finding INDs with arity > 2 and pruning them from the search space would speed up the algorithm while not significantly reducing the quality of its result.

3.3 Detecting Whether an IND Is Spurious

Algorithms FIND_2 and *Zigzag* both treat testing a single IND as an elementary operation with a binary result. A test for binary IND validity can simply be performed by formulating a database query. In SQL, one could employ the EXCEPT (set-difference) operator, since $R[A] \subseteq S[B] \iff \left(\left|R[A] \backslash S[B]\right| = 0\right)$. This however does not generate any information about the "spuriousness" of the IND.

In order to assess the probability for spurious INDs to occur we now look at a statistical model. Consider a sample N of size n obtained by sampling with replacement from a set K of k objects. Given a certain set $R \subseteq K$ of size $r \leq n$, consider the probability that all values in R are included in the sample N and denote it by $P(n, r, k)$. It can be computed by the following formula.

Theorem 1. *Consider a set $R = \{e_1, \ldots, e_r\}$ of r distinct elements from a universe K of k distinct elements. The probability that a random sample (obtained by sampling with replacement) of size n from K contains set R is*

$$P(n,r,k) = 1 - \frac{\sum_{i=1}^{r} (-1)^{i+1} \cdot \binom{r}{i} \cdot (k-i)^n}{k^n} = 1 - \sum_{i=1}^{r} (-1)^{i+1} \cdot \binom{r}{i} \cdot \left(1 - \frac{i}{k}\right)^n \quad (1)$$

Proof. There are k^n different samples of size n from k distinct elements (sampling with replacement). We compute how many of those do *not* contain R. A sample that does not contain R is missing at least one element from R. Let us denote by $A_{\overline{e}}$ the set of all samples that are missing element e. Then, the number of samples that do not contain at least one element from R is $r_0 = |A_{\overline{e_1}} \cup A_{\overline{e_2}} \cup \cdots \cup A_{\overline{e_r}}|$.
We now need to determine the size of the union of all those sets. The size of each $A_{\overline{e}}$ is $(k-1)^n$. In analogy, the size of $A_{\overline{e_1}} \cap A_{\overline{e_2}}$ (the set of all samples missing *two* given elements) is $(k-2)^n$, and so on. Since we can compute the sizes of their intersections, we can use the *inclusion-exclusion rule* of combinatorics[3], and get $r_0 = \sum_{i=1}^{r} (-1)^{i+1} \cdot \binom{r}{i} \cdot (k-i)^n$. We then get the probability $P' = \frac{r_0}{k^n}$ that a sample does *not* contain R. Therefore $P(n,r,k) = 1 - P'$, □

In order to now determine the probability of "spurious INDs', assume two relations R and S and the problem of assessing whether a valid IND $\sigma = R[A] \subseteq S[B]$ is spurious. Let A have r distinct values. Furthermore, set $n = |S|$, i.e., n is the number of (non-distinct) values in attribute B. One can argue that since the values in A are a subset of the values in B, the values in both attributes are from a common domain K with k distinct elements.

We are interested in the "chance" that attribute A just "happens" to be included in attribute B. This "chance" can be assessed by the probability that a sample (with replacement) of size n from K contains A, which is $P(n,r,k)$.

Now note that $\lim_{n\to\infty} (1 - \frac{i}{n})^n = e^{-i}$. Define $k = \frac{n}{c}$ and insert it into the rightmost term in Equation 1. Since $\lim_{n\to\infty} (1 - \frac{ci}{n})^n = e^{-ic}$, that means that for large n and k, the value of $P(n,r,k)$ depends approximately only on r and $c = \frac{n}{k}$.

In Table 1 we have listed the maximum value of c for which $P(n,r,k)$ remains lower than 5%, for different r. That is, for a given number of distinct values in an attribute A, we can estimate how likely it is that A is contained in an attribute B by chance, given the size of B and the size of the common domain of A and B. This is a measure of how likely $R[A] \subseteq S[B]$ is to be spurious.

Of course, the size of domain K is unknown. However, since we have assumed initially that $R[A] \subseteq S[B]$, we could assume that K is given by the distinct values in B. In this case, $n > k$ and thus $c \geq 1$. In this case, we get a $P < 0.05$ only if $r > 7$.

We conclude that inclusion dependencies where the included attribute has less than 6 or 7 distinct values have a high chance of being valid by statistical coincidence, rather than by semantic relationships between the attributes. We exploit this result to restrict the search space of our algorithm.

[3] This is a generalization of $|A \cup B| = |A| + |B| - |A \cap B|$. See also [9].

Table 1. Minimum number of distinct values to avoid spurious INDs.

r	$P(n,r,k) < 0.05$ for $c = n/k$ less than	r	$P(n,r,k) < 0.05$ for $c = n/k$ less than
2	0.25	7	1.06
3	0.46	10	1.35
4	0.64	20	1.97
5	0.80	50	2.85
6	0.93	100	3.53

4 Heuristics for IND-validity Testing

From the observations above, we have derived two heuristics which are useful in reducing the number of IND candidates considered in a discovery problem.

4.1 The Number-of-Distinct-Values (DV) Heuristic

Based on our definition of spuriousness, the DV heuristic states that an IND $R[A] \subseteq S[B]$ should *not* be used as a node or edge in a hypergraph in algorithm FIND_2 (or similar algorithms such as Zigzag) if the attribute (or attribute set) A has few distinct values (tuples). That is, this heuristic simply discards all inclusion dependencies in which the included attribute has less than n distinct values.

This method is supported by our theoretical results in Sec. 3.3, which state that $r = \delta(R[A])$ (the number of distinct values in attribute A) must be relatively large for the IND $R[A] \subseteq S[B]$ to not be considered spurious. From the theory, we would set a value of $n = 7$, a choice that is confirmed by our experiments.

The DV heuristic can only be used to test for *valid* INDs, i.e., an IND that is already considered invalid will not be affected. It may produce false negatives, i.e., declare INDs as spurious that are in fact not. Therefore, this heuristic has to be used carefully, as explained in Sec. 4.3.

4.2 The Attribute-Value-Distribution (AVD) Heuristic

The Attribute Value Distribution (AVD) heuristic has strong predictive power for many data sets. It is based on the hypothesis that two attributes A and B that form a non-spurious IND (i.e., are semantically related) have the same frequency distribution of values.

Obviously, this is strictly only true if A and B are both randomly taken from a common set of values. However, for the purpose of this paper, we are assuming that semantically related attributes *are* both taken from such a common set. Therefore, the additional assumption that they are *random* samples seems reasonable at least for some cases. The heuristic then states the following:

If the values of attributes A and B in a valid IND $\sigma = R[A] \subseteq S[B]$ do not *show the same value distribution, the attributes are not semantically related.*

That is, if the value distribution is found to be different, the σ can be considered spurious. If it is not different, no new information is gained about σ. This heuristic can produce false negatives when attributes that are actually semantically related are rejected due to the fact that they actually do not have similar frequency distributions. The statistical hypothesis testing itself, which is probabilistic in nature, may also lead to false negatives.

Performing Statistical Hypothesis Testing for AVD For the hypothesis test, we use the widely applicable χ^2-Test [20], in particular a χ^2-Test for independence. This test is designed to assess the independence of two categorical variables x and y. The χ^2-Test then tests under the null hypothesis that the two variables x and y are independent, i.e., that the value of variable x does not influence the value of variable y.

For our purpose we perform the following mapping: Given an IND $R[A] \subseteq S[B]$, we set $x = \{\mathrm{A}, \mathrm{B}\}$ (i.e., the *names* A and B) and $y = \delta(R[A]) \cup \delta(R[B])$, where $\delta(R[A])$ denotes the set of distinct values in attribute A of relation R. The *contingency table* used for the χ^2-Test is then filled with the *counts* of each distinct data value in each of the two attributes.

We are therefore testing for the null hypothesis: "the distribution of values in an attribute does not depend on the choice of attribute (out of $\{A, B\}$) from which the values are taken". If this hypothesis is rejected (i.e., if the value distribution *is* dependent on the choice of attribute), we conclude that the value distributions in the two attributes are different, and consequently an IND between them is spurious.

The attribute value distribution in a single attribute can be obtained easily through an SQL-query and can be pre-computed for all attributes. For larger INDs, attribute values can be concatenated to compute AVDs.

4.3 Incorporating Heuristics into the IND-checking Algorithm

The heuristic-based IND-checking function, called $\mathsf{CHECK}_\mathsf{H}$, is shown in Fig. 4. While the basic FIND_2 algorithm uses a simple database query to detect the validity of an IND, the heuristic algorithm, called FIND_H, uses this heuristic check function. $\mathsf{CHECK}_\mathsf{H}$ employs the DV and AVD heuristics introduced above, and also performs a simple check for compatible domains. Note that the AVD heuristic is only used when (1) the IND is valid in the database and (2) the DV heuristic rejects the IND. The intuition is that the AVD heuristic is a stronger test of spuriousness than the DV heuristic and can detect a semantic relationship (and thus "pass" the IND) where the DV heuristic failed. The CHECK-function performs a validity check of a single IND against the source database(s) through a database query and returns a Boolean value.

The computational complexity of IND-checking against the database is quite high, as a check involves computing a set difference, and is consequently of $O(n \log n)$ complexity in the number of tuples in the relations. De Marchi [10] proposes the use of an inverted index of data values in order to facilitate the computation of unary INDs only. This approach is not applicable for binary or higher-order INDs. Further improvements in the *testing* of INDs (rather than the generation of IND *candidates*) could be beneficial.

```
function CHECK_H(Relation R, AttList A of R, Relation S, AttList B of S)
    if (domains of R[A] and S[B] incompatible)
        return invalid
    else if (CHECK(R, A, S, B) = invalid)   //a check against the database
            return invalid
        else if (DV heuristic does not reject IND)
                return valid
            else if (AVD heuristic rejects IND)
                    return invalid  //false negative possible
                else return valid
```

Fig. 4. The heuristic IND-checking function $\mathsf{CHECK}_\mathsf{H}$ used by algorithm FIND_H

4.4 Detecting INDs in the Presence of False Negatives

Consider a complete graph (i.e., a graph with all possible edges) $G = (V, E)$. Then, the set of nodes V forms a clique in G. Now remove a single edge from E. Clearly, the clique property does no longer hold, but rather G will now contain at least two distinct maximal cliques. Those cliques are likely to have a substantial overlap (i.e., common set of nodes).

If any of our heuristics produces false negatives, some edges (or even nodes, i.e., unary INDs) of any graph or hypergraph considered by FIND_2 may be missing. The clique finding algorithms used by FIND_2 will then no longer find cliques that correspond to the maximal INDs in the problem given, but rather find only smaller subsets of those cliques. Simulations show that the removal of as few as 5 random edges from a clique of 40 or 50 nodes will generally produce a graph with around 20 distinct maximal cliques. However, those sub-cliques will often show substantial overlaps. Therefore, we use the following strategy: *When heuristics are used in* FIND_2 *that may produce false negatives (i.e., reporting non-spurious INDs as invalid), and* FIND_2 *reports several large, overlapping INDs, then we merge those INDs by computing the union of their nodes.*

Naturally, merging *all* INDs found by algorithm FIND_2 will in general not lead to a valid INDs, unless the (true) cover of INDs actually contains only one IND. Therefore, we merge INDs of decreasing size, starting from the largest, until adding another IND to the result will no longer produce a valid IND.

Our experiments show that the IND-merging heuristic is powerful enough to find large or maximal valid INDs even in cases when many underlying edges are pruned in earlier stages of heuristic discovery (Sec. 7).

5 Quality of Results in Heuristic IND Discovery

Discovery of INDs typically has the goal of discovering relationships between databases. As such, finding the *largest* set of related attributes between two given tables is an important subgoal. If that largest IND is large (has many attributes) compared to

any other INDs in the solution, it represents more information about the relatedness of the databases than the smaller INDs. The reason for this is that a 10-ary (or any high-arity) IND is very unlikely to hold by accident. If a high-dimensional IND is found between two tables, it most likely represents an actual semantic relationships between the attributes in the IND, rather than a random pattern that is true due to statistical coincidence.

See Table 2 for a typical result of an IND discovery. In the example, two tables have a set of 10 attributes each, which stand in an Inclusion Dependency relationship to each other. This is represented by the 10-ary IND discovered by the exact algorithm (left column in the table). The exact algorithm also found 7 more INDs, one unary one, 5 binary ones, and one 3-ary one, which are not implied by the 10-ary IND. Those 7 INDs are most likely spurious by our definition (Def. 3) since they do not seem to represent a semantic relationship between the tables. See also Fig. 5.

Table 2. Sets of Maximal Distinct INDs discovered by Heuristic FIND_H and Exact FIND_2 Algorithms.

IND Size	Exact Algorithm (FIND_2)	Heuristic Algorithm (FIND_H)
1	1	1
2	5	1
3	1	0
4	0	0
5	0	1
6	0	0
7	0	3
9	0	0
10	1	0

Furthermore, a large IND implies many smaller INDs (Sec. 2.1). In the example in Table 2, the 10-ary IND σ_1^{10} implies $2^{10} - 2 = 1022$ smaller INDs, whereas all the remaining maximal INDs ($\sigma_1^1 \ldots \sigma_1^3$) together imply only $5 \cdot (2^2 - 2) + 1 \cdot (2^3 - 2) = 16$ more smaller INDs, most of which are already subsumed by σ_1^{10} or are duplicates of each other.

On the other hand, the heuristic FIND_H algorithm did not find all of the INDs between the two tables (Fig. 2). Instead of σ_1^{10}, it found three 7-ary INDs, which are fragments of the 10-ary true IND. Note that the union of the attributes of the three INDs $\sigma_1^7 \ldots \sigma_3^7$ is exactly σ_1^{10}.

This means that using IND-merging, the heuristically found solution is essentially as useful as the exact one. However, the 7-ary INDs imply only a total of $3 * (2^7 - 2) = 372$ INDs, less than 40% of the total number of INDs implied by σ_1^{10}. Most of those INDs are also duplicates of each other since those three INDs have a 6-attribute overlap. Therefore, counting the *total* number of INDs in the solution is not a good measure for the quality of the result.

$$\begin{aligned}
\sigma_1^1 &= R[C] \subseteq S[B] \\
\sigma_1^2 &= R[B, J] \subseteq S[C, J] \\
\sigma_2^2 &= R[F, J] \subseteq S[I, J] \\
\sigma_3^2 &= R[H, J] \subseteq S[I, J] \\
\sigma_4^2 &= R[H, J] \subseteq S[F, J] \\
\sigma_5^2 &= R[E, H] \subseteq S[E, F] \\
\sigma_1^3 &= R[E, F, J] \subseteq S[E, H, J] \\
\sigma_1^{10} &= R[A, B, C, D, E, F, G, H, I, J] \subseteq \\
&\quad S[A, B, C, D, E, F, G, H, I, J]
\end{aligned}$$

Fig. 5. INDs discovered by Exact FIND_2 Algorithm.

$$\begin{aligned}
\sigma_1^1 &= R[C] \subseteq S[B] \\
\sigma_1^2 &= R[B, J] \subseteq S[C, J] \\
\sigma_1^5 &= R[B, C, D, G, I] \subseteq S[B, C, D, G, I] \\
\sigma_1^7 &= R[A, B, C, D, E, F, I] \subseteq \\
&\quad S[A, B, C, D, E, F, I] \\
\sigma_2^7 &= R[A, B, C, D, E, H, I] \subseteq \\
&\quad S[A, B, C, D, E, H, I] \\
\sigma_3^7 &= R[A, B, C, D, E, I, J] \subseteq \\
&\quad S[A, B, C, D, E, I, J]
\end{aligned}$$

Fig. 6. INDs discovered by Heuristic FIND_H Algorithm.

The primary difference between the exact and the heuristic algorithms is their treatment of unary and binary INDs, since the heuristics are not applied for higher-arity INDs. Therefore, we can also compare the counts of those unary and binary INDs as a measure of result quality. In the example above, the exact algorithm FIND_2 discovered 16 valid (not necessarily maximal) unary INDs. On the other hand, the heuristic FIND_H algorithm regarded 4 of those unary INDs as spurious. Those 4 INDs were actually not implied by the large IND σ_1^{10}, which means the heuristic correctly disregarded them. The FIND_H algorithm then generated only 62 possible binary INDs to test against the database, as opposed to 105 in the exact algorithm, which represents a 40% savings in runtime for this phase. However, the distinct-value heuristic for the binary INDs rejected 8 of the valid 46 binary INDs, some of which *were* implied by σ_1^{10}. Thus, the cause of quality loss in this case was the distinct-value heuristic for binary INDs.

6 Heuristics for Discovering IND-like Database Similarities

Algorithms FIND_H and FIND_2 discover INDs, which are strict set-inclusion patterns. However, they can also be used to discovery patterns that are not technically INDs, but rather "IND-like" pattern. In particular, the discovery of *similarities (near inclusion)* between tables rather than strict inclusion is possible if the similarities are strong enough.

The current algorithm uses SQL set-difference queries (see also Sec. 3.3) to detect inclusion of a given projection of the two tables in question. A projection $\pi_{\overline{A}}(R)$ on a table R is included in a projection $\pi_{\overline{B}}(S)$ in table S if the result of the relational query $\Delta = \pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)$ is empty. However, if Δ is not empty, its size $|\Delta|$ (i.e,. the number of tuples in the difference relation) can be an indicator for the relatedness of the projections. In its simplest form, a small $|\Delta|$ indicates a good relationship, while a large $|\Delta|$ suggests no relationship. A somewhat stronger heuristic is to use the ratio

of $|\Delta|$ to the number of distinct tuples in $\pi_{\overline{A}}(R)$ and/or $\pi_{\overline{B}}(S)$. This "relatedness" score can be used to rank IND-like patterns, which is important when at some stage in the algorithm, too many patterns are discovered to consider all. This ranking is also useful in the final output of the algorithm, as it gives additional information on whether a pattern discovered is real or not.

Partial Overlap Heuristic Determining whether two projections $\pi_{\overline{A}}(R)$ and $\pi_{\overline{B}}(S)$ of relations R and S are related if they do not satisfy an IND (i.e. if $\pi_{\overline{A}}(R) \not\subseteq \pi_{\overline{B}}(S)$) can be done in the following way:

- If $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| < c_1$, the projections are considered related. c_1 can either be a integer constant ($c_1 \geq 1$) or can be a function of the number of attributes in $\overline{A}$ (i.e,. the arity of the IND-like pattern). The rationale is that a very small number of "violating" tuples in the set difference between the two projections could be caused by noise in the data rather than a non-relatedness. An empirically found useful value for c_1 is $c_1 = 3$.
- If $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| < c_2 \cdot |\pi_{\overline{A}}(R)|$ the projections are considered related as well, with $0 < c_2 \leq 1$. The rationale here is that if the number of distinct tuples in the set-difference is less than a fraction of the number of distinct tuples in the left ("smaller") relation of the IND-like pattern, a relationship between the projections is likely. We experimented successfully with a $c_2 = 0.49$, which represents the fact that the smallest useful domain in a relational database must have a size of two (two distinct values, one of which could be null). In this way, for example, an attribute with a two-valued domain would not be considered related to another attribute unless there is a true IND between the two attribute sets (i.e., $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| = 0$), while two attribute sets $\overline{A}$ and $\overline{B}$ with larger domains could be considered related even if $\pi_{\overline{A}}(R) \not\subseteq \pi_{\overline{B}}(S)$.

If either of those two conditions is satisfied, the projections will be considered related, and treated like valid INDs. That is, they are then passed on to the other heuristics, filtering out spurious INDs, and then used in the $\mathsf{FIND}_{\mathsf{H}}$ algorithm.

While this heuristic works well for many cases (see Experiment 7, Sec. 7), the underlying assumption is that related tables have some data in common. With this simple scheme, a discovery of "relatedness" is possible if there is some extensional overlap between the relations to be compared. If the relations have no tuples in common, the use of set-difference queries is not meaningful for the discovery of relationships.

7 Experiments and Evaluation

7.1 Experimental Setup

Experiments were performed on several Linux-PCs with a dedicated machine running a relational database server. We obtained data from the UC Irvine KDD Archive (`http://kdd.ics.udi.edu`), specifically subsets of the CUP98, CENSUS, INSURANCE, and INTERNET data sets, which (converted into relational tables) had between 40 and 90 attributes each.

In order to "discover" inclusion dependencies, we used different projections and selections of each dataset and compared those to each other. An interesting feature of some of the data sets is that they have very small domains, as many of their attributes are categorical data. Furthermore, they are encoded as small integers, such that many unrelated attributes match each other (i.e., form spurious unary INDs). While one could "join" those columns with their "dimension tables" and obtain distinct domains, we left the tables as they were as a challenge to our algorithms. The effect was a high number of spurious INDs, which we could use to assess the performance of our solution.

7.2 Experiment 1: Comparison with Alternative IND Discovery Techniques

The performance of the FIND_2 algorithm (without heuristics) was compared with the previously published Apriori-like IND discovery algorithm [10], which serves as the baseline algorithm for this problem (and can be faster for very small problems). The latter algorithm was implemented in the same environment (Java over relational DB) as the FIND_2 algorithm. The test case consisted of a set of selections of the CENSUS table, with 500 rows each. Each table had 41 attributes. The total number of INDs between those tables varied, and the size of the largest IND between any of the tables tested also varied, between 5 and 16 attributes.

As can be seen from Fig. 7, the runtime of the FIND_2 algorithm is substantially shorter than that of the Apriori-like algorithm. The runtime recorded represents CPU time only; the number of database queries is also lower for the FIND_2 algorithm than for the Apriori-algorithm. As expected, the latter algorithm shows exponential runtime behavior in the size of the largest IND in the solution (note that the y-Axis is logarithmic). The runtime of FIND_2 is not affected by the size of the largest IND.

On the other hand, the runtime of FIND_2 does depend on the *size of the solution*, i.e., the number of distinct maximal INDs in the result (Fig. 8 shows data and linear regression curve; the correlation coefficient is $r^2 = 0.95$). As explained in Section 5, the size of the true solution is often small, but can be increased greatly by spurious INDs. Even though many of those spurious INDs are eventually purged from the search space in later phases of the algorithm, they slow down IND discovery significantly, and can even lead to aborted discovery runs due to memory problems. Here, using the heuristics proposed in this paper can help to speed up IND discovery.

7.3 Experiment 2: Performance and Quality Effects of Heuristics

This experiment was conducted to assess the runtime of the algorithm and the quality of its output for a given data set, with and without the use of heuristics. For this experiment, we used a 5000-tuple random subset CENSUS1 of data set CENSUS and a further random subset of 4500 tuples (90%) of CENSUS1, called CENSUS2 (i.e., CENSUS2 $\subset$ CENSUS1 $\subset$ CENSUS). This choice was made to emulate a certain randomness in real-world data. We compared the performance and quality of algorithms FIND_2 and FIND_H. We used different projections on those tables, which all originally have 41 attributes. Figure 9 shows the runtime of algorithms FIND_2 and FIND_H, for different size projections, illustrating the large performance benefits of the heuristic strategy.

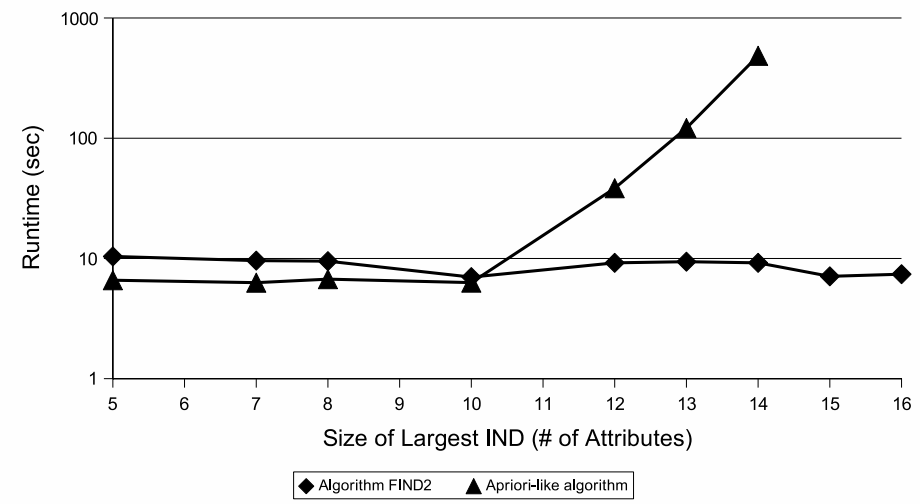


Fig. 7. Comparison of Algorithm FIND_2 with Apriori-like algorithm

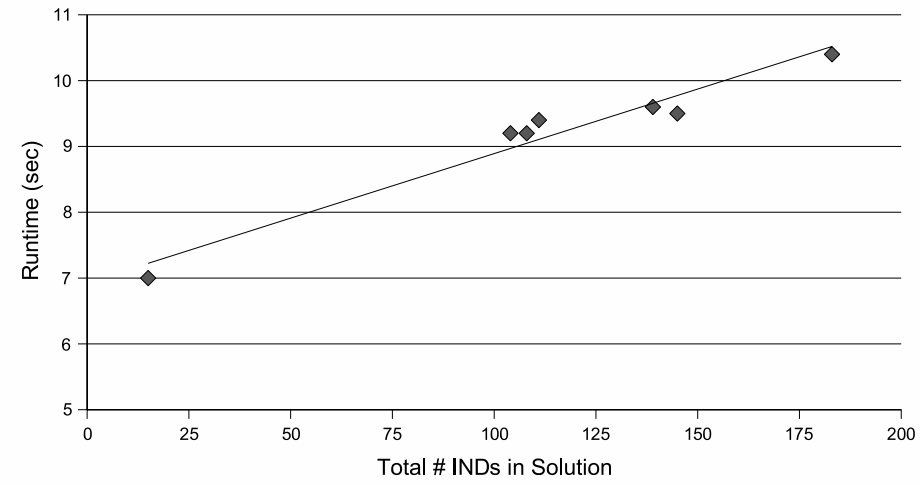


Fig. 8. Runtime Behavior of Algorithm FIND_2 under different size solutions.

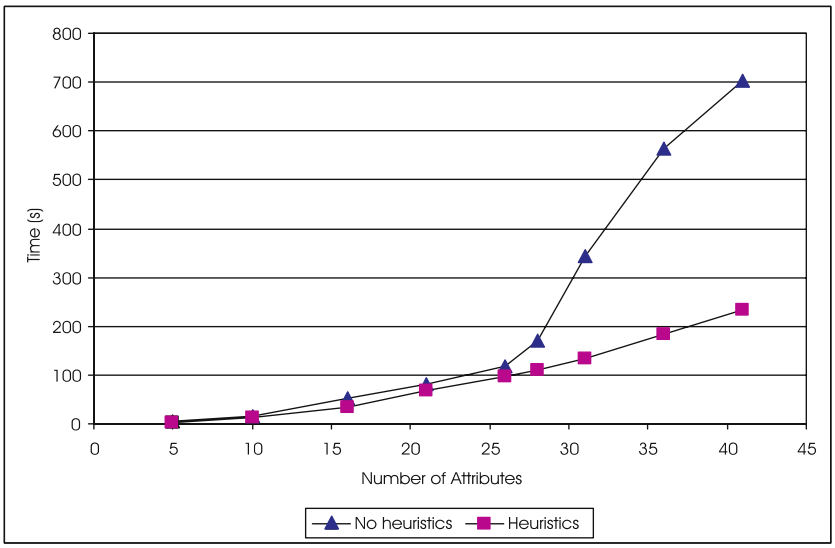


Fig. 9. Performance of algorithms FIND_2 and FIND_H, respectively, for discovering INDs between CENSUS2 and CENSUS1. The time for merging INDs is included in the runtime measurements.

There is a penalty in accuracy as a tradeoff for the lower runtime. The full cover of INDs is not found by the heuristic algorithm. Rather, FIND_H reports a maximum IND whose arity is about 70%-85% of the largest valid IND between the test data sets. However, through IND merging (Sec. 4.4), we still correctly find the largest IND in this data set. In other cases, the results of clique merging are not perfect as here, but still large INDs are found, as shown below.

7.4 Experiment 3: Assessing Result Quality for Heuristic FIND_H Algorithm

As explained in Sec. 5, the size of the largest IND discovered is a useful measure for the quality of the algorithm's performance. However, to assess the quality reduction of FIND_H, we conducted an experiment assessing the precision and recall of *unary* and *binary* INDs in the respective solutions.

For our test case of the CENSUS dataset, we recorded all unary and binary INDs that the heuristic and non-heuristic algorithms had considered and discovered, and compared with the true solutions.

See Table 3 for the results. The table contains the values for precision and recall for unary and binary INDs for both algorithms. Precision is computed in the usual manner as percentage of discovered INDs that are correct, while recall is computed as percentage of discovered INDs that are found by the algorithm.

In the test case, there was a single 41-ary IND to discover, such that the number of correct unary INDs was 41, and the number of correct binary INDs was $\binom{41}{2} = 820$. The

Table 3. Precision and Recall of Heuristic Algorithm FIND_H for Unary and Binary INDs

Algorithm	Unary INDs		Binary INDs	
	Precision	Recall	Precision	Recall
FIND_2	59%	100%	83%	100%
FIND_H	89%	100%	99%	97%

non-heuristic IND algorithm of course discovers all those INDs, but considers many more, spurious, INDs. In this experiment, the heuristics worked very well for unary INDs, achieving 100% recall and only considering very few INDs that turned out to be spurious. This is partially due to good performance of the AVD-heuristic which compares frequency distribution of values.

In other cases, the recall is not as good but still sufficient to discover large INDs efficiently. This experiment demonstrates the effect of the FIND_H algorithm: increase in precision of discovery of small INDs, at the expense of a reduction in recall.

7.5 Experiment 4: Effect of Low Numbers of Distinct Values in Data Set

In this experiment, we assess the quality of the heuristic algorithm in a data set with many spurious INDs. Table INSURANCE is such a data set, as it contains almost exclusively attributes with small integer domains (often less than ten distinct values) and consequently nearly 50% of its unary INDs are valid. For the full data set of 86 attributes, 4000 unary INDs are valid in the database, which would lead to a prohibitively large hypergraph with 4000 nodes.

In fact, the non-heuristic FIND_2 algorithm fails for this data set for all cases with more than 10 attributes, so no performance results for the non-heuristic algorithm can be reported for comparison.

Table 4 shows the quality achieved by the heuristic algorithm $\mathsf{CHECK}_\mathsf{H}$ for this case, for different size projections of table INSURANCE. Both the size of the largest IND found directly and the size of the largest *merged* IND are reported. The reason for the reduction in quality for larger relations is that in order for the algorithm to finish, we had to prune the search space by limiting the number of nodes and edges of the search hypergraph. The increase of quality for large relations may be due to the random projections of relations that were performed to obtain problem subsets.

The power of the IND-merging strategy (Sec. 4.4) becomes clear for very large relations, as the size of the largest discovered IND (relative to the size of the largest *existing* IND) actually increases.

7.6 Experiment 5: Number-of-Distinct-Tuples Parameter in DV Heuristic

The Distinct-Value (DV) heuristic rejects valid INDs as spurious when the number of distinct values is lower than a certain threshold n. A study of the statistic effects of this heuristic is given in Sec. 3.3. In this experiment, we varied the parameter n, whose

Table 4. Size of largest IND discovered relative to size of largest valid IND in a difficult case. In cases marked "N/A", the algorithm did not finish.

# of Attributes	Algorithm		
	Non-heuristic	Heuristic	Heuristic w/ IND-merging
10	100%	100%	100%
20	N/A	95%	95%
30	N/A	50%	50%
40	N/A	33%	33%
52	N/A	38%	38%
64	N/A	41%	44%
86	N/A	36%	53%

default value is 7, from 0 to 15. An $n = 0$ represents the exact (non-heuristic) algorithm. Table 5 shows the higher-arity INDs that the FIND_H algorithm found for a test case from the CENSUS experiment set, for different n.

For this experiment, there was a single true IND to be discovered, with 41 attributes (column 1). Two effects are apparent: First, the size of the largest IND discovered by the heuristic algorithm FIND_H decreases as the DV heuristic declares more and more small INDs spurious. Second, the algorithm also discovers more INDs, with different sizes, such that the solution becomes "spread out". For $n > 7$, the solution quickly deteriorates, as predicted by the theory. Note that for this experiment, IND-merging (i.e., computing the union of the attribute sets in all discovered INDs) yielded the "true" solution of a 41-ary IND.

Furthermore, the algorithm actually becomes slower for larger values of n. One reason is that the size of the discovered solution (i.e, the number of minimal unique INDs) increases (see also Fig. 8). Another reason is that the (time-consuming) AVD-heuristic (Sec. 4.2) is used more often as the DV-heuristic declares more INDs spurious (since the AVD-heuristic is applied to INDs rejected by the DV heuristic).

7.7 Experiment 6: Accuracy of the χ^2-Test and the AVD Heuristic

The *attribute value distribution (AVD) heuristic* relies on the assumption that attributes that stand in an inclusion relationship to one another are semantically related and thus show a similar distribution of their values. This will be true if the two relations in question are actually random samples of some larger real-world data set. However, if algorithm FIND_2 is run on two relations R and S, with one or both of R and S being selected from a larger set D on a predicate ($R = \sigma_{C_1}(D) \vee S = \sigma_{C_2}(D)$), the value distribution in some attributes in R might be different from the value distribution in some attributes in S.

Thus, we performed a number of experiments in which we generated subsets of our data sets using *predicates* rather than random sampling. The expectation is that the AVD heuristic will produce many false negatives in the presence of such predicates, which

Table 5. Large INDs discovered with Different Thresholds for the DV Heuristic

IND Size	Declare IND spurious if $n \leq$						
	0	4	6	7	8	10	15
≤ 24							4
25						1	1
26							3
27							1
28						2	4
29							2
30						1	2
31					1		
32				2	1	6	
33			1	2	5	2	
34			1	2	4		
35			4	1			
36			2	1			
37		2					
38							
39		2					
40							
41	1						
Runtime (sec)	650	381	356	334	388	408	455

motivates the design to only run this heuristic after the DV heuristic has already rejected an IND (Sec. 4.3).

Table 6 shows the quality (ratio of size of largest IND found to size of largest existing IND) of the result in data set INTERNET for four different predicates. The data set represents a survey in Internet usage data, and we selected the following four attributes for predicates: *gender*, *household income*, *country/state of origin* (encoded in a single attribute in the original data source), and *major occupation*, with conditions that had selectivities between 0.45 and 0.8. For example, selecting tuples with a predicate such as `GENDER<>'female'` will change the value distribution of the values in the other columns if they are gender-specific. Likewise, a predicate such as `HOUSEHOLD_INCOME<75,000` will probably change the value distribution in the other columns of this table, which represents an Internet usage survey.

We performed similar experiments with our other data sets and found that the AVD heuristic helps to find between 50% (data set CUP98) and 10% (data set INSURANCE) larger INDs than the algorithm with only the DV heuristic, averaged over several different predicates. This experiment shows that using the AVD heuristic gives better results (i.e., more accurate large INDs) in most of our experimental cases in which it was actually applied. It never reduces the quality of the result due to the way it is used in algorithm $\mathsf{CHECK}_\mathsf{H}$.

Table 6. Relative size of largest discovered IND, with subsets selected by predicate.

Attribute	Predicate	with AVD	without AVD
GENDER	`<>'female'`	81%	43%
HOUSEHOLD_INCOME	`< 75000`	94%	43%
COUNTRY	`='US' AND state <= 'North Carolina'`	85%	42%
MAJOR_OCCUPATION	`<>'other'`	88%	43%

7.8 Experiment 7: Discovering Non-exact Relationships

In this experiment, we tested the hypothesis established in Sec. 6, that the FIND_H algorithm can be used to discover approximate relationships between tables that are not exact inclusions of one another.

Our test case consisted of data from the US Census database[4], with the goal of letting the algorithm discover that the census data from two small states (North and South Dakota) are related. As explained in Sec. 6, due to the use of set-difference queries at the lowest level, the algorithm in its current form can not be used to compare distinct relations; some overlap is required. We therefore generated two overlapping tables by introducing tuples of each state's microcensus table into the other. We obtained two tables with about 5,000 tuples each, which had an intersection of about 3,500 tuples, and about 1,500 unique tuples each. For this experiment, we then projected those tables to 15 randomly selected attributes. We then let the FIND_2 and the FIND_H algorithms attempt to discover the one-to-one attribute correspondence between the two tables (i.e., a single 15-ary inclusion-dependency-like pattern, which implies 15 unary and 105 binary patterns).

While the non-heuristic FIND_2 algorithm generated 126 unary and over 6000 binary patterns, and subsequently did not finish, the FIND_H algorithm performed very well. It generated only 37 unary and 275 binary patterns (with the DV and AVD heuristics in place), well within the capabilities of the clique-finding algorithm. It finished after 248 seconds, and found a 13-ary relationship between the two input tables (after merging). The two attributes not found both had only 2 values in their domains and were highly correlated, making them indistinguishable for the algorithm.

This experiment suggests that the FIND_H algorithm can be used to discover relationships between database even in the presence of substantial noise, or even if the tables only partially overlap rather than form subsets of one another.

8 Related Work

There is substantial work on the discovery of patterns in databases. Much work is concentrated on functional dependencies (FDs), such as Lim and Harrison [21].

[4] One-percent microcensus:
`ftp://ftp2.census.gov/census_2000/datasets/PUMS/`.

An important related paper is by Kantola, Mannila *et al.* [2]. The authors describe an algorithm for discovering functional dependencies and also mention inclusion dependencies. However, no algorithm for IND discovery is given, and only a very rough upper bound for the complexity of the IND-finding problem is presented (in addition to a proof of NP-completeness of the problem).

Much database pattern discovery uses the concept of *levelwise search*, which has a well known instantiation in the *Apriori*-algorithm for association rule mining [11]. Mannila and Toivonen [12] give a theory of levelwise searches, and introduce the concept of *borders of theories* for discovery algorithms.

Zaki [22] uses levelwise search as well as the idea of cliques (but not hypercliques) for association rule mining. In this paper, the author also mentions *clique-merging*, which is similar to our *IND-merging*.

Hypergraphs have been used in other areas of databases and data mining. For example, Mannila and Räihä [3] give an algorithm for the discovery of functional dependencies that maps the problem to a hypergraph traversal.

Inclusion dependencies have been widely studied on a theoretical level. Fundamental work is done by Casanova, Fagin and Papadimitriou [1]. They present a simple axiomatization for INDs. While their work focuses on inference of INDs, not discovery, we use their "projection and permutation" axiom as the basis for the FIND_2 algorithm. Casanova et al. further prove that the decision problem for INDs (i.e., deciding whether a given IND can be derived from a given set of INDs through inference) is PSPACE-complete. Chandra and Vardi [23] prove undecidability of the problem. Mitchell [24] developed inference rules for INDs. No discovery on the data-level is mentioned in that body of work.

De Marchi *et al.* first proposed a levelwise algorithm for IND discovery [10]. The algorithm is competitive for very small problems, especially due to the use of an inverted index for unary IND discovery, but suffers from the dimensionality curse for IND sizes beyond about 8. More recently, deMarchi *et al.* proposed the *Zigzag* algorithm [14] which is very similar to the FIND_2 algorithm presented by the authors in [9,13]. There are significant differences such as the hypergraph model (we use k-uniform hypergraphs vs. de Marchi's general hypergraphs) and the discovery algorithm (our hypercliques vs. de Marchi's minimal hypergraph traversals). In addition, de Marchi treats invalid large IND candidates (such as c_2 in Fig. 2) differently from us, by attempting to validate them by removing single attributes. The choice of strategy is guided by a heuristic based on the number of tuples violating the IND property in the proposed IND. His ideas are orthogonal to ours, and we expect that a pooling of ideas might lead to an overall more optimized algorithm. In any case, the results from this paper would apply equally to FIND_2 and *Zigzag*.

There is substantial related work on the mathematical foundations of some of the heuristics that we have used to restrict problem spaces in our algorithm. Work on the theory of attribute value distributions can be found in [25] and [26]. The statistical χ^2-Test itself is described in statistics textbooks such as [20].

Schema integration is not limited to the discovery of INDs. In fact, there is a very large body of work in meta-data driven (as opposed to data-driven) schema integration.

Rahm and Bernstein [8] give an overview over some recent schema-integration projects; an earlier survey is [27].

Larson *et al.* [28] give a theory in which they infer attribute equivalence by a variety of indicators, such as domains, maximal and minimal values, and some constraints imposed by the (relational) database system. Their work is complementary to ours in some sense but ignores the actual data inside the attributes. Therefore, it is very sensitive to the availability and correctness of their assumed constraints.

More ideas on schema matching are contained in the *SemInt* project [29], in which attribute equivalence is inferred based on 20 different features of an attribute, five of which (minimum, maximum, average, coefficient of variance, standard deviation) are based on data but represent very simple properties and apply only to numeric attributes. These 20 dimensions are then used to train a neural network classifier for inferring attribute relatedness. Doan et al. [30] use a similar machine-learning approach to infer related schema elements in semistructured databases.

Kang and Naughton [31] present another schema matching approach, in which they map each of two relations into a graph and then perform graph matching to achieve schema matching. They use the assumption that attributes with similar entropy are related and also take intra-relational mutual information of attributes into account. The entropy heuristic applies to all data types and is somewhat related to our AVD measure, but is only a one-dimensional measure which incurs many false positives. The authors report that their approach does not scale beyond 15–20 attributes due to the deterioration of their heuristic.

9 Conclusion

In this paper, we have proposed heuristics that help to scale hypergraph-based inclusion dependency discovery algorithms [13,14]. We have shown that significant performance benefits are possible by applying the concept of *spurious* IND. This concept is used to reduce the problem size for exponential-complexity algorithms. This strategy makes it possible to automatically discover overlaps between almost any pair of real-world size relations. Even relations with many meaningless single-attribute overlaps (introduced by domains with few and accidentally identical values between those attributes) can be used for robust discovery.

Applications of this work lie in database integration (particularly, schema matching), reorganization, and query optimization. It could also be potentially beneficial in other application domains, since exponential-complexity mapping problems are common in subset and similarity discovery problems.

A potential direction into which to take this work is a further generalization of the problem, moving away from the discovery of *exact subsets* between relations and towards true *similarity*. This would entail relaxing the assumptions (1) that all tuples in the "included" relation actually exist in the other and (2) overcoming the problem that values across the attributes must match exactly for an inclusion dependency, both of which are receiving some attention in the research community already While the first problem is addressed in this paper and the $\mathsf{FIND}_{\mathsf{H}}$ algorithm can be used for discovery tasks in this category, see for example [31] for the second problem.

References

1. Casanova, M.A., Fagin, R., Papadimitriou, C.H.: Inclusion dependencies and their interaction with functional dependencies. In: Proceedings of ACM Conference on Principles of Database Systems (PODS). (1982) 171–176
2. Kantola, M., Mannila, H., Räihä, K.J., Siirtola, H.: Discovering functional and inclusion dependencies in relational databases. International J. Of Intelligent Systems **7** (1992) 591–607
3. Mannila, H., Räihä, K.J.: Algorithms for inferring functional-dependencies from relations. Data & Knowledge Engineering **12** (1994) 83–99
4. de Marchi, F., Lopes, S., Petit, J.M., Toumani, F.: Analysis of existing databases at the logical level: the DBA companion project. SIGMOD Record (ACM Special Interest Group on Management of Data) **32** (2003) 47–52
5. Lee, A.J., Nica, A., Rundensteiner, E.A.: The EVE approach: View synchronization in dynamic distributed environments. IEEE Transactions on Knowledge and Data Engineering (TKDE) **14** (2002) 931–954
6. Gryz, J.: Query folding with inclusion dependencies. In: Proc. Intl. Conf. on Data Engineering, IEEE Computer Society (1998) 126–133
7. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. Proceedings of IJCAI (2003) 16-21
8. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal: Very Large Data Bases **10** (2001) 334–350
9. Koeller, A.: Integration of Heterogeneous Databases: Discovery of Meta-Information and Maintenance of Schema-Restructuring Views. PhD thesis, Worcester Polytechnic Institute, Worcester, MA, USA (2001)
10. de Marchi, F., Lopes, S., Petit, J.M.: Efficient algorithms for mining inclusion dependencies. In: Proceedings of International Conference on Extending Database Technology (EDBT). (2002) 464–476
11. Aggarwal, C.C., Yu, P.S.: Online generation of association rules. In: Proceedings of IEEE International Conference on Data Engineering. (1998) 402–411
12. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1** (1997) 241–258
13. Koeller, A., Rundensteiner, E.A.: Discovery of high-dimensional inclusion dependencies. In: Proceedings of IEEE International Conference on Data Engineering, Bangalore, India, IEEE (2003) 683–685
14. de Marchi, F., Petit, J.M.: Zigzag: A new algorithm for mining large inclusion dependencies in databases. In: 3rd Intl. Conf. on Data Mining, Melbourne, Florida, IEEE (2003) 27–34
15. Mitra, P., Wiederhold, G., Jannink, J.: Semi-automatic integration of knowledge sources. In: Proc. of the 2nd Int. Conf. On Information Fusion (FUSION'99), Sunnyvale, California (1999)
16. Beneventano, D., Bergamaschi, S., Castano, S., et al.: Information integration: The MOMIS project demonstration. In: International Conference on Very Large Data Bases. (2000) 611–614
17. Koeller, A., Rundensteiner, E.A.: Heuristic Strategies for Inclusion Dependency Discovery In: Proceedings of 3rd International Conference on Ontologies, Databases and Applications of Semantics (ODBASE) (2004) 891–908
18. Koeller, A., Rundensteiner, E.A.: Discovery of high-dimensional inclusion dependencies. Technical Report WPI-CS-TR-02-15, Worcester Polytechnic Institute, Dept. of Computer Science (2002)

19. Demetrovics, J., Thi, V.D.: Some remarks on generating armstrong and inferring functional dependencies relation. Acta Cybernetica **12** (1995) 167–180
20. Rice, J.A.: Mathematical Statistics and Data Analysis. 2nd edn. Duxbury Press (1995)
21. Lim, W., Harrison, J.: Discovery of constraints from data for information system reverse engineering. In: Proc. of Australian Software Engineering Conference (ASWEC '97), Sydney, Australia (1997)
22. Zaki, M.J.: Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering (TKDE) **12** (2000) 372–390
23. Chandra A.K., Vardi, M.Y.: The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput. **3** (1985) 671–677
24. Mitchell, J.C.: Inference rules for functional and inclusion dependencies. In: Proceedings of ACM Symposium on Principles of Database Systems, Atlanta, Georgia (1983) 58–69
25. Mannino, M.V., Chu, P., Sager, T.: Statistical profile estimation in database systems. ACM Computing Surveys **20** (1988)
26. Hon, W.C., Zhang, Z., Zhou, N.: Statistical inference of unknown attribute values in databases. In: Proceedings of International Conference on Information and Knowledge Management. (1993) 21–30
27. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18** (1986) 323–364
28. Larson, J.A., Navathe, S.B., Elmasri, R.: A theory of attribute equivalence in databases with application to schema integration. IEEE Transactions on Software Engineering **15** (1989) 449–463
29. Li, W., Clifton, C.: SemInt: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. Data and Knowledge Engineering **33(1)** (2000) 49–84
30. Doan, A., Domingos, P., Halevy, A.: Learning source description for data integration. In: Proceedings of the Third International Workshop on the Web and Databases (WebDB), Dallas (2000) 81–86
31. Kang, J., Naughton, J.F.: On schema matching with opaque column names and data values. Proceedings of SIGMOD (2003) 205–216

Aligning Ontologies, Evaluating Concept Similarities and Visualizing Results

Kleber Xavier Sampaio de Souza[1,2⋆], Joseph Davis[2], and Silvio Roberto de Medeiros Evangelista[1]

[1] Embrapa Informática Agropecuária
Caixa Postal 6041 CEP 13083-886 Campinas SP, Brazil
{kleber,silvio}@cnptia.embrapa.br
[2] The University of Sydney, School of Information Technologies
Madsen Building F09, Sydney NSW 2006, Australia
jdavis@it.usyd.edu.au

Abstract. Ontologies have been created for many different subjects and by independent groups around the world. The nonexistence of a commonly accepted and used general purpose upper-ontology makes it difficult to integrate these ontologies through merge and alignment operations. The majority of the algorithms proposed so far rely on syntactic analysis, disregarding the structural properties of the source ontologies. In our previous work, we proposed an alignment method that considers the structural properties of an upper-ontology constructed using a thesaurus and Formal Concept Analysis technique (FCA). We also analyzed the FCA's lattice structure and proposed a measure of similarity based on Tversky's model, which allowed us to identify closely related concepts in different source ontologies. In this paper, we apply the alignment method to ontologies developed for a completely different domain, and enhance the solution by providing a navigational aid for the lattice. It is well known that one of the main drawbacks of the application of FCA is that the resulting lattice soon becomes cluttered when the number of objects and attributes increases. The proposed solution is based on hyperbolic visualization and on structural elements of the lattice.
Keywords: ontology alignment, Formal Concept Analysis, lattice visualization, similarity measures

1 Introduction

The merging and alignment of ontologies has been receiving increasing attention from the research and development community. Even before the advent of the Semantic Web Initiative, proposed by Tim Berners-Lee [1, 2], the problem had already been addressed by database researchers in the search for better methods for schema integration.

In merging and alignment, both the organization and content of ontologies are important and they have to be addressed in any merge/alignment method.

⋆ Research supported by Capes-Brazil grant BEX0687/03-0

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 211–236, 2006.

The way current research approaches the problem varies greatly on the amount of attention given to each aspect of the problem. Some proposals, like the FCA-Merge method [3], builds the structure of the merged ontology based on an analysis of a set of documents referenced to by the source ontologies. The previously existing structure of these ontologies is disregarded. Others [4, 5], try to balance between content and structure in the construction of similarity measures to evaluate the proximity between concepts in different ontologies.

The necessity of content assessment arises from semiotics, because the symbols that we compose to form the ideas (concepts) contained in an ontology are arbitrary, in the sense that the association between a symbol and its referent [3] was established at a certain time and continued to be valid ever since. For instance, someone must learn that the set of elements of an alphabet *c-a-r* refers to something that has usually four wheels and is self propelled. In this paper, the definition of ontology is that they are *knowledge specifications of conceptualizations* [6], and are constituted of symbols (entities) and relations between symbols [4].

In a previous work [7], we proposed a method for the alignment of sub-domain ontologies using an upper-ontology based on a thesaurus. To the best of our knowledge, this was the first time that FCA was applied to ontology alignemnt. The formalization for the alignment was carried out via Formal Concept Analysis [8, 9], a data analysis technique based on lattice theory. Firstly, we embedded the thesaurus in FCA's formalism (details in Section 6). Then, because every entity in the original ontologies had been previously mapped to a set of thesaurus' terms, when the *concept lattice* is constructed, every node (concept) in the lattice represents a set of objects which share the same thesaurus' terms. As there is a mapping between the original ontologies and the new one, the concept lattice is an **articulation of two ontologies**, as defined by Kalfoglou and Schorlemmer [10]. We call this articulation an **upper-ontology**.

In this paper, we adopt the definitions proposed by Kalfoglou and Schorlemmer [10] for mapping, articulation, merging, and alignment of ontologies. In the ontology **mapping**, the vocabularies and axioms of ontologies A and B are put in correspondence (please, see Fig. 1(a)), in such a way as to preserve both the mathematical structure (e.g. partial ordering of elements) and ontological axioms. Ontology **alignment** is a pair of ontological mappings M_1 and M_2 (please, see Fig. 1(b)) between an intermediate ontology, called **articulation of two ontologies**, and the source ontologies. This articulation of ontologies is represented in this work by the upper-ontology.

In another work [11], we explained how the *join* and *meet* operations over the lattice could be used to process user queries in the form of a set of terms and AND/OR connectors. As Chaudron observed in [12], the concept lattice corresponds to a zero order logical language. This logical language arises naturally from Galois Connection among subsets of objects and attributes.

[3] The entity which the symbol refers to.

[4] This definition is essentially pragmatic, a reduction of the original Aristotelian idea, which dates back to ancient Greece, more than 2,300 years ago.

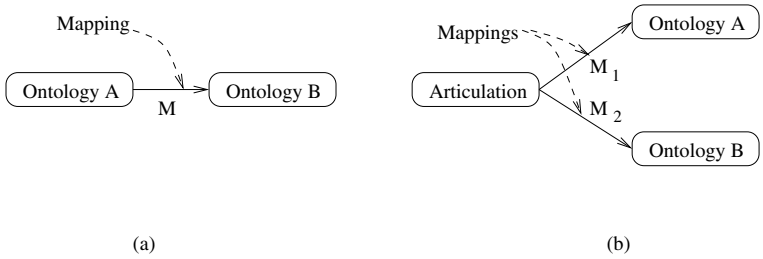


Fig. 1. (a) ontological mapping between Ontologies A and B; (b) Ontologies A and B are aligned with the Articulation Ontology

We also developed [13] a similarity measure based on Tversky's Model of Similarity [14]. Since the assessment of similarity requires the establishment of a common basis over which the judgment is realized, we used the concept lattice as a basis. The concept lattice is a rich structure from the point of view of the amount of information that it provides. Firstly, lattices are based on Powersets of a POSET. Secondly, because the lattice generated by FCA is a complete lattice, and complete lattices have, by definition, **least upper bound** (or supremum) and **greatest lower bound** (or infimum) for every pair of elements. This enables us to evaluate which concept subsumes other concepts juxtaposed in the hierarchy provided by the thesaurus.

As we would like to have a structural measure, not one which simply counts the number of common attributes, we selected the meet-irreducible elements of the lattice as a basis for our structural measure. An element is called **join-irreducible** if it cannot be written as a join of other elements. Similarly, an element is **meet-irreducible** if it cannot be written as a meet of other elements. There is an easy way to visually identify these elements in the lattice: join-irreducible elements are linked downwards by just one edge, whereas meet-irreducible are linked upwards by just one edge. The nodes marked in Fig. 2 correspond to meet-irreducible elements.

In this article, we apply the alignment method proposed in [7, 13] to ontologies developed for a completely different domain. This is intented as an indication that the method can be applied to any knowledge domain, although we do not prove it. We also propose a new form of visualization for the concept lattice. This new visualization is based on a special spanning tree for the lattice using the irreducible elements, which is rendered through a hyperbolic tree.

The remainder of the paper is as follows. In the next section, related work on ontology merging, similarity measures and concept lattice visualization is presented. Then, for sake of completeness, we repeat the main parts of the formalism presented in [7, 11, 13], associated with Formal Concept Analysis and Lattice Theory, used in the construction of the alignment method. A similarity measure is used in association with the alignment to evaluate the similarity between con-

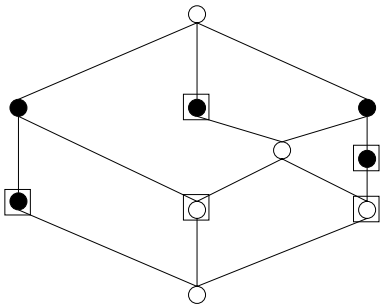


Fig. 2. Meet-irreducible (•) and join-irreducible (□) elements in a lattice

cepts in different ontologies, in the subsequent section. Finally, the visualization technique is proposed and conclusions are drawn.

2 Related Work

Ontology Merging and Alignment: The design and integration of ontologies have been addressed by many researchers [15, 16, 17]. They use heuristic rules to find appropriate matches among nodes in corresponding source ontologies. Although they have powerful features to support the user in the task of finding the best match for a given node, there still remains a lot of work that the user must carry out in order to produce a merged ontology. Kalfoglou and Schorlemmer [10] provide an excellent review on the subject. One of their conclusions is that the process of producing a fully automated method for ontology mapping has not been achieved by any of the proposed methods. Moreover, full automation of the actual mapping methods would lead to combinatorial explosion.

Schema Integration: Rahm and Bernstein [18] defined in their review on the subject that, database schemas are composed of elements connected by some structure. The choice of the elements and the structure of a schema depends on the particular representation chosen: objects, inheritance mechanisms, pointers and interfaces in OO models; entities and relationships in ER models; elements and ID references in XML; and nodes and edges in graphs. Several of these elements are also present in the definition of ontologies and, for this reason, some authors tend to see ontology merge and alignment simply as schema matching. However, as pointed out in [19] there are many differences between the two approaches. Nevertheless, these approaches share the majority of the techniques. For instance, in [20], Description Logics is used to formalize an enhanced ER model, and linguistic analysis is used to compare attributes existing in different models.

Information Integration: There is a large number of initiatives describing the use of ontologies in integration of information [21]. OBSERVER system [22], for example, explores syntactic relations among elements in ontologies (formal-

ized in Description Logics) to translate a query across multiple related ontologies. Our approach differs from the syntactic ones because the alignment of ontologies anchored in a thesaurus provides a structural rather than syntactical comparison between ontologies (details in Section 5).

Ontology Merging and FCA: Formal Concept Analysis has been applied to a number of domains, including ontology merging [9]. The FCA-Merge method uses a set of documents related to the two ontologies to be merged and processes them through natural language processing techniques, producing a pruned concept lattice. That lattice is then used for the generation of the final merged ontology. In our approach, the documents contained in the source ontologies are not re-processed to find their best classification in the aligned ontology. As of their original classification, they were already linked to the appropriate terms in the thesaurus and were associated with the nodes in the corresponding ontology [23].

FCA and Thesaurus: The formalization of botanical taxonomies with Formal Concept Analysis was studied in [24]. Another work associating thesaurus and FCA was reported in [25]. In this work, the association was structured to organize medical discharge summaries. None of the approaches, however, addressed the alignment of ontologies anchored on a thesaurus.

In our work [7], instead of merging the common corpus between the two ontologies to be merged (as in FCA-Merge), every term (nodes organized in a part-of relation) in the source ontologies is mapped into a term in an upper-ontology constructed using a thesaurus.

Similarity Measures and Ontologies: Rodriguez and Egenhofer [5] proposed an assessment of semantic similarity among entity classes in different ontologies. Their matching process is based on Tversky's measure of similarity [14] and uses synonym sets, distinguishing features and semantic relations of entity classes. Doan et al. [26, 4] proposed the application of machine learning techniques to create, semi-automatically, mappings between ontologies. In their model, they used a probabilistic distribution-based similarity measure called Jaccard coefficient. Like these approaches, our work deals with similarity measures to evaluate appropriate matches. However, our approach is different in that we are using a thesaurus to provide a common partial ordering over which the matches are evaluated, instead of a plain hierarchy of classes. In this work, we assume that the instances have already been classified in the thesaurus.

Visualization of Concept Lattices: FCA's concept lattice becomes rapidly cluttered as the number of objects and attributes increases. The solutions proposed so far are directed mainly at reduction of the lattice to a manageable size. The first approach, proposed in [9], is the use of *conceptual scales* in which subsets of the attributes are analysed separately. This approach is used in the Toscana System [27, 28]. Another approach is to apply data mining techniques and display only concepts having at least a minimum level of support. Support is introduced in the closure operator, producing the so called *iceberg concept lattices* [29]. With this technique, only the top-most part of the lattice is shown. It is a very interesting approach because it reduces the size of the lattice for very

large databases. Our approach differs in that it is based on a spanning tree for the lattice. However, as the spanning tree has the same number of concepts of the originating lattice, its size is also very large for large lattices. This difficulty is overcome with the hyperbolic visualization, discussed below.

Graph Visualization: the problem of lattice visualization is a special case of the general problem of graph visualization. In an excellent review on the subject focusing specifically on visualization applied to navigation in information, Hernan et al. [30] discuss the problems related to the size of the graph. Yet, they point out that the usability problem becomes an issue even when we are still able to identify all the nodes and edges displayed. Some of the techniques used to manage the size problem are distortion, such as *fisheye*, which enlarges the area being focused [31], and hyperbolic layout [32, 33], which distorts the spanning tree of a graph, increasing the focusing area, similar to what occurs with fisheye distortion. In our paper, we will adopt the hyperbolic layout. Please refer to Section 8 for details.

3 Overview of the Proposed Solution

The problem: given two ontologies developed for closely related domains, we would like to compare how similar they are, and visualize both of them in a common graphical structure. The graphical structure must support ontologies with hundreds of nodes, at the very least.

The background: in [7, 11, 13] we had ontologies designed for sub-domains of the agricultural domain. We developed an alignment method anchored in a thesaurus (Agrovoc), discussed how the lattice could be used as a unified view by a search engine, and designed similarity measures to identify cross-ontology related concepts;

The structure used in the analysis: we established as a requirement that the formalism employed in the alignment provided a good structural view of both ontologies, so that their commonalities and differences would be clearly visible, even without any mathematical similarity measure. It turned out that Lattice Theory, the basis of Formal Concept Analysis [5] (FCA) provided such a structural perspective. We had only to select the appropriate elements to display in the lattice;

Thesaurus terms as elements of the structure: the set of elements represents the characteristics of the objects that we are analysing. They can be either objects from the original ontologies, terms obtained from statistical analysis, or terms from a set of predefined elements, like a thesaurus, for example. Since all the information in the system had been catalogued using, among other information, terms contained in a thesaurus (Agrovoc [34]), and a thesaurus is a Partial Ordered Set(POSET) [27], we decided to align the ontologies using a thesaurus;

Similarity between Concepts: the lattice showing the result of the alignment gives some clues about which concepts are closer to or farther from

[5] Please see Section 5 for FCA applied to alignment of ontologies.

some given concept. However, as we would like to express mathematically this degree of sharing, we created a structural similarity measure to evaluate this precise degree of sharing;

The remaining problems: in our previous papers, the only way to compare graphically the objects and visualize the whole structure of the alignment was by inspecting the concept lattice. This task turned out to be hard whenever the lattice had more than fifty nodes. Here, we develop a visualization aid to support this analysis.

The first part of this paper (Sections 5 and 6) is dedicated to the definition of the common basis over which we perform a structural analysis, viz. the Galois Lattice obtained by the application of Formal Concept Analysis to the set of objects. In the second part (Sections 7 and 8), this analysis is used in the construction of a structural similarity measure, and in the development of a heuristic method for the visualization of the lattice.

4 A Motivating Example

In [7, 11, 13] we used ontologies designed for sub-domains of the agricultural domain, namely *Beef Cattle* and *Dairy Cattle*. For alignment purposes, we used a thesaurus that is widely known in the agricultural domain, the Agrovoc thesaurus. Here, the example that we are providing is more general, and has been proposed by Mitra et al. [35] for articulation of ontologies (please see Fig. 3), and later by Compatangelo et al. [20] for schema integration. Since it has been used in different (but correlated) approaches, we thought that it would provide a good indication of the generality of our proposed method.

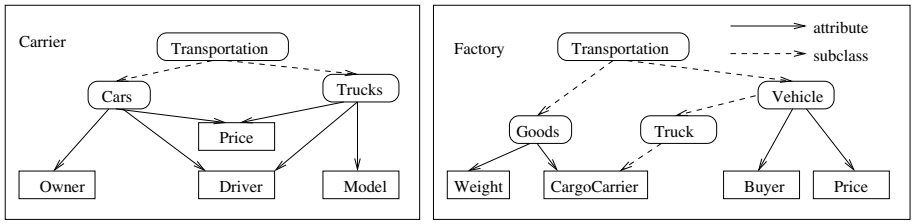


Fig. 3. Motivating example: fragments of ontologies *carrier* and *factory* (adapted from [35].)

It is a very simple simple fragment of two ontologies: *carrier* and *factory*. These ontologies define transportation from two different points of view: the one of those who use vehicles to transport goods, and the other of a manufacturer. In [35] these ontologies were aligned using a *graph-oriented model*, and in [20] they were aligned using Description Logics and linguistic analysis.

5 Formal Concept Analysis and Lattice Theory

Since it was first proposed in the early 1980's, Formal Concept Analysis (FCA), or Galois Lattice [8, 9], has been applied to many domains: from structuring information systems [27], to knowledge discovery in databases [29], political science, understanding building regulations and psychology [25]. FCA is a method for data analysis based on Lattice Theory and Propositional Calculus. It is especially suitable for exploration of symbolic knowledge (concepts) contained in a formal context, such as a corpus, a database, or an ontology.

Due to space limitations, we will avoid giving a detailed explanation of the FCA theoretical background. Please refer to [8, 9] for further information. Rather, we will include here only the essential definitions and theorems necessary for the understanding of this paper.

The *concept lattice*, resulting from the application of FCA to a matrix containing a set of objects and their associated attributes, structures the abstraction of concepts present in human thought in an elegant way. The concepts are classes of things having certain attributes. If a concept A is above a concept B in the lattice, and the two are linked, then concept A is more general than B and, as such, it inherits part of attributes of B. As a consequence, we can say that whenever B happens, A is also happening, which suggests a logical entailment. In the lattice, we cannot only see a hierarchy of concepts, but also the whole set of binary relations present among concepts. That makes the visual analysis of data superior to the one we can obtain by looking at a hierarchy of classes.

Definition 1 (Formal Concept). *Let O be a set of objects, A be a set of attributes and $R \subseteq O \times A$, a binary relation between O and A. A pair (E, I), with $E \subseteq O$, $I \subseteq A$ is a* **formal concept***, if, and only if, $E' = I$ and $I' = E$, where:*

$$E' = \{a \in A \mid \forall o \in E : (o, a) \in R\} \tag{1}$$

$$I' = \{o \in O \mid \forall a \in I : (o, a) \in R\} \tag{2}$$

The set of all formal concepts is called ***formal context****, denoted by (O, A, R).*

E is called the **extent** and I the **intent** of the formal concept (E, I). It can be seen from the definition above that E is the set of all objects that share the same attributes in I. Similarly, I is the set of all attributes that are shared by the same objects in E. The definition of E' and I', together with the restriction that $I' = E$ and $E' = I$, satisfy the necessary and sufficient conditions to provide a **Galois Connection** between E and I. These equations also establish a **subconcept-superconcept** relation, such that:

$$(E_1, I_1) \leq (E_2, I_2) \Leftrightarrow E_1 \subseteq E_2 \tag{3}$$

$$(E_1, I_1) \leq (E_2, I_2) \Leftrightarrow I_1 \supseteq I_2 \tag{4}$$

This partial ordering results in a complete ordering among all elements of the formal context (O, A, R), with corresponding *infimum (or meet)* and *supremum (or join)*. Moreover, this ordered set is a lattice, called **concept lattice** [9].

Theorem 1 (The basic theorem on concept lattices (adapted from [9])). *The concept lattice* $\underline{\mathfrak{B}}(O, A, R)$ *is a complete lattice in which infimum and supremum are given by:*

$$\bigvee_{j \in J} (E_j, I_j) = ((\bigcup_{j \in J} E_j)'', \bigcap_{j \in J} I_j) \tag{5}$$

$$\bigwedge_{j \in J} (E_j, I_j) = (\bigcap_{j \in J} E_j, (\bigcup_{j \in J} I_j)'') \tag{6}$$

Where J is the set of all elements in the lattice.

6 Associating Thesaurus and FCA to Construct the Alignment

To make the paper self-contained, we repeat here the main definitions and theorems proposed and proved in [7]. For further details, please refer to that paper.

After we selected FCA as the structural basis over which the assessment of similarity is going to be performed, we had now to embed appropriately the thesaurus terms as elements in the FCA formalism. The thesaurus, together with FCA, provides a unified view of the two source ontologies anchored in a common partial ordering.

It is interesting to note that, normally, this special procedure is not necessary in standard FCA. This is particularly so if the set of elements used as attributes is flat, i.e. does not contain any ordering among its elements, in which case Theorem 2 and Corollary 1 are not necessary. In our case, however, this ordering is essential because we use it to evaluate the most specific concept that subsumes any two given concepts.

The definition of thesaurus used in this paper is that, a **thesaurus** $\langle T, \succeq \rangle$ is a set of terms $t_i \in T$ organized in accordance with a partial order $\succeq$. It is organized into many sub-trees. Each sub-tree contains the term's definition, its hypernyms and hyponyms. Usually, thesauri also contain related terms and **use-for/used-for** relations, which establish that instead of using a term with a certain name, one should use another one considered standard. However, we do not explore this part in our work because we are only interested in the partial ordering of terms.

The embedding of the thesaurus in the lattice is realized in the following way: initially, each term $t_I \in T$ is transformed into one attribute $a_I \in A$ of the formal context (O, A, R). Then, the partial order $\succeq$ is guaranteed by requiring that the inclusion of a term implies the inclusion of all of its predecessors (hypernyms). This embedding is stated formally in Theorem 2.

Theorem 2. *Let $\underline{\mathfrak{B}}(O, A, R)$ be a concept lattice, $\langle T, \succeq \rangle$ a thesaurus of terms T embedded in $\underline{\mathfrak{B}}$, and (E_1, I_1) and (E_2, I_2) two formal concepts with $(E_1, I_1) \geq (E_2, I_2)$. If $a_1 \in I_1, a_2 \in I_2$ and $a_1 \succeq a_2$, then $a_1 \in I_2$ holds.*

Proof. From (4) above, $(E_1, I_1) \geq (E_2, I_2) \Leftrightarrow I_2 \supseteq I_1$. Therefore, if $a_1 \in I_1$ and $a_2 \in I_2$ and $a_1 \succeq a_2$, then $a_1 \in I_2 \square$.

The equation $a_1 \in I_2$ in Theorem 2 states that either a_1 and a_2 are in the same node of the lattice, or a_1 must come from a node above. This result holds even when three nodes are compared, as it can be seen in the following corollary.

Corollary 1. *Let a_1, a_2 and a_3 be attributes such that $a_1 \in I_1$, $a_2 \in I_2$, $a_3 \in I_3$. If $a_1 \succeq a_2$ and $a_1 \succeq a_3$, then $a_1 \in (E_2, I_2) \vee (E_3, I_3)$.*

Proof. From Theorem1, $(E_2, I_2) \vee (E_3, I_3) = ((E_2 \cup E_3)'', (I_2 \cap I_3))$. If $a_1 \in I_2$ and $a_1 \in I_3$, then $a_1 \in (I_2 \cap I_3)$.$\square$

The preceding corollary shows that if the term a_1 is a common attribute between two nodes and it is a hypernym of terms a_2 and a_3 in the thesaurus ordering, then it is an element of the least upper bound (or join) of these nodes. This means that the in the lattice a_1 is in a position nearer to the top than a_2 and a_3.

Having established the common ordering through which ontological similarities and differences can be observed, the *articulation of two ontologies*, which we call **upper-ontology** of ontologies can now be defined:

Definition 2. *Let O_1, O_2, A_1, A_2 and R_1, R_2 be the set of objects, attributes and relations of ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. The formal context representing the upper-ontology is defined by $\mathcal{O}_U = ((O_1 \cup O_2), (A_1 \cup A_2), (R_1 \cup R_2))$.*

6.1 Example Continued

Before we can perform the alignment, it is necessary to associate thesaurus terms with the objects encountered in the ontologies we would like to align. For the agricultural domain, we took advantage of the fact all objects had been previously classified using Agrovoc. For this example, however, this association was manually done, selecting terms from WordNet © thesaurus [6], and it is shown in Table 1. Another way of performing this classification is by using machine learning techniques, as described in [26, 4].

Whenever a term was selected from WordNet © we identified the ordering all the way up to the root node in this thesaurus, in such a way to satisfy Theorem 2. The partially ordered set identified is shown in Figure 4.

Table 1 shows the formal context aligning part of the ontologies Carrier and Factory. The Objects correspond to rows in the table and Attributes, to columns. Whenever there is a relation between an object and an attribute, the intersection is marked in the table with an X. Objects relating to Factory ontology are marked with an A before the name, and to Carrier ontology with a B.

[6] WordNet: a lexical database for the English language. Cognitive Science Laboratory, Princeton University. URL http://www.cogsci.princeton.edu/~wn/

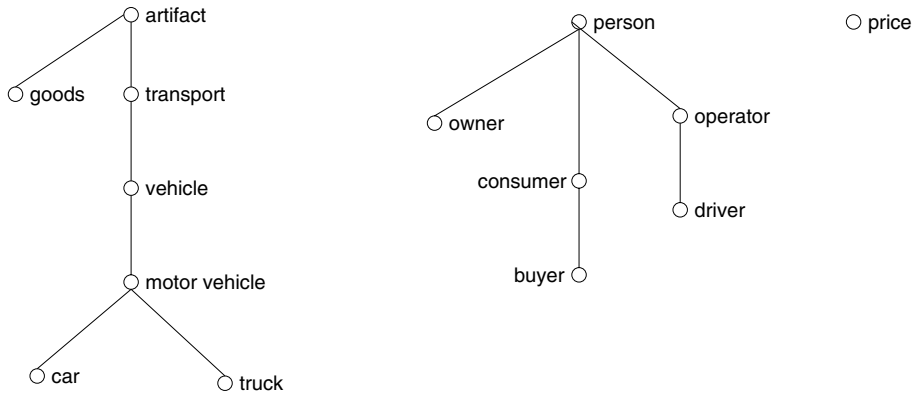


Fig. 4. Partially ordered set selected from WordNet © thesaurus regarding the carrier-factory alignment example.

	transport→artifact	driver→person	person	vehicle→transport	truck→motorVehicle	car→motorVehicle	motorVehicle→vehicle	buyer→person	owner→person	price	goods→artifact	artifact
A transportation	X											X
A vehicle	X		X	X				X		X		X
A buyer			X					X				
A price										X		
A truck	X			X	X		X					X
A cargo carrier	X			X	X		X					X
A goods	X			X	X		X				X	X
B transport	X											X
B cars	X	X	X	X		X	X		X			X
B trucks	X	X	X	X	X		X					X
B driver		X	X									
B owner			X						X			
B price										X		

Table 1. Alignment formal context for Factory (A) and Carrier (B) Ontologies.

The Hasse diagram corresponding to the formal context displayed in Table 1 is shown in Fig. 5. The names near each node correspond to WordNet thesaurus'

terms and the names in boxes are objects of ontologies A and B, respectively. The objects positioned at a certain node of the diagram inherit all the attributes of the nodes in the path from it to the top node. Thus object `A CargoCarrier`, for example, is linked in its context to `truck`, `motorVehicle`, `vehicle`, `transport` and `artifact`.

To illustrate how the thesaurus was correctly embedded in the lattice (as predicted in Theorem 2), consider the sequence:

`truck`→`motorVehicle`→`transport`→`artifact`

The term `motorVehicle` in Fig. 5 is placed in a concept above the one corresponding to `truck`, `vehicle` is above `motorVehicle`, and `transport` is above `vehicle`. The same happens with `person` and its specializations `buyer`, `driver` and `owner`.

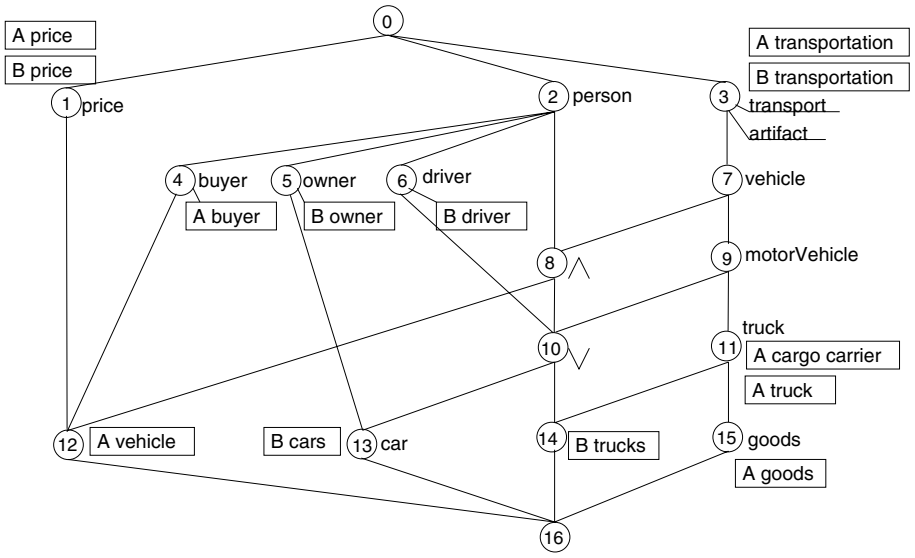


Fig. 5. Hasse Diagram corresponding to the formal context of Table 1.

One indication that two objects of source ontologies are close to each other is that they share most of the same attributes, i.e. the closer their intent, the closer the objects are regarding the concepts they represent. `A transportation` and `B transportation` could be merged, the same happening to `A price` and `B price`.

The objects `A truck` and `B truck` are very close to each other. They share the attributes `truck`, `motorVehicle`, `vehicle`, `transport` and `artifact`, but `B trucks` has two more attributes: `driver` and `person`.

The operation of meet ($\wedge$) and join ($\vee$) allows us to evaluate the exact degree of sharing, in terms of attributes, that any two objects have. For example, `(B`

`cars` ∨ `B trucks`)[7] gives as result the node above their intersection upwards, which is identified by (∨) (see Fig. 5). These two objects share many attributes, i.e. `motorVehicle`, `vehicle`, `transport`, `artifact`, `driver` and `person`.

However, there are attributes which belong only to one of the objects, like `truck` in the case of `B trucks`, and `owner` in the case of `B cars`. These commonalities and differences motivated us to introduce measures of similarity to express mathematically the precise degree of sharing.

This does not mean, as one might think however, that the alignment only exists when we have such measures. The alignment was concluded when we generated the **concept lattice**, because, it represents the **articulation of the two ontologies**, as stated in Section 2. It is a single structure containing objects of two (or more) ontologies, and its **formal context** realizes the mappings M_1 and M_2 linking the objects in the formal context with original ones in the source ontologies.

7 Using the Alignment to Evaluate Similarity Between Concepts

The assessment of similarity occupies a central role in the cognition process [36, 37]. For example, we cannot say much about `trx5226` unless we are told that it is a truck. Once we know this, we can make inferences like: it will serve as means of transportation, we can use it to deliver goods, and so on. In the inference process, we are using our knowledge of trucks in general, the purpose they are used for, and making a similarity assessment between this kind of truck and other vehicles. During inference and judgment of similarity, what we are trying to do is to categorize, as precisely as possible, a recently known concept, viz. *trx5226*.

The relation between categorization and knowledge is bidirectional [38]. Knowing that trx5226 shares some properties with other trucks enables us to categorize it as a truck. Conversely, knowing that trx5226 is a truck, enables us to infer that it can be used to transport goods, like other trucks. Therefore, when we use a similarity measure, we expect it to support the inference process as well.

7.1 Models of Similarity

There are many models of similarity. They can be broadly divided into two main groups [39]: continuous metric space models and set-theoretic matching models. One example of the former is the Shepard Model, which is based on probabilistic distributions. The latter group, which we will be using in our work, can be further subdivided into geometric, transformational, featural and alignment-based

[7] We are abusing of the notation here. The correct would be (13 ∨ 14), because these are the identifiers of the nodes, and the operation ∨ is defined only for lattice nodes. However, we do so because `(B cars ∨ B trucks)` will not require the reader to go to the lattice and find the corresponding objects.

models. *Geometric* models are based on distances (calculated in n-dimensional space) between vectors representing the characteristics of an entity, viz. every attribute is marked as zero/one in that vector indicating its presence/absence of that characteristic. *Transformational* models are based on the number of transformations required to make two entities equal, viz. the DNA sequence ACCG requires two transformations to become ACGA. *Featural* models, consider the sets of common as opposed to distinctive features. One example is Tversky's ratio model [14], given in equation 7, where A and B are the set of features of a and b, respectively, f denotes a measure over the feature sets, $(A-B)$ represents the set of features present in A but not in B and $(B-A)$, those present in B but not in A.

$$S(a,b) = \frac{f(A \cap B)}{f(A \cap B) + \alpha f(A-B) + \beta f(B-A)} \tag{7}$$

The parameters α and β were introduced in the model because Tversky observed in psychological experimentation that the assessment of similarity is not symmetrical. One example usually cited is that people consider North Korea to be more similar to China than China to North Korea.

In *alignment-based* models [40], structural parts that are placed in correspondence influence more than those parts which cannot be aligned. For example, if an entire sub-tree of a tree is identical to a sub-tree in another hierarchy, we can say that they are structurally aligned. Gentner and Markman [40] argue that because people focus on alignable differences rather than on nonalignable ones, the former has a greater impact on similarity assessment. As a result, people have found it easier to enumerate differences between *motel* and *hotel* rather than between *magazine* and *kitten*. This may also explain why we find aliens more realistic in science fiction movies if they have head, two arms, two legs, mouth and teeth, all of them structurally positioned in correspondence to what we are accustomed to in intelligent beings, viz. the mouth is located in the head and the head is in the upper part of the alien. We make the correspondence one to one (structurally) and state that the alien has a double set of teeth instead one, has a brain larger than ours, and so on.

7.2 The Structural Similarity Measure

One important fact about Concept Lattices (proved in Theorem1) is that the infimum (meet) and supremum (join) between every pair of objects is defined in terms of the usual set operators ($\cap$, $\cup$, $\subset$, $\supset$). Moreover, the supremum of two elements serves as a basis of comparison between them because it contains all the common attributes of these two elements. For example, in Fig. 5, the objects `A vehicle` and `B trucks` have in common the attributes (**join**) `person`, `vehicle`, `transport` and `artifact`, because the concept corresponding the supremum of the formers, identified as ($\vee$), has these attributes as its intents.

However, as we would like to have a true alignment measure, it should not be based on a common set of attributes, but rather on a common set of structural

elements of the lattice. Yet, those structural elements really exist. They are called join-irreducible and infimum-irreducible elements, defined in Section 1.

Meet-irreducible elements play an important role in our similarity measure. As we commented in Section 6.1, attributes in the lattice are introduced from the top to the bottom. Every meet-irreducible element corresponds to one new attribute being added, although the opposite is not necessarily true. In Fig. 5, the node 9 is meet-irreducible and introduces the thesaurus attribute `motorVehicle`, whereas the node 13 introduces the attribute `car` but is not meet-irreducible. That happens because `car` occurs only in conjunction with the attributes `onwner`, `driver` and `motorVehicle`. It does not occur in isolation. For this reason, `car` does not add any relevant information to the lattice and could, therefore, be eliminated without any loss of structural information. The lattice could be completely reconstructed without the presence of the `car` attribute. That is why meet-irreducible elements are so important from the point of view of attributes, viz. we can identify which attributes are really structurally necessary.

In [13], we developed a structural similarity measure using the meet-irreducible elements of the concept lattice. Tversky's similarity model was used as a basis for this measure, because of its conformance with an information-theoretic definition of similarity [41], its application in Computer Science [5] and its psychological experimental confirmation. This similarity measure is defined in Eq. 8.

$$S(a,b) = \frac{|(a \vee b)^{\wedge}|}{|(a \vee b)^{\wedge}| + \alpha|(a-b)^{\wedge}| + (1-\alpha)|(b-a)^{\wedge}|} \tag{8}$$

In (8), the set of common features is given by the set of common meet-irreducible elements. Using lattice operations (join ($\vee$) and meet ($\wedge$)), the set of common meet-irreducible elements is given by the meet-irreducible elements which are intent of $a \vee b$. This set is represented as $(a \vee b)^{\wedge}$.

As set of distinctive features, we considered both the set of meet-irreducible elements which are in a, but not in b, represented as $(a-b)^{\wedge}$, and the set of meet-irreducible elements which are in b, but not in a, represented as $(b-a)^{\wedge}$.

Instead of varying the parameter α in accordance with the relative depths of the nodes, as in [5], we left α fixed in 0.5. That means that we designed our measure as a symmetrical one, i.e. a is similar to b in the same measure as b is similar to a. The example below show the calculation of the similarity measure between the concepts $a = 12$ and $b = 14$, whose attached objects in Fig. 5 are `A vehicle` and `B trucks`, respectively.

$$(a \vee b)^{\wedge} = \{2, 3, 7\} \tag{9}$$

$$(a-b)^{\wedge} = \{1, 4\} \tag{10}$$

$$(b-a)^{\wedge} = \{6, 9, 11\} \tag{11}$$

$$S(a,b) = \frac{3}{3 + 0.5 * 2 + 0.5 * 3} = 0.545 \tag{12}$$

Table 2 shows the result of the similarity measure between all concepts in concept lattice of Fig. 5. It is interesting to note that, as the similarity measure considers nodes of the lattice (the meet-irreducible ones), our measure actually measures the similarity between concepts, rather than between objects. This means that the similarity between `B trucks` and `A truck` is the same as between `B trucks` and `A cargo carrier`, because `A cargo carrier` and `A truck` are associated with the same node in the lattice.

Another point that is worth mentioning is that we are counting the number of meet-irreducible elements of the lattice and not the number of attributes. As a consequence, the node containing attributes `transport` and `artifact` count as one node only.

node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	0	0	0	0	0	0	0	0	0.29	0	0	0
2		1	0	0.67	0.67	0.67	0	0.5	0	0.33	0	0.33	0.29	0.29	0
3			1	0	0	0	0.67	0.5	0.5	0.33	0.4	0.33	0.29	0.29	0.33
4				1	0.5	0.5	0	0.4	0	0.29	0	0.57	0.25	0.25	0
5					1	0.5	0	0.4	0	0.29	0	0.29	0.5	0.25	0
6						1	0	0.4	0	0.57	0	0.29	0.5	0.5	0
7							1	0.8	0.8	0.57	0.67	0.57	0.5	0.5	0.57
8								1	0.67	0.75	0.57	0.75	0.67	0.67	0.5
9									1	0.75	0.86	0.44	0.67	0.67	0.75
10										1	0.67	0.6	0.91	0.91	0.6
11											1	0.44	0.6	0.8	0.89
12												1	0.54	0.54	0.4
13													1	0.83	0.54
14														1	0.72
15															1

Table 2. Similarity measure applied to formal concept lattice of Fig 5.

The dashed lines in Fig. 6 shows some[8] of the relations between concepts for which $S(a, b) > 0.5$. Besides the example above, there are also similarities identified between `A truck` and `B cars`, `A vehicle` and `B cars`, `A goods` and `B trucks`, and so on. This threshold establishes the degree of precision with which the alignment is being considered. Whenever we increase the threshold the number of identified matches decrease.

The last association, between `A goods` and `B trucks` may seem strange at first sight, because goods has no common ancestor with trucks in the thesaurus hierarchy. However, what is being identified here is the relationship between these two concepts. Goods, in ontology A exists in the association with `A CargoCarrier`, because the factory is concerned about who is going to trans-

[8] We did not plot the complete results of Tab. 2 because this would make the diagram difficult to read.

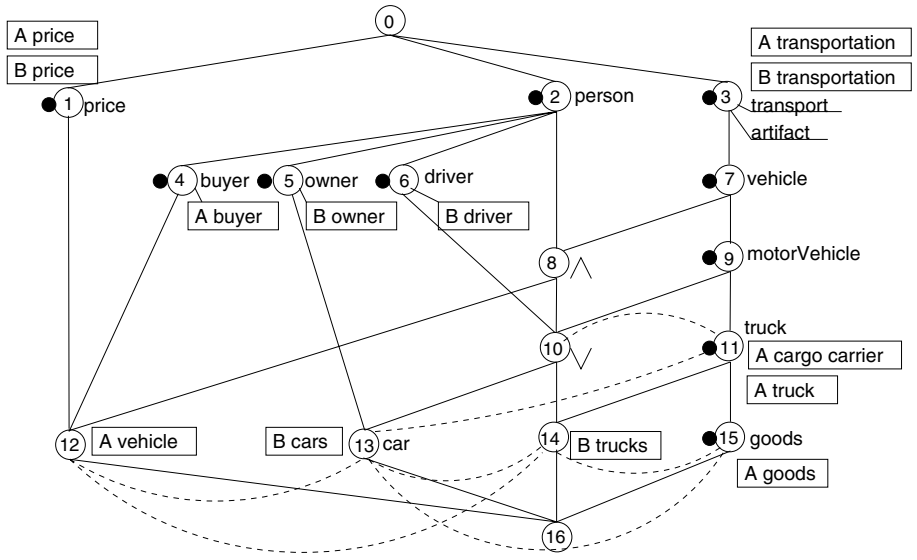


Fig. 6. Hasse Diagram displaying the meet-irreducible elements (marked with black dots) and some of the aligned nodes (dashed lines) sharing more than 50% of similarity.

port its goods. One advantage of FCA is that it allows one to break the whole hierarchy of sub-classes and reassemble it in a different way, where the relations are stronger. This can partially explain why it has been successful in software reengineering [42].

If we compare the results given by our approach with those obtained by [35] and [20], we can see that all the matches found in these papers have also been found here. For instance, `A vehicle`, `B trucks` and `B cars` have been associated with a common entity called `CarsTrucks` in [35], and `Car or Truck` in [20]. `A truck`, `A cargoCarrier` and `B trucks` have been associated with `VehicleCargoCarrier` in [35]. `CarsTrucks` corresponds to concept 7 (`vehicle`) in the lattice displayed in Fig. 6, and `VehicleCargoCarrier` corresponds to concept 11. Below concept 7 there are all objects that relate to vehicles in general, whereas below concept 11 there are only those related to cargo vehicles.

`A buyer`, `B owner` and `B driver` are all associated with `Person` in [35], whereas `A buyer` and `B owner` are associated with `Owner` in [20]. In our approach, `A buyer` has only a similarity of 50% with `A owner`, and this comes from the fact that both are specializations of `person`. This occurs because in WordNet, the buyer is not viewed as an owner (please see Figure 4), but rather as a consumer.

`A price` and `B price` have been aligned to `price` both here and in [35], and `A transportation` and `B transportation` have been aligned to `transportation` here and in [35].

8 Visualization Through Hyperbolic Trees

An important limitation of FCA is the visualization problem. As the lattice grows to more than fifty nodes, we are not able to visualize it comfortably. Ganter and Wille [9] suggested the use of *conceptual scales* to reduce the size of the lattice and show only a *slice* adequate for the analysis. A scale consists of a subset of the set of attributes and values for these attributes. Because it selects columns in the formal context matrix, this approach is often referred to as vertical [29].

Another way of reducing the size of the lattice is by producing an *iceberg concept lattice* [29]. This approach is called horizontal, because it uses Data Mining techniques and prunes the lattice below a certain threshold (the support level in Data Mining). That leaves us only with the upper part of the lattice, in this case, the part considered most relevant for the data analysis.

The approach we propose in this paper differs from these two in that we make a case not to reduce the lattice, but instead, we seek a visualization method that supports the data analysis that we perform over the entire lattice.

Lattice visualization is a special case of graph visualization. An excellent survey of graph visualization is presented in [30]. This survey is especially important for our work because it focuses on graph visualization from the perspective of information visualization. It discusses several methods and techniques used to deal with usability and discernability when graphs become very large. One of these techniques is fisheye distortion, which works like a lens, augmenting a certain region of the graph [31]. However, simply looking at something in detail does not eliminate all the existing crossings in the graph. Hernan et al. [30] also comment that reducing the crossings is a very important aesthetic factor.

The crossings can be eliminated by laying out a *spanning tree* for the graph. It is well known [30] that tree layout algorithms are simpler to implement, and because of their lower computational complexity, they also perform very well in real time. This is the approach that we have selected for our visualization heuristics.

Naturally, as we wish to display the totality of the nodes, the tree size has to be managed somehow. One alternative is to use Cone Trees, because they expand subtrees in 3D space, allowing for good management of the space available. Another technique is hyperbolic layout [32, 33], which distorts the spanning tree of a graph, increasing the focusing area, similar to what occurs with fisheye distortion. One point in favor of hyperbolic layout is that it eliminates the clutter present in Cone Trees. Due to its characteristics and ease of use, we adopted hyperbolic layout in our research.

After the selection of the visualization form, there still remains the problem of finding an appropriate spanning tree. The trivial solution would be the application depth-first or breadth-first strategy and linking all the nodes to the root

node. However, as we seek to propose a visual tool that supports data analysis, there are some additional requirements that our spanning tree and associated tool must satisfy:

1. When visiting a lattice node (concept), we often seek to identify its intent (attributes added up to that node) and its extent (objects that has these attributes), as defined in Theorem 1;
2. The meet operations in the lattice allows the identification of objects having certain characteristics. For example, in Fig. 5, we might want to know which objects relate `person` and `vehicle`. There are three objects that satisfy this query, namely `A vehicle`, `B cars` and `B trucks`;
3. There is also the inverse query (join), i.e. given a set of objects, which attributes are shared by them. For example, `B cars` and `A cargo carrier` have in common the attributes `truck`, `motorVehicle`, `vehicle`, `transport` and `artifact`.

These requirements demand more than a simple spanning tree. The second and third items require that we be able to solve, in real time, AND/OR operations over the lattice, and to find the corresponding concept. Nevertheless, it is intrinsic to the nature of transversalization that whenever there is a meet, one element stays in one sub-tree and the other in another sub-tree. Otherwise the cycles would make the construction of the tree impossible . For example, we may connect the node 10 to the node 6,8 and 9 in Fig. 5. If we connect to 8, we loose the meets $(6 \wedge 8)$, $(8 \wedge 9)$ and $(6 \wedge 9)$.

As well, there are also problems regarding the join operation. Given that a tree naturally joins its branches upwards, the join operation can be satisfied to a certain extent, but not completely, because when we eliminate the cycles, we also eliminate some of the joins. If we connect 9 and 10 in the example, we have the join $(10 \vee 11)$, but we loose the join $(12 \vee 10)$.

As the joins and meets are combinations of some structural elements, the best scenario that we can expect is to have in the same branch as many structural elements as possible. As discussed in Section 7, every new meet-irreducible element corresponds to a new attribute being added. It means that if we align on the same path as many meet-irreducible elements as possible, the corresponding tree will have branches with a high cohesion in terms of intent. This proximity in terms of intent is determined by our similarity measure, as explained in the previous section. So, we will use this similarity measure as an attraction force to determine which node a certain node should be connected to.

Figure 7 shows the spanning tree obtained from the application of the results in Table 2. In some cases, the attraction forces determined by the similarity measure between a node and two or more of its parents are the same. When this happens, the node could be connected to more than one parent. For example, we could connect either nodes 8 and 10, or 9 and 10 (please see Table 2). In this case, we based our decision on the number of alignable meet-irreducible elements. As node 9 has tree meet-irreducible elements in the way up to the top node, and node 8 has only two, we connect node 10 to node 9.

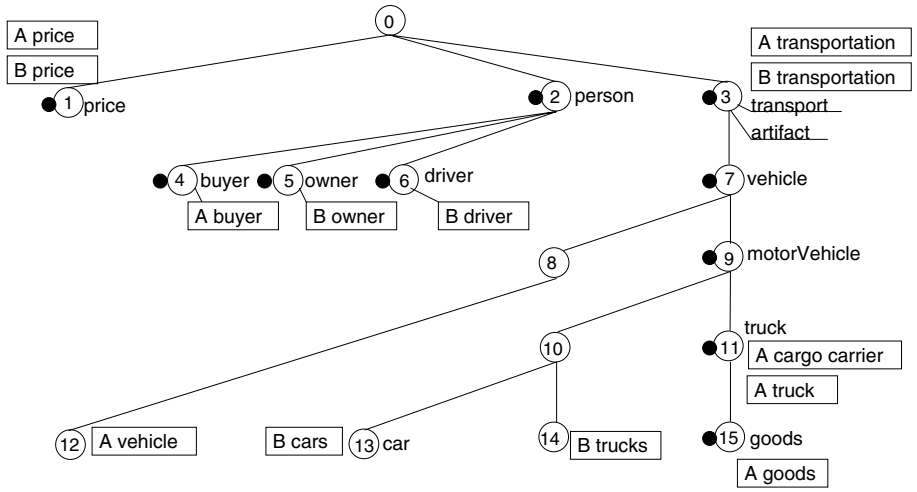


Fig. 7. Spanning tree for the lattice of the example, determined using the similarity measure.

The hyperbolic tree corresponding to the spanning tree in Fig. 7 is shown in Fig. 8. The original lattice nodes correspond to the inner nodes, every leaf node represents an object, and the nodes indicated with arrows[9] are the broken links. The broken links serve as an indication to the reader that there was another alternative way down the lattice but that was broken during the construction of the spanning tree. The broken link 10 coming out of node 8, for example, is indicating that there is a link between 8 and 10, but the node 10 and its successors were drawn elsewhere. Actually, it is linked to node `motorVehicle`. Every node has an associated number, but to enable the user to easily identify the meet-irreducible elements, we put the names of the associated attributes instead of the number in these nodes. Hence, the label `motorVehicle` is displayed instead of the label "node 9".

The small window on the upper left corner is used for searching in the lattice. If we type "price", it will mark all nodes containing words, either on its title or on the associated data structure, beginning with the string "price". This can also be used to find a node label. For example, if we had found the node 10 and we would like to find the actual node 10, we would have just to type "10" in the small window and follow the paths leading to node 10. One of them would be the actual node 10.

To solve the problem related to meets suppressed by the construction of the spanning tree, the visualization tool was also designed to process AND queries. With this feature, we are able to search for nodes having (`truck` $\wedge$ `goods`). Nevertheless, this is not enough to identify the meets correctly, because the

[9] In the software tool, they are displayed in a different color

word truck, for example, may be used either as part of an object name or as an attribute.

To eliminate this problem, every meet-irreducible element is associated with a unique identifier, like `#pc` for price and `#ve` for `vehicle`. Moreover, when constructing the hyperbolic tree, we attach the intent of every node to the node itself. For example, if a node has `price` and `vehicle` it will have `#pc` and `#ve` as value in its data structure. As a consequence, if we type `#pc` in the upper left corner of Fig. 8 all the nodes that have `price` as intent will be highlighted with a dot and the path up to those nodes will be drawn (dashed line shown in Fig. 8). From the path marked in Fig. 8, we can see that there are three objects that have `#pc` as attribute, namely `B price`, `A price` and `A vehicle`.

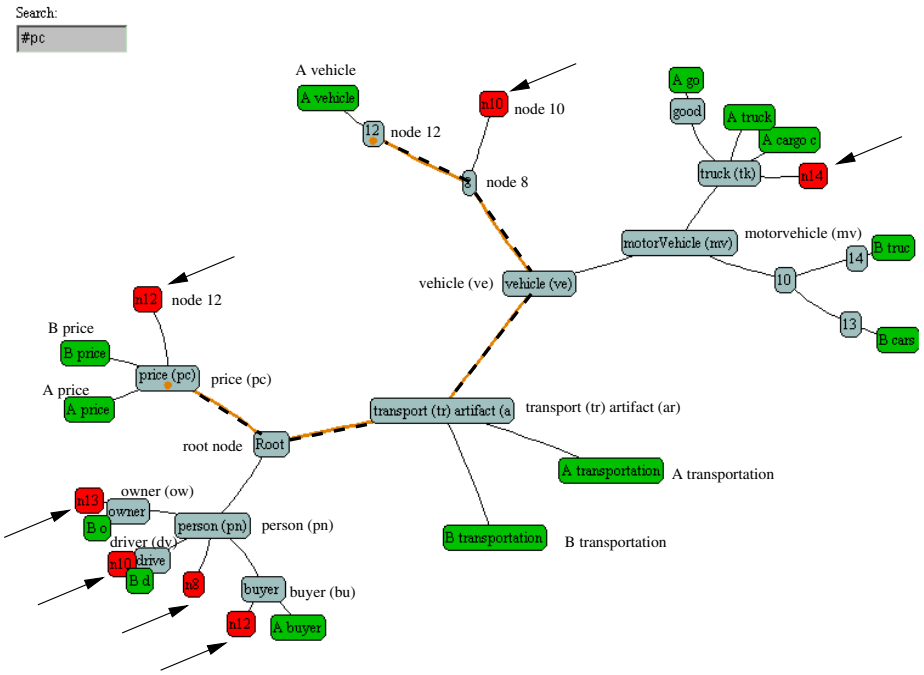


Fig. 8. Hyperbolic tree for the lattice of the example.

Since the search routine is able to process AND sentences, we can, for example, search for all nodes that satisfy (`vehicle` ∧ `price`). This is an interesting meet that was separated by the spanning tree construction. When we type "`#ve` & `#pc`", we can find that the only node satisfying this condition is node 12 and the associated object `A vehicle`. Figure 9 shows the result for this case. It is represented by the dashed line from the root node up only to node 12, which is the only one satisfying this conjunction.

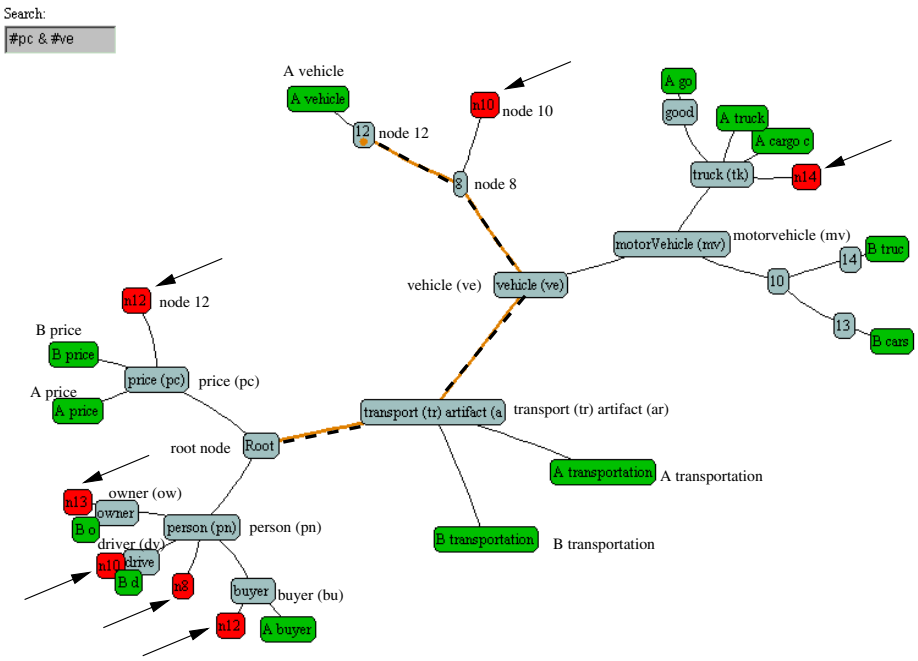


Fig. 9. Hyperbolic tree highlighting the nodes satisfying the condition: "`#ve` & `#pc`".

The heuristics for determining the spanning tree can be summarized as follows:

1. Determine the similarity measure between each pair of nodes in the Hasse Diagram;
2. Starting from the bottom of the Hasse Diagram, which we call N-Level, determine which parent each node from that level should be linked to;
3. If two parents have the same similarity value, select the one with the largest number of meet-irreducible elements, counting from that node up to the top node;
4. Repeat for the levels $(N-1), (N-2), \ldots, 1$
5. Determine a unique identifier for each attribute that corresponds to a meet-irreducible element;
6. Add to data structure associated to each node the identifiers of all attributes which are part of the intent for this node;
7. Add to each node, all the objects directly linked to it.

One point that is worth mentioning is that we have used the hyperbolic tree visualization for the analysis that we have performed in the concept lattice. However, we did not conduct any user tests to evaluate proposed solution. As pointed

out by Herman et al. [30], there is paucity of research addressing the problem of user interface evaluation in this area. This is probably due to the need for the application of both cognitive science and human factors in such evaluations. These areas, however, have few findings that can be applied in practice.

We adopted a different approach: given that cluttering, edge crossing and discernability are important usability factors for information visualization, we have proposed a solution that minimized those problems. Nevertheless, our heuristics represent an application of Cognitive Science, in that it uses a similarity measure developed from psychological studies.

9 Conclusion

This article presented a complete solution for the alignment of ontologies, composed of structural alignment, evaluation of similarities between concepts and visualization of the result.

The alignment method is based on Formal Concept Analysis, or Galois Lattices, a data analysis technique grounded in Lattice Theory and Propositional Calculus. Two alternatives were considered: (a) transform each ontology in a concept lattice and merge them, and (b) align the ontologies with an upper-ontology constructed using a thesaurus, namely Agrovoc. The latter option was adopted.

The results showed that anchoring two ontologies in a common partial ordering provided by a lattice of terms (thesaurus) is an excellent analysis tool for the structure of these ontologies. Actually, a complete logical system can be constructed using lattice terms as first order predicates [12]. As a result, a complex logical system can be built over the lattice, enabling the system to process elaborate queries involving logical operators.

We also analyzed the FCA's lattice structure and proposed a measure of similarity based on Tversky's model, which allowed us to identify closely related concepts in different source ontologies. The similarity measure developed considered a special set of elements in the lattice called meet-irreducible. These elements are similar to the concept of a basis in Linear Algebra, in the sense that the entire lattice can be reconstructed from them.

To address one of the main drawbacks of the application of FCA, viz. the cluttering in visualization when the lattice grows, we proposed a new approach. This approach consists of a special spanning tree representation of the lattice, constructed considering our structural similarity measure. By doing so, we proposed that the nodes structurally closer to each other stayed. The result was then visualized through a hyperbolic tree tool, for which we implemented some special features. The tool's features were developed taking into account the navigation needs in concept lattices.

References

[1] Berners-Lee, T.: Semantic web road map. Internal note, World Wide Web Consortium (1998) URL http://www.w3.org/DesignIssues/Semantic.html.
[2] Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (2001)
[3] Stumme, G., Maedche, A.: Fca-merge: Bottom-up merging of ontologies. In: Proc. 17th Intl. Conf. on Artificial Intelligence (IJCAI '01), Seattle, WA, USA. (2001) 225–230
[4] Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Ontology matching: A machine learning approach. In Staab, S., Studer, R., eds.: Handbook on Ontologies. International Handbooks on Information Systems, Springer (2004) 385–404
[5] Rodríguez, M.A., Egenhofer, M.J.: Determining semantic similarity among entity classes from different ontologies. IEEE Transactions on Knowledge and Data Engineering **15** (2003) 442–456
[6] Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Aquisition **5** (1993) 199–220
[7] de Souza, K.X.S., Davis, J.: Aligning ontologies through formal concept analysis. In: Proceedings of The Sixth International Conference on Information Integration and Web Based Applications & Services (iiWAS2004), Jakarta, Indonesia, Austrian Computer Society (2004)
[8] Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In Rival, I., ed.: Ordered Sets. Volume 83 of NATO Advanced Study Institute Series C. Reidel, Dordrecht (1982) 445–470
[9] Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin - Heidelberg - New York (1999)
[10] Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review **18** (2003) 1–31
[11] de Souza, K.X.S., Davis, J.: Using an aligned ontology to process user queries. In: Artificial Intelligence: Methodology, Systems, and Applications: 11th International Conference, AIMSA 2004, Varna, Bulgaria, September 2-4, 2004. Proceedings. Number 3192 in Lecture Notes in Computer Science, Springer-Verlag Heidelberg (2004) 44–53
[12] Chaudron, L., Maille, N., Boyer, M.: The cube lattice model and its applications. Applied Artificial Intelligence **17** (2003) 207–242
[13] de Souza, K.X.S., Davis, J.: Aligning ontologies and evaluating concept similarities. In: On The Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Lanarca, Cyprus. Proceedings. Number 3291 in Lecture Notes in Computer Science, Springer-Verlag Heidelberg (2004) 1012–1029
[14] Tversky, A.: Features of Similarity. Psychological Review **84** (1977) 327–352
[15] McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: An environment for merging and testing large ontologies. In Cohn, A.G., Giunchiglia, F., Selman, B., eds.: KR2000: Principles of Knowledge Representation and Reasoning, San Francisco, Morgan Kaufmann (2000) 483–493
[16] Noy, N.F., Musen, M.: PROMPT: Algorithm and tool for automated ontology merging and alignment. In: Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), Austin, Texas, AAAI Press (2000) 450–455
[17] Chalupsky, H.: Ontomorph: A translation system for symbolic knowledge. In: Principles of Knowledge Representation and Reasoning. (2000) 471–482

[18] Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10** (2001) 334–350
[19] Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems **6** (2004) 428–440
[20] Compatangelo, E., Meisel, H.: Intelligent support to knowledge sharing through the articulation of class schemas. In: Proc. of the 6th Intl. Conf. on Knowledge-Based Intelligent Information & Engineering Systems(KES'2002), IOS Press (2002) 306–310
[21] Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hubner, S.: Ontology-based integration of information - a survey of existing approaches. In Stuckenschmidt, H., ed.: IJCAI-01 Workshop: Ontologies and Information Sharing. (2001) 108–117
[22] Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.: Domain specific ontologies for semantic information brokering on the global information infrastructure. In: Proceedings of the 1st International Conference on Formal Ontology in Information Systems(FOIS98). (1998) 269–283
[23] de Souza, K.X.S., Davis, J., Souza, M.I.F.: Organizing information for the agribusiness sector: Embrapa's Information Agency. In: Proceedings of 2004 International Conference on Digital Archive Technologies, Taipei, Taiwan, Institute of Information Science - Academia Sinica (2004) 159–169
[24] Priss, U.: Formalizing botanical taxonomies. In: Conceptual Structures for Knowledge Creation and Communication. Proceedings of the 11th International Conference on Conceptual Structures. Number 2746 in LNAI, Springer Verlag (2003) 309–322
[25] Cole, R., Eklund, P.: Application of formal concept analysis to information retrieval using a hierarchically structured thesauris. In: Supplementary Proceedings of International Conference on Conceptual Structures, ICCS '96, University of New South Wales (1996) 1–12
[26] Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map between ontologies on the semantic web. In: The Eleventh International WWW Conference, Hawaii, USA (2002)
[27] Groh, B., Strahinger, S., Wille, R.: Toscana-systems based on thesauri. In: Proceedings 6th International Conference on Conceptual Structures. Number 1453 in LNAI, Springer Verlag, Berlin (1998) 127–138
[28] Becker, P., Hereth, J., Stumme, G.: ToscanaJ: An open source tool for qualitative data analysis. In Duquenne, V., Ganter, B., Liquiere, M., Nguifo, E.M., Stumme, G., eds.: Advances in Formal Concept Analysis for Knowledge Discovery in Databases. Proc. Workshop FCAKDD of the 15th European Conference on Artificial Intelligence (ECAI 2002). Lyon, France, July 23, 2002. (2002)
[29] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with titanic. Journal on Knowledge and Data Engineering (KDE) **42** (2002) 189–222
[30] Herman, I., Melançon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: A survey. IEEE Transactions on Visualization and Computer Graphics **6** (2000) 24–43
[31] Sarkar, M., Brown, M.H.: Graphical fisheye views of graphs. In Bauersfeld, P., Bennett, J., Lynch, G., eds.: Proceedings of the Conference on Human Factors in Computing Systems, ACM Press (1992) 83–92

[32] Lamping, J., Rao, R., Pirolli, P.: A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In Katz, I.R., Mack, R., Marks, L., Rosson, M.B., Nielsen, J., eds.: Proceedings of the Conference on Human Factors in Computing Systems (CHI'95), Denver, CO, USA, ACM Press (1995) 401–408
[33] Munzner, T.: H3: laying out large directed graphs in 3d hyperbolic space. In: IEEE Symposium on Information Visualization (InfoVis '97), IEEE (1997) 2–10
[34] FAO (Food and Agriculture Organization of the United Nations): FAO (Food and Agriculture Organization of the United Nations). AGROVOC: Multilingual Agricultural Thesaurus (1995) FAO. Rome.
[35] Mitra, P., Wiederhold, G., Kersten, M.L.: A Graph-Oriented Model for Articulation of Ontology Interdependencies. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Volume 1777 of Lecture Notes in Computer Science (LNCS), Springer-Verlag., Konstanz, Germany (2000) 86–100
[36] Heit, E.: Features of similarity and category-based induction. In: Proceedings of the Interdisciplinary Workshop on Categorization and Similarity, University of Edinburgh (1997) 115–121
[37] Goldstone, R.L., Kersten, A.: Concepts and caterogization. In Healy, A., Proctor, R., eds.: Comprehensive Handbook of Psychology. Wiley, New Jersey (2003) 599–621
[38] Sloutsky, V.M.: The role of similarity in the development of categorization. TRENDS in Cognitive Sciences **7** (2003) 246–251
[39] Tenenbaum, J.B., Griffiths, T.L.: Generalization, similarity, and bayesian inference. Behavioral and Brain Sciences **24** (2001) 629–640
[40] Gentner, D., Markman, A.B.: Structure mapping in analogy and similarity. American Psychologist **52** (1997) 45–56
[41] Lin, D.: An information-theoretic definition of similarity. In: Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc (1998) 296–304
[42] Godin, R., Mili, H.: Building and maintaining analysis-level class hierarchies using galois lattices. In Paepcke, A., ed.: Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, Washington, DC, USA, ACM Press (1993) 394–410

Author Index