Lecture Notes in Computer Science 2593

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Akmal B. Chaudhri Mario Jeckle
Erhard Rahm Rainer Unland (Eds.)

Web, Web-Services, and Database Systems

NODe 2002 Web- and Database-Related Workshops
Erfurt, Germany, October 7-10, 2002
Revised Papers

Springer

Volume Editors

Akmal B. Chaudhri
Wimbledon, London SW19 3DF, UK
E-mail: akmal@soi.city.ac.uk

Mario Jeckle
DaimlerChrysler Corporate Research Center Ulm
Wilhelm Runge Straße 11, 89073 Ulm, Germany
E-mail: mario@jeckle.de

Erhard Rahm
Universität Leipzig, Institut für Informatik
Augustusplatz 10-11, 04109 Leipzig, Germany
E-mail: rahm@informatik.uni-leipzig.de

Rainer Unland
University of Essen, Institute for Computer Science
Schützenbahn 70, 45117 Essen, Germany
E-mail: UnlandR@informatik.uni-essen.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek.
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

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

ISSN 0302-9743
ISBN 3-540-00745-8 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e.K.
Printed on acid-free paper SPIN: 10872506 06/3142 5 4 3 2 1 0

Preface

Net.ObjectDays (NODe) has established itself as one of the most significant events on Objects, Components, Architectures, Services and Applications for a Networked World in Europe and in the world. As in previous years, it took place in the Messekongresszentrum (Fair and Convention Center) in Erfurt, Thuringia, Germany, this time during 7–10 October 2002. Founded only three years ago as the official successor conference to JavaDays, STJA (Smalltalk and Java in Industry and Education) and JIT (Java Information Days), NODe has grown into a major international conference that attracts participants from industry, research and users in equal measure since it puts strong emphasis on the active exchange of concepts and technologies between these three communities.

Over the past few years, the NODe conference has developed a remarkable track record: a new paradigm (*Generative Programming*) was born at NODe (citation James Coplien), nearly all of the most prominent researchers and contributors in the object-oriented field (and beyond) have given keynotes at NODe, new topics have been integrated (like Agent Technology and Web-Services) and, now, for the first time, postconference proceedings are being published by Springer-Verlag. Altogether three volumes will be available. This volume is compiled from the best papers of the *Web Databases* and the *Web-Services* workshops. Two additional volumes will be published, one containing the best contributions of the main conference and another one with the best contributions to the agent-related workshops and the *3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB 2002)* that were cohosted with NODe 2002: M. Aksit, M. Mezini, R. Unland (editors) *Objects, Components, Architectures, Services, and Applications for a Networked World* (*LNCS 2591*) and R. Kowalczyk, J. Müller, H. Tianfield, R. Unland (editors) *Agent Technologies, Infrastructures, Tools, and Applications for e-Services* (*LNAI 2592*).

This volume contains the keynote speeches as well as 19 peer-reviewed, original papers that were chosen from the papers accepted for the workshops. This means that the papers in this volume are a subset of the papers presented at the conference, which in turn were selected by the programme committees out of the submitted papers based on their scientific quality, the novelty of the ideas, the quality of the writing, and the practical relevance. This double selection process not only guaranteed high-quality papers but also allowed the authors to consider comments and suggestions they had received during the conference and to integrate them into their final version. Furthermore, authors were allowed to extend their papers to fully fledged versions. We hope that the results will convince you as much as they did us and that these proceedings give you many new inspirations and insights.

The contents of this volume can best be described by an excerpt from the original Call for Papers:

Web Databases
The workshop centers on database technology and the Web. Flexibility, scalability, heterogeneity and distribution are typical characteristics of the Web infrastructure. Whereas these characteristics offer rich potential for today's and future application domains, they put high demands on database systems and especially their architecture, testing interoperability, access to and maintenance of structured and semistructured (heterogeneous) data sources, and the integration of applications. Of particular importance and interest to the workshop is database support for XML (Extensible Markup Language) and Web-services.

WS-RSD 2002
Web-services promise to ease several current infrastructure challenges. Especially they are expected to herald a new era of integration: integration of data, processes, and also complete enterprises on the basis of a set of loosely related lightweight approaches that hide all technical implementation details except the Web, which serves as the basic transport and deployment mechanism. Interoperability will surely prove itself as the critical success factor of the Web service proliferation promised by some analysts. In order to accomplish this interoperability, various standardization bodies such as the W3C, UN and OASIS have initiated activities working on specifications of building blocks of a Web-service architecture. Nevertheless, a significant number of vendors now offer the first commercial solutions, which have been implemented to create some real-world services.

As editors of this volume, we would like to thank once again all programme committee members, as well as all external referees for their excellent work in evaluating the submitted papers. Moreover, we would like to thank Mr. Hofmann from Springer-Verlag for his cooperation and help in putting this volume together.

December 2002

Akmal B. Chaudhri,
Mario Jeckle,
Erhard Rahm,
Rainer Unland

2nd Annual International Workshop "*Web Databases*" of the Working Group "Web and Databases" of the German Informatics Society (GI)

Organizers

Akmal B. Chaudhri, IBM developerWorks, USA
Erhard Rahm, University of Leipzig, Germany
Rainer Unland, University of Essen, Germany

Members of the International Programme Committee

Wolfgang Benn, TU Chemnitz
Ron Bourret, Independent Consultant
Hosagrahar V. Jagadish, University of Michigan
Udo Kelter, University of Siegen
Alfons Kemper, University of Passau
Thomas Kudraß, HTWK Leipzig
Meike Klettke, University of Rostock
Marco Mesiti, University of Genoa
Ingo Macherius, Infonyte GmbH
Kjetil Nørvåg, Norwegian University of Science and Technology
Jaroslav Pokorný, Charles University
Awais Rashid, University of Lancaster
Werner Retschitzegger, University of Linz
Harald Schöning, Software AG
Gottfried Vossen, University of Münster
Gerhard Weikum, University of Saarbrücken

Workshop on "*Web-Services – Research, Standardization, and Deployment*" (WS-RSD 2002)

Organizer

Mario Jeckle
DaimlerChrysler Corporate Research Center Ulm
Wilhelm Runge Str. 11
89073 Ulm, Germany
E-mail: mario@jeckle.de

Members of the International Programme Committee

Dieter Fensel, Vrije Universiteit Amsterdam
Christopher Ferris, Sun Microsystems
Bogdan Franczyk, University of Leipzig
Frank Leymann, IBM
Jean-Philippe Martin-Flatin, CERN
Erich Ortner, Technical University of Darmstadt
Günther Specht, University of Ulm
Michael Stal, Siemens
Matthias Weske, Hasso-Plattner Institute Potsdam
Liang-Jie Zhang, IBM T.J. Watson Research Center

Table of Contents

Applications Based on Web-Services

Web and XML Databases

Indexing and Accessing

Mobile Devices and the Internet

XML Query Languages

GRIDs and Ambient Computing

Keith G. Jeffery

Director, IT; CLRC Rutherford Appleton Laboratory;
Chilton; Didcot; Oxfordshire OX11 0QX UK
k.g.Jeffery@rl.ac.uk

Abstract. GRIDs are both a new and an old concept. Many of the components have been the subject of R&D and some exist as commercial products. The GRIDs concept represents many different things to different people: metacomputing, distributed computing, advanced networking, distributed database, information retrieval, digital libraries, hypermedia, cooperative working, knowledge management, advanced user interfaces, mobile and pervasive computing and many others. More importantly, end-users see the GRIDs technology as a means to an end - to improve quality, speed of working and cooperation in their field. GRIDs will deliver the required information in an appropriate form to the right place in a timely fashion. The novelty of GRIDs lies in the information systems engineering required in generating missing components and putting the components together. Ambient computing provides new possibilities in connectivity of a person (with or without sensors or data detectors) to a GRIDs environment allowing previously unimaginable possibilities in information delivery, data collection, command and control, cooperative working, communications, learning and entertainment.

1 Introduction

We live in a complex and fast-changing world. Modern sensors and detectors produce vast quantities of raw or calibrated data. Data acquisition systems produce large quantities of reduced and structured data. Modelling and simulation systems produce very large amounts of generated, processed data. The requirements of business, retail sales, production, travel, tourism, agriculture, scientific R&D and other activities all require information and knowledge to be extracted from these vast volumes of data in such a way as to assist human decision-making. Furthermore, the end-users require its presentation to be correct and in the appropriate form, at the right place in a timely fashion.

The challenge is to collect the data in such a way as to represent as accurately as possible the real world, and then to process it to information (here defined as structured data in context) from which can be extracted knowledge (here defined as justified commonly held belief).

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 1–11, 2003.

2 GRIDS and Ambient Computing

Present-day systems are heterogeneous and poorly interlinked. Humans have to search for information from many sources and find it stored in different character sets, languages, data formats. It may be stored in different measurement units, at different precision, with different accuracies more-or-less supported by calibration data. Worse, having found appropriate sources they have to find sufficient computing power to process the data, to integrate it together to a useful form for the problem at hand and suitable visualisation capabilities to provide a human-understandable view of the information. Suitable expert decision support systems and data mining tools for the creation of knowledge from information, and communications environments for group decision-making, are also hard to find and use. The new paradigms of GRIDs and Ambient Computing are an attempt to overcome these problems.

The paper is organised as follows: the GRIDs concept, with its components, is described. Ambient Computing is similarly described. Then the use of the technology for environmental applications is explored.

3 GRIDs

3.1 The Idea

In 1998-1999 the UK Research Council community was proposing future programmes for R&D. The author was asked to propose an integrating IT architecture. The proposal was based on concepts including distributed computing, metacomputing, metadata, middleware, client-server and knowledge-based assists. The novelty lay in the integration of various techniques into one architectural framework.

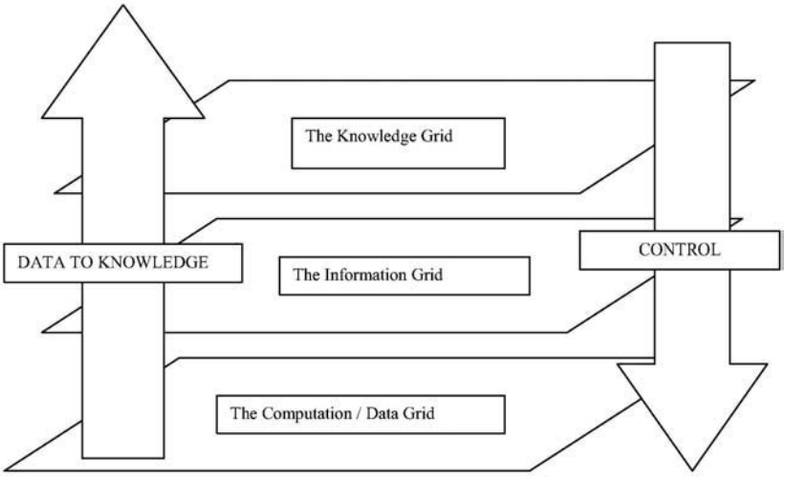


Fig. 1. The 3-Layer GRIDs Architecture

3.2 The Requirement

The UK Research Council community of researchers was facing several IT-based problems. Their ambitions for scientific discovery included post-genomic discoveries, climate change understanding, oceanographic studies, environmental pollution monitoring and modelling, precise materials science, studies of combustion processes, advanced engineering, pharmaceutical design, and particle physics data handling and simulation. They needed more processor power, more data storage capacity, better analysis and visualisation – all supported

3.3 Architecture Overview

The architecture proposed consists of three layers (Fig. 1). The computation / data grid has supercomputers, large servers, massive data storage facilities and specialised devices and facilities (e.g. for VR (Virtual Reality)) all linked by high-speed networking and forms the lowest layer. The main functions include compute load sharing / algorithm partitioning, resolution of data source addresses, security, replication and message rerouting. The information grid is superimposed on the computation / data grid and resolves homogeneous access to heterogeneous information sources mainly through the use of metadata and middleware. Finally, the uppermost layer is the knowledge grid which utilises knowledge discovery in database technology to generate knowledge and also allows for representation of knowledge through scholarly works, peer-reviewed (publications) and grey literature, the latter especially hyperlinked to information and data to sustain the assertions in the knowledge.

The concept is based on the idea of a uniform landscape within the GRIDs domain, the complexity of which is masked by easy-to-use interfaces. To this facility are connected external appliances - ranging from supercomputers, storage access networks, data storage robots, specialised visualisation and VR systems, data sensors and detectors (e.g. on satellites) to user client devices such as workstations, notebook computers, PDAs (Personal Digital Assistants – 'palmtops') and enabled Mobile phones. The connection between the external appliances and the GRIDs domain is through agents, supported by metadata, representing the appliance (and thus continuously available to the GRIDs systems). These representative agents handle credentials of the end-user in their current role, appliance characteristics and interaction preferences (for both user client appliances and service appliances), preference profiles and associated organisational information. These agents interact with other agents in the usual way via brokers to locate services and negotiate use. The key aspect is that all the agent interaction is based upon available metadata.

3.4 The GRID

In 1998 – in parallel with the initial UK thinking on GRIDs – Ian Foster and Carl Kesselman published a collection of papers in a book generally known as 'The GRID Bible' [FoKe98]. The essential idea is to connect together supercomputers to provide more power – the metacomputing technique. However, the major contribution lies in

the systems and protocols for compute resource scheduling. Additionally, the designers of the GRID realised that these linked supercomputers would need fast data feeds so developed GRIDFTP. Finally, basic systems for authentication and authorisation are described. The GRID has encompassed the use of SRB (Storage Request Broker) from SDSC (San Diego Supercomputer Centre) for massive data handling. SRB has its proprietary metadata system to assist in locating relevant data resources.

The GRID corresponds to the lowest grid layer (computation / data layer) of the GRIDs architecture.

4 The GRIDs Architecture

4.1 Introduction

The idea behind GRIDs is to provide an IT environment that interacts with the user to determine the user requirement for service and then satisfies that requirement across a heterogeneous environment of data stores, processing power, special facilities for display and data collection systems thus making the IT environment appear homogeneous to the end-user.

As an example an end-user might require to make decisions about settling of a polluting industrial plant in a location. The user would require access to healthcare data and social data (census statistics) for an area to determine the population characteristics and potential health risk. Access would also be required to environmental data such as historical records of wind directions and speed (to determine likely tracks for the pollution) and rainfall (to 'scrub' the pollution from the air). Similarly botanical records would be needed to determine the effect of pollution on plant-life. Soil data would also be required for similar reasons. The econometric data or the areas would also be needed in order to assess the positive economic effects of new jobs. Data on transport routes and capacity, water supply, electricity supply would so be required to assure (or estimate the cost of providing) suitable infrastructure.

Clearly, such a user request will require access to data and information sources, resources to integrate the heterogeneous information both spatially and temporally, computation resources for the modelling and visualisation facilities to display the correlated results. The idea of the GRIDs IT environment is to make all that transparent to the end-user.

A huge range of end-user requirements, from arranging a holiday to buying a fitted kitchen, from operating on the stock exchange to operating in a hospital, all can be satisfied by the GRIDs architecture. In the R&D domain, problems in all scientific areas can be tackled and advanced towards solution using GRIDs.

4.2 The Architecture Components

The major components external to the GRIDs environment are (Fig. 2):

- users: each being a human or another system;
- sources: data, information or software
- resources: such as computers, sensors, detectors, visualisation or VR (virtual reality) facilities

Each of these three major components is represented continuously and actively within the GRIDs environment by:

- metadata: which describes the external component and which is changed with changes in circumstances through events
- an agent: which acts on behalf of the external resource representing it within the GRIDs environment.

As a simple example, the agent could be regarded as the answering service of a person's mobile phone and the metadata as the instructions given to the service such as 'divert to service when busy' and / or 'divert to service if unanswered'.

Finally there is a component which acts as a 'go between' between the agents. These are brokers which, as software components, act much in the same way as human brokers by arranging agreements and deals between agents, by acting themselves (or using other agents) to locate sources and resources, to manage data integration, to ensure authentication of external components and authorisation of rights to use by an authenticated component and to monitor the overall system.

From this it is clear that they key components are the metadata, the agents and the brokers.

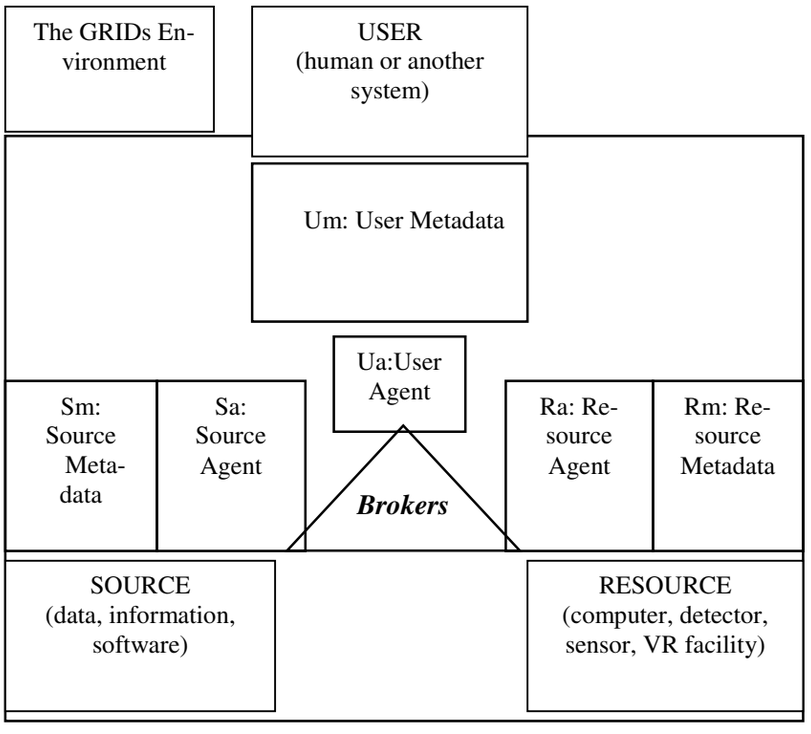


Fig. 2. GRIDs Architecture

4.3 Metadata

Metadata is data about data [Je00]. An example might be a product tag attached to a product (e.g. a tag attached to a piece of clothing) that is available for sale. The metadata on the product tag tells the end-user (human considering purchasing the article of clothing) data about the article itself – such as the fibres from which it is made, the way it should be cleaned, its size (possibly in different classification schemes such as European, British, American) and maybe style, designer and other useful data. The metadata tag may be attached directly to the garment, or it may appear in a catalogue of clothing articles offered for sale (or, most likely, both). The metadata may be used to make a selection of potentially interesting articles of clothing before the actual articles are inspected, thus improving convenience. Today this concept is widely-used. Much e-commerce is based on B2C (Business to Customer) transactions based on an online catalogue (metadata) of goods offered. One well-known example is amazon.com for books and now a wide range of associated products.

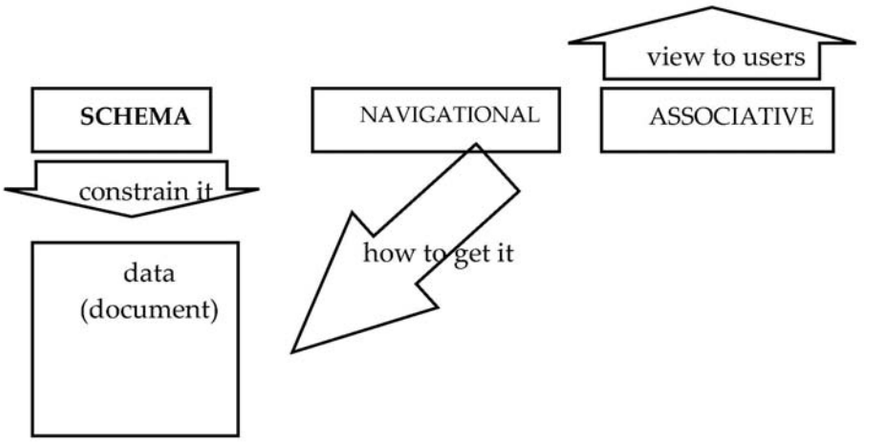


Fig. 3. Metadata Classification

What is metadata to one application may be data to another. For example, an electronic library catalogue card is metadata to a person searching for a book on a particular topic, but data to the catalogue system of the library which will be grouping books in various ways: by author, classification code, shelf position, title – depending on the purpose required.

It is increasingly accepted that there are several kinds of metadata. The classification proposed (Fig. 3.) is gaining wide acceptance and is detailed below.

4.3.1 Schema Metadata

Schema metadata constrains the associated data. It defines the intension whereas instances of data are the extension. From the intension a theoretical universal extension can be created, constrained only by the intension. Conversely, any observed instance

should be a subset of the theoretical extension and should obey the constraints defined in the intension (schema). One problem with existing schema metadata (e.g. schemas for relational DBMS) is that they lack certain intensional information that is required [JeHuKaWiBeMa94]. Systems for information retrieval based on, e.g. the SGML (Standard Generalised Markup Language) DTD (Document Type Definition) experience similar problems.

It is noticeable that many ad hoc systems for data exchange between systems send with the data instances a schema that is richer than that in conventional DBMS – to assist the software (and people) handling the exchange to utilise the exchanged data to best advantage.

4.3.2 Navigational Metadata

Navigational metadata provides the pathway or routing to the data described by the schema metadata or associative metadata. In the RDF model it is a URL (universal resource locator), or more accurately, a URI (Universal Resource Identifier). With increasing use of databases to store resources, the most common navigational metadata now is a URL with associated query parameters embedded in the string to be used by CGI (Common Gateway Interface) software or proprietary software for a particular DBMS product or DBMS-Webserver software pairing.

The navigational metadata describes only the physical access path. Naturally, associated with a particular URI are other properties such as:

1. security and privacy (e.g. a password required to access the target of the URI);
2. access rights and charges (e.g. does one have to pay to access the resource at the URI target);
3. constraints over traversing the hyperlink mapped by the URI (e.g. the target of the URI is only available if previously a field on a form has been input with a value between 10 and 20). Another example would be the hypermedia equivalent of referential integrity in a relational database;
4. semantics describing the hyperlink such as ‘the target resource describes the son of the person described in the origin resource’

However, these properties are best described by associative metadata which then allows more convenient co-processing in context of metadata describing both resources and hyperlinks between them and – if appropriate – events.

4.3.3 Associative Metadata

In the data and information domain associative metadata can describe:

1. a set of data (e.g. a database, a relation (table) or a collection of documents or a retrieved subset). An example would be a description of a dataset collected as part of a scientific mission;
2. an individual instance (record, tuple, document). An example would be a library catalogue record describing a book ;

3. an attribute (column in a table, field in a set of records, named element in a set of documents). An example would be the accuracy / precision of instances of the attribute in a particular scientific experiment ;
4. domain information (e.g. value range) of an attribute. An example would be the range of acceptable values in a numeric field such as the capacity of a car engine or the list of valid values in an enumerated list such as the list of names of car manufacturers;
5. a record / field intersection unique value (i.e. value of one attribute in one instance) This would be used to explain an apparently anomalous value.

In the relationship domain, associative metadata can describe relationships between sets of data e.g. hyperlinks. Associative metadata can – with more flexibility and expressivity than available in e.g. relational database technology or hypermedia document system technology – describe the semantics of a relationship, the constraints, the roles of the entities (objects) involved and additional constraints.

In the process domain, associative metadata can describe (among other things) the functionality of the process, its external interface characteristics, restrictions on utilisation of the process and its performance requirements / characteristics.

In the event domain, associative metadata can describe the event, the temporal constraints associated with it, the other constraints associated with it and actions arising from the event occurring.

Associative metadata can also be personalised: given clear relationships between them that can be resolved automatically and unambiguously, different metadata describing the same base data may be used by different users.

Taking an orthogonal view over these different kinds of information system objects to be described, associative metadata may be classified as follows:

1. descriptive: provides additional information about the object to assist in understanding and using it;
2. restrictive: provides additional information about the object to restrict access to authorised users and is related to security, privacy, access rights, copyright and IPR (Intellectual Property Rights);
3. supportive: a separate and general information resource that can be cross-linked to an individual object to provide additional information e.g. translation to a different language, super- or sub-terms to improve a query – the kind of support provided by a thesaurus or domain ontology;

Most examples of metadata in use today include some components of most of these kinds but neither structured formally nor specified formally so that the metadata tends to be of limited use for automated operations – particularly interoperation – thus requiring additional human interpretation.

4.4 Agents

Agents operate continuously and autonomously and act on behalf of the external component they represent. They interact with other agents via brokers, whose task it is to locate suitable agents for the requested purpose. An agent's actions are con-

trolled to a large extent by the associated metadata which should include either instructions, or constraints, such that the agent can act directly or deduce what action is to be taken. Each agent is waiting to be 'woken up' by some kind of event; on receipt of a message the agent interprets the message and – using the metadata as parametric control – executes the appropriate action, either communicating with the external component (user, source or resource) or with brokers as a conduit to other agents representing other external components.

An agent representing an end-user accepts a request from the end-user and interacts with the end-user to refine the request (clarification and precision), first based on the user metadata and then based on the results of a first attempt to locate (via brokers and other agents) appropriate sources and resources to satisfy the request. The proposed activity within GRIDs for that request is presented to the end-user as a 'deal' with any costs, restrictions on rights of use etc. Assuming the user accepts the offered deal, the GRIDs environment then satisfies it using appropriate resources and sources and finally sends the result back to the user agent where – again using metadata – end-user presentation is determined and executed.

An agent representing a source will – with the associated metadata – respond to requests (via brokers) from other agents concerning the data or information stored, or the properties of the software stored. Assuming the deal with the end-user is accepted, the agent performs the retrieval of data requested, or supply of software requested.

An agent representing a resource – with the associated metadata – responds to requests for utilisation of the resource with details of any costs, restrictions and relevant capabilities. Assuming the deal with the end-user is accepted the resource agent then schedules its contribution to providing the result to the end-user.

4.5 Brokers

Brokers act as 'go betweens' between agents. Their task is to accept messages from an agent which request some external component (source, resource or user), identify an external component that can satisfy the request by its agent working with its associated metadata and either put the two agents in direct contact or continue to act as an intermediary, possibly invoking other brokers (and possibly agents) to handle, for example, measurement unit conversion or textual word translation.

Other brokers perform system monitoring functions including overseeing performance (and if necessary requesting more resources to contribute to the overall system e.g. more networking bandwidth or more compute power). They may also monitor usage of external components both for statistical purposes and possibly for any charging scheme.

4.6 The Components Working Together

Now let us consider how the components interact. An agent representing a user may request a broker to find an agent representing another external component such as a source or a resource. The broker will usually consult a directory service (itself controlled by an agent) to locate potential agents representing suitable sources or resources. The information will be returned to the requesting (user) agent, probably with recommendations as to order of preference based on criteria concerning the offered services. The user agent matches these against preferences expressed in the

metadata associated with the user and makes a choice. The user agent then makes the appropriate recommendation to the end-user who in turn decides to 'accept the deal' or not.

5 Ambient Computing

The concept of ambient computing implies that the computing environment is always present and available in an even manner. The concept of pervasive computing implies that the computing environment is available everywhere and is 'into everything'. The concept of mobile computing implies that the end-user device may be connected even when on the move. In general usage of the term, ambient computing implies both pervasive and mobile computing.

The idea, then, is that an end-user may find herself connected to the computing environment all the time. The computing environment may involve information provision (access to database and web facilities), office functions (calendar, email, directory), desktop functions (word processing, spreadsheet, presentation editor), perhaps project management software and systems specialised for her application needs – accessed from her end-user device connected back to 'home base' so that her view of the world is as if at her desk. In addition entertainment subsystems (video, audio, games) should be available.
A typical configuration might comprise:

1. a headset with earphones and microphone for audio communication, connected by Bluetooth wireless local connection to;
2. a PDA (personal digital assistant) with small screen, numeric/text keyboard (like a telephone), GSM/GPRS (mobile phone) connections for voice and data, wireless LAN connectivity and ports for connecting sensor devices (to measure anything close to the end-user) in turn connected by Bluetooth to;
3. an optional notebook computer carried in a backpack (but taken out for use in a suitable environment) with conventional screen, keyboard, large hard disk and connectivity through GSM/GPRS, wireless LAN, cable LAN and dial-up telephone

The end-user would perhaps use only (a) and (b) (or maybe (b) alone using the built in speaker and microphone) in a social or professional context as mobile phone and 'filofax', and as entertainment centre, with or without connectivity to 'home base' servers and IT environment. For more traditional working requiring keyboard and screen the notebook computer would be used, probably without the PDA. The two might be used together with data collection validation / calibration software on the notebook computer and sensors attached to the PDA.

The balance between that (data, software) which is on servers accessed over the network and that which is on (one of) the end-user device(s) depends on the mode of work, speed of required response and likelihood of interrupted connections. Clearly the GRIDs environment is ideal for such a user to be connected.

6 Use of the Technology

The GRIDs environment provides (when completed) an environment in which an end-user can have homogeneous, personalised, customised access to heterogeneous data sources, software and therefore relevant information. The end-user has access to global information and software sources, and to compute resources and specialised visualisation and VR (Virtual Reality) facilities, possibly subject to rights issues and / or charging.

The Ambient computing environment provides end-user connectivity into such a GRIDs environment for professional, scientific work and also for management, administration and social functions.

It is a small step in imagination to envisage the end-user utilising the GRIDs environment for analysis, modelling, simulation, visualisation of data to information used in improved understanding and decision-making. Furthermore data mining may uncover new hypotheses and advance science or provide new business opportunities. Similarly, the end-user would use the ambient computing environment for travelling salesman data collection and geo-location, for advising 'home base' of what is observed and receiving suggestions or instructions on what to do next. Such a system with GRIDs and Ambient Computing provides an ideal basis for both professional and personal lifestyle support.

7 Conclusion

The GRIDs architecture will provide an IT infrastructure to revolutionise and expedite the way in which we handle information and decision-making. The Ambient Computing architecture will revolutionise the way in which the IT infrastructure intersects with our lives, both professional and social. The two architectures in combination will revolutionise human behaviour.

References

[FoKe98] I Foster and C Kesselman (Eds). The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kauffman 1998

[Je00] K G Jeffery. 'Metadata': in Brinkkemper,J; Lindencrona,E; Solvberg,A: 'Information Systems Engineering' Springer Verlag, London 2000. ISBN 1-85233-317-0.

[JeHuKaWiBeMa94] K G Jeffery, E K Hutchinson, J R Kalmus, M D Wilson, W Behrendt, C A Macnee, 'A Model for Heterogeneous Distributed Databases' Proceedings BNCOD12 July 1994; LNCS 826 pp. 221–234 Springer-Verlag 1994

Natix: A Technology Overview

Thorsten Fiebig[1], Sven Helmer[2], Carl-Christian Kanne[3], Guido Moerkotte[2], Julia Neumann[2], Robert Schiele[2], and Till Westmann[3]

[1] Software AG Thorsten.Fiebig@softwareag.com
[2] Universität Mannheim {mildenbe|rschiele}@uni-mannheim.de, {helmer|moerkotte}@informatik.uni-mannheim.de
[3] data ex machina GmbH {kanne|westmann}@data-ex-machina.de

Abstract. Several alternatives to manage large XML document collections exist, ranging from file systems over relational or other database systems to specifically tailored XML base management systems. In this paper we review Natix, a database management system designed from scratch for storing and processing XML data. Contrary to the common belief that management of XML data is just another application for traditional databases like relational systems, we indicate how almost every component in a database system is affected in terms of adequacy and performance. We show what kind of problems have to be tackled when designing and optimizing areas such as storage, transaction management comprising recovery and multi-user synchronization as well as query processing for XML.

1 Introduction

As XML [5] becomes widely accepted, the need for systematic and efficient storage of XML documents arises. For this reason we have developed Natix, a native XML base management system (XBMS) that is custom tailored to the processing of XML documents. A general-purpose XBMS for large-scale XML processing has to fulfill several requirements: (1) To store documents effectively and to support efficient retrieval and update of these documents or parts of them. (2) To support standardized declarative query languages like XPath [7] and XQuery [4]. (3) To support standardized application programming interfaces (APIs) like SAX [21] and DOM [18]. (4) Last but not least a safe multi-user environment via a transaction manager has to be provided including recovery and synchronization of concurrent access.

We give an overview of the techniques used in the major components of Natix' runtime system. We review a storage format that clusters subtrees of an XML document tree into physical records of limited size. Our storage format meets the requirement for the efficient retrieval of whole documents and document fragments. The size of a physical record containing the XML subtree is typically far larger than the size of a physical record representing a tuple in a relational database system. This affects recovery. We briefly describe the techniques *subsidiary logging* to reduce the log size, *annihilator undo* to accelerate undo and

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 12–33, 2003.

selective redo to accelerate restart recovery. Our flexible multi-granularity locking protocol features an arbitrary level of granularities, which guarantees serializability and allows high concurrency even if transactions directly access some node in a document tree without traversing down from the root. (Note that existing tree locking protocols fail here.) Finally, we give a brief overview of Natix' query execution engine. Evaluating XML queries differs vastly from evaluating SQL queries. For example, SQL queries never produce an XML document.

2 Architecture

This section contains a brief overview of Natix' system architecture. We identify the different components of the system and their responsibilities.

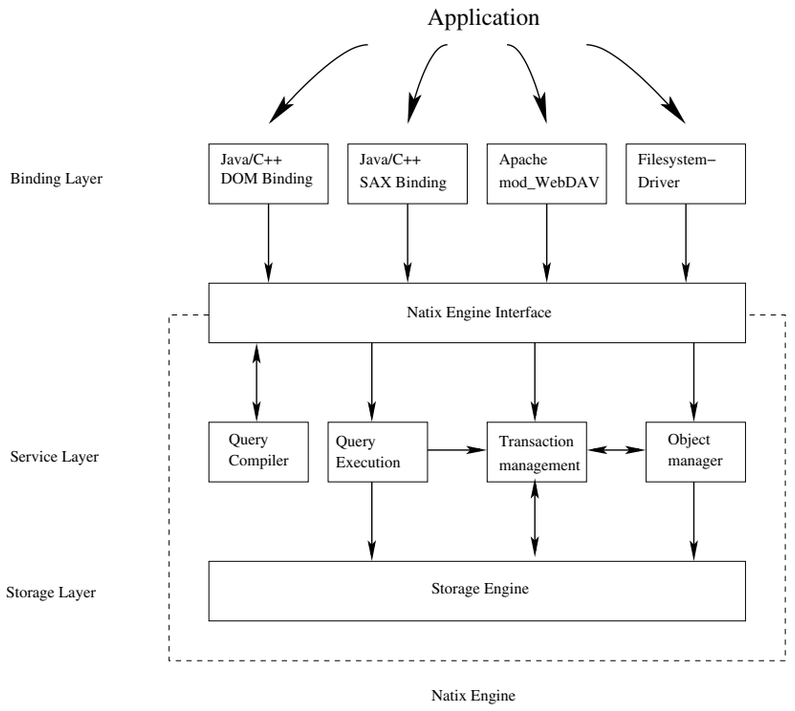


Fig. 1. Architectural overview

Natix' components form three layers (see Fig. 1). The bottommost layer is the *storage layer*, which manages all persistent data structures. On top of it, the *service layer* provides all DBMS functionality required in addition to simple storage and retrieval. These two layers together form the *Natix engine*.

Closest to the applications is the *binding layer*. It consists of the modules that map application data and requests from other APIs to the Natix Engine Interface and vice versa.

2.1 Storage Layer

The storage engine contains classes for efficient XML storage, indexes and metadata storage. It also manages the storage of the recovery log and controls the transfer of data between main and secondary storage. An abstraction for block devices allows to easily integrate new storage media and platforms apart from regular files. Details follow in Sec. 3.

2.2 Service Layer

The database services communicate with each other and with applications using the *Natix Engine Interface*, which provides a unified facade to specify requests to the database system. These requests are then forwarded to the appropriate component(s). After the request has been processed and result fields have been filled in, the request object is returned to the caller. Typical requests include 'process query', 'abort transaction' or 'import document'.

There exist several service components that implement the functionality needed for the different requests. The Natix *query execution* engine (NQE), which efficiently evaluates queries, is described in Sec. 5. The *query compiler* translates queries expressed in XML query languages into execution plans for NQE. Additionally, a simple compiler for XPath [16] is available. They both are beyond the scope of the paper. *Transaction management* contains classes that provide ACID-style transactions. Components for recovery and isolation are located here. Details can be found in section 4. The *object manager* factorizes representation-independent parts for transferring objects between their main and secondary memory representations since this transformation is needed by several APIs.

All of these components bear challenges with respect to XML, which are related to the different usage profiles (coarse grain vs. small grain processing). Typically, a simple mapping of operations on coarse granularities to operations on single nodes neutralizes a lot of performance potential. If both access patterns have to be supported in an efficient way, sophisticated techniques are needed.

2.3 Binding Layer

XML database management is needed by a wide range of application domains. Their architectural and interface requirements differ. Apart from the classic client-server database system, there are scenarios with Internet access, possibly using protocols like HTTP or WebDAV [13]. For embedded systems it might be more appropriate to use an XML storage and processing library with a direct function call interface. For legacy applications that can only deal with plain files,

the database has to be mounted as a file system. Other interfaces will arise when XML database systems are more widely used.

The responsibility of the binding layer is to map between the Natix Engine Interface and different application interfaces. Each such mapping is called a *binding*.

Applications may call the Natix Engine Interface directly. However, for rapid application development, it is often desirable to have interface abstractions that are closer to the application's domain. An example for such a higher-level API is the file system interface, a demonstration of which is available for download [8]. Using this binding, documents, document fragments, and query results can be accessed just like regular files. The documents' tree structure is mapped into a directory hierarchy, which can be traversed with any software that knows how to work with the file system. Internally, the application's file system operations are translated into Natix Engine Interface requests for exporting, importing, or listing documents.

Wherever feasible, the specification of a request to the Natix Engine is not only possible using C++ data types, but also by a simple, language independent string. A small parser is part of the Engine Interface. It translates strings into request objects. This simple control interface for the system can easily be incorporated into generic high-level APIs: by using request strings as URLs, for example, the HTTP protocol can be used to control the database system.

3 Storage Engine

At the heart of every data base management system lies the storage engine that manages all persistent data structures and their transfer between main and secondary memory. The system's overall speed, robustness, and scalability are determined by the storage engine's design.

We briefly summarize the architecture of the storage engine, describe our novel XML storage method, and the employed indexing techniques. For a more detailed look at the subject, see [9].

3.1 Architecture

Storage in Natix is organized into *partitions*, which represent storage devices that can randomly read and write disk pages (random-access block devices). Currently, the following flavors are available: Unix files, raw disk access under Linux and Solaris, and C++ iostreams.

Disk pages are logically grouped into *segments*, which export the main interfaces to the storage system. They implement large, persistent object collections, where an object may be larger than a page (depending on the segment type). The page collection used to store the object collections is maintained using an extent-based system [28] that organizes segments into consecutive page groups (*extents*) of variable size. Intra-segment free space management is done using a Free Space Inventory (FSI) [23] describing the allocation state and free space

on pages. A caching technique similar to [20] is used. We support several different segment types, each implementing a different kind of object collection. The most important segment types are standard *slotted page segments* supporting unordered collections of variable-size records, index segments (e.g. for B-Trees) and *XML segments*. The *XML segments* for XML document collections are novel and described below.

Disk pages resident in main memory are managed by the *buffer manager*, which is responsible for transferring pages between main and secondary memory and synchronizing multithreaded access to the data pages using latches. Special calls exist to avoid I/O for reading or writing newly allocated or deallocated pages.

While a page resides in main memory, it is associated with a *page interpreter object* that abstracts from the actual data format on the page. The page interpreters form a class hierarchy with a single base class, from which one or more data-type specific classes are derived for each segment type. For example, a B-Tree segment might use one page interpreter class for inner pages and leaf pages each.

3.2 XML Storage

One of the core segment types in Natix is the novel XML storage segment, which manages a collection of XML documents. Our method offers the following distinguishing features: (1) Subtrees of the original XML document are stored together in a single (physical) record (and, hence, are clustered). Thereby, (2) the inner structure of the subtrees is retained. (3) To satisfy special application requirements, their clustering requirements can be specified by a *split matrix*.

We start our description with the logical document data model used by the XML segment to work with documents, and the storage format used by the XML page interpreters to work with document fragments that fit on a page. Then, we show how the XML segment type maps logical documents that are larger than a page to a set of document fragments possibly spread out on different disk pages. Finally, we briefly sketch the maintenance algorithm for this storage format.

```
<SPEECH>
<SPEAKER>OTHELLO</SPEAKER>
<LINE>Let me see your eyes;</LINE>
<LINE>Look in my face.</LINE>
</SPEECH>
```

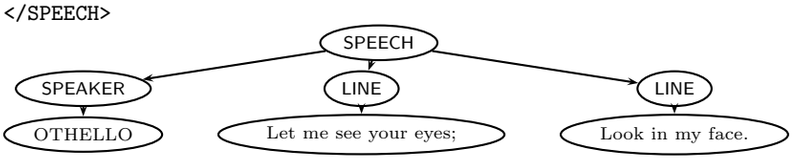


Fig. 2. A fragment of XML with its associated logical tree

Logical data model. The XML segment's interface allows to access an unordered set of trees. New nodes can be inserted as children or siblings of existing nodes, and any node (including its induced subtree) can be removed. The individual documents are represented as ordered trees with non-leaf nodes labeled with a symbol taken from an alphabet Σ_{Tags}. Leaf nodes can, in addition to a symbol from Σ_{Tags}, be labeled with arbitrarily long strings over a different alphabet. Figure 2 shows an XML fragment and its associated tree.

Mapping between XML and the logical model. A small wrapper class is used to map the XML model with its node types and attributes to the simple tree model and vice versa. The wrapper uses a separate segment to map tag names and attribute names to integers, which are used as Σ_{Tags}. All the documents in one XML segment share the same mapping. The interface of this so-called *declaration table* allows for small, hashed per-thread caches for those parts of the mapping that are in use. The caches can be accessed very fast without any further synchronization.

Elements are mapped one-to-one to tree nodes of the logical data model. Attributes are mapped to child nodes of an additional *attribute container* child node, which is always the first child of the element node the attributes belong to. Attributes, PCDATA, CDATA nodes and comments are stored as leaf nodes. External entity references are expanded during import, while retaining the name of the referenced entity as a special internal node. Some integer values are reserved in addition to the ones for tag and attribute names, to indicate attribute containers, text nodes, processing instructions, comments and entity references.

XML page interpreter storage format. A (physical) record is a sequence of bytes stored on a single page. The logical data tree is partitioned into subtrees (see Sec. 3.2). Each subtree is stored in a single record and, hence, must fit on a page. Additionally to the subtree, a record contains a pointer to the record containing the parent node of the root node of its subtree (if it exists), and the identifier of the document the contained subtree belongs to.

XML page interpreters are used to maintain the subtrees' records on data pages. They are based on a regular slotted page implementation, which maintains a collection of variable-length records on a data page. Each record is identified by a slot number which does not change even if the record is moved around on the page for space management reasons.

We introduced several optimizations for representing subtrees inside records (for details see [9]). Native storage that only consumes about as much space as plain file XML documents is the main result of these optimizations.

XML segment mapping for large trees. Typical XML trees may not fit on a single disk page. Hence, document trees must be partitioned. Typical BLOB (*binary large object*) managers achieve this by splitting large objects at arbitrary byte positions [3,6,19]. We feel that this approach wastes the available structural

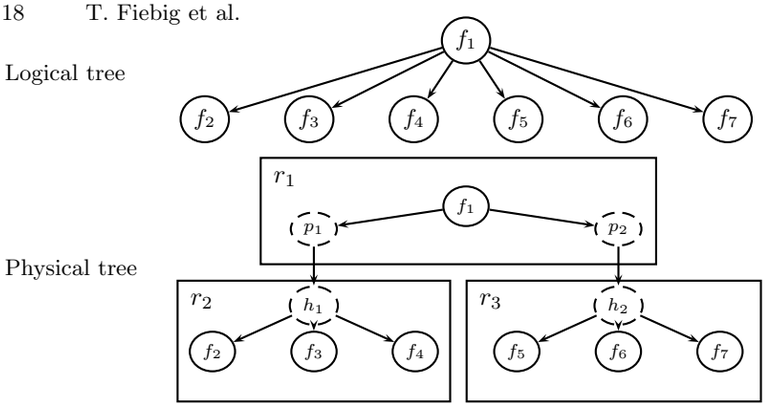


Fig. 3. One possibility for distribution of logical nodes onto records

information. Thus, we *semantically* split large documents based on their tree structure, i.e. we partition the tree into subtrees.

When storing the physical representation of a data tree to disk, we distinguish between two major classes of nodes: *facade nodes* and *scaffolding nodes*. The former represent the logical nodes, i.e. these are the nodes the user gets to see. The latter are used for organizing the data, i.e. for linking subtrees together. Facade nodes are divided further into *aggregate nodes* for representing the inner nodes and *literal nodes* for representing the leaf nodes of an XML document's logical tree. Among scaffolding nodes we find *proxy nodes*, which refer to subtrees not stored in the same record, and *helper aggregate nodes*, which group together (a subset of) children of a node.

Figure 3 shows a logical tree and one possible physical tree. In this example f_1 is an aggregate (facade) node and f_2 to f_7 are literate (facade) nodes. The scaffolding nodes (marked by dashed ovals) are divided into the proxy nodes p_1 and p_2 and the helper aggregate nodes h_1 and h_2. The physical tree is distributed over the records r_1, r_2 and r_3.

Updating documents. We briefly present Natix' algorithm for dynamic maintenance of physical trees (for more technical details, see [9]). The principal problem addressed is that a record containing a subtree grows larger than a page. In this case, the subtree has to be partitioned into several subtrees, each fitting on a page. Scaffolding nodes (proxies and maybe aggregates) have to be introduced into the physical tree to link the new records together.

When inserting a new node f_n into the logical data tree as a child node of another node, we have to decide on an insertion location in the physical tree. In the presence of scaffolding nodes, several alternatives may exist. In Natix, this choice is determined by the split matrix (which we will describe in a moment).

Having decided on the insertion location, it is possible that the designated record's disk page is full. First, the system tries to move the record to a page with more free space. If this is not possible because the record as such exceeds the net page capacity, the record is split. Similar to B-trees, we divide the data

elements on a page (in our case nodes of a subtree) into two partitions and a *separator*. However, splitting trees is more complex than merely looking for a median, which is the couterpart of the separator in B-Trees.

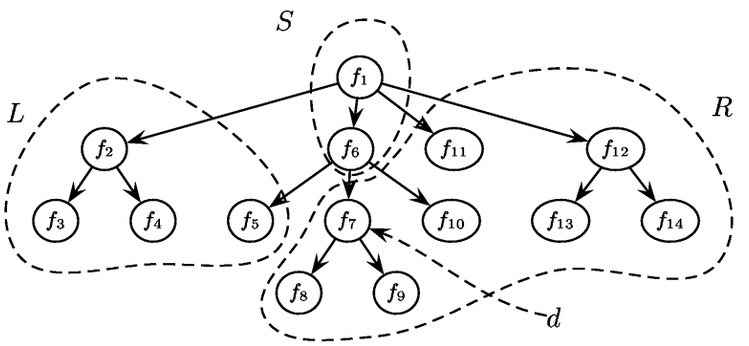


Fig. 4. Splitting a subtree

In Natix's split algorithm, the separator S is defined by a single node d. Figure 4 shows an exemplary split situation, where $d = f_7$. The nodes on the path from the subtree's root to d, but excluding d, form the separator. The left partition L comprises all nodes that are left siblings of nodes in S, and all descendants of those nodes. The nodes which are not part of S or L make up the right partition R.

When searching for a separator, the split algorithm has to consider the ratio of the sizes of L and R we aim at. Typical settings for this configuration parameter, called the *split target*, are either (roughly) equal size for L and R or very small R partitions to leave room for anticipated insertions (when creating a tree in pre-order). Another configuration parameter available for fine-tuning is the *split tolerance*, which states how much the algorithm may deviate from this ratio. Essentially, the split tolerance specifies a minimum size for d's subtree. Subtrees smaller than this value are not split, but completely moved into one partition to prevent fragmentation.

To determine d, the algorithm starts at the subtree's root and recursively descends into the child whose subtree contains the physical "middle" (or the configured split target) of the record. It stops when it reaches a leaf or when the size of the subtree in which it is about to descend is smaller than allowed by the split tolerance parameter. In the example in figure 4, the size of the subtree below f_7 was smaller than the split tolerance, otherwise the algorithm would have descended further and made $d = f_7$ part of the separator.

After splitting the subtree we have to distribute the nodes of the partitions L and R among records and generate the appropriate proxy nodes. The nodes of the separator move up to replace the proxy node in the parent record. This is done

via recursive insertion (the parent record may overflow during this procedure). Finally, we insert the new node into its designated record.

With the help of a *split matrix* we can introduce the requirements of an application into the strategy of the split algorithm. Let us illustrate this by briefly mentioning scenarios, where a split matrix may help. If we frequently navigate from one type of parent node to a certain type of child node, we want to prevent the split algorithm from storing them in separate records. In other contexts, we want certain kinds of subtrees to be always stored in a separate record, for example to collect some kinds of information in their own physical database area. In addition, since Natix's basic granularity for concurrency control is a physical record (see Sec. 4.6), concurrency can be enhanced by placing certain node types into separate records.

The split matrix S consists of elements $s_{ij}, i, j \in \Sigma_{\text{Tags}}$. The elements express preferences regarding the clustering behavior of a node x with label j as child of a node y with label i:

$$s_{ij} = \begin{cases} 0 & x \text{ is always kept as a standalone record and never clustered with } y \\ \infty & x \text{ is kept in the same record with } y \text{ as long as possible} \\ \textbf{other} & \text{the algorithm may decide} \end{cases}$$

3.3 Index Structures in Natix

In order to support query evaluation efficiently, we need powerful index structures. The main problem in building indexes for XML repositories is that ordinary full text indexes do not suffice, as we also want to consider the structure of the stored documents. Here we describe the approaches taken by us to integrate indexes for XML documents in Natix. We have followed two principle avenues of approach. On the one hand we enhanced a traditional full text index, namely inverted files, in such a way as to be able to cope with semistructured data. As will be shown, we opted for a versatile generic approach, InDocs (for Inverted Documents) [22], that can deal with a lot more than structural information. On the other hand we developed a novel index structure, called XASR (eXtendend Access Support Relation) [12], for Natix.

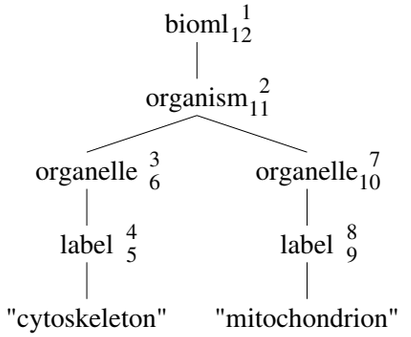
Full Text Index Framework. Inverted files are the index of choice in the information retrieval context [1,29]. In the last years the performance of inverted files improved considerably, mostly due to clever compression techniques. Usually inverted files store lists of document references to indicate in which documents certain words appear. Often offsets within a document are also saved along with the references (this can be used to evaluate near-predicates, for example). However, in practice inverted files are handcrafted and tuned for special applications. Our goal is to generalize this concept by storing arbitrary contexts (not just offsets) with references without compromising the performance. Let us briefly sketch the architecture of the list implementation in Natix:

Index. The main task of the class Index is to map search terms to list identifiers and to store those mappings persistently. It also provides the main interface for the user to work with inverted files.

ListManager. This class maps the list identifiers to the actual lists, so it is responsible for managing the directory of the inverted file. If the user works directly with identifiers and not with search terms, it is possible to use ListManager directly. We have implemented efficient methods for bulkload and bulkremoval of lists, as well as for concatenation of lists, namely union and intersection.

FragmentedList, ListFragment. We describe these modules together, because they are tightly coupled with each other. ListFragment is an implementation of lists that need at most one page of memory to store. The content of a fragment can be read and written sequentially. All fragments that belong to one inverted list are linked together and can be traversed sequentially. The job of the class FragmentedList is to manage all the fragments of one list and control insertions and deletions on this list.

ContextDescription. This class determines the actual representation in which data is stored in a list. With representation we do not only mean what kind of data is stored, but also the compression technique that is used. We have implemented the traditional context consisting of a document ID and the positions within the document where a certain term appears. More importantly, we devised contexts for XML data. A simple node context consists of a document ID, a node ID, and the position of the search term within the node, whereas a more complex node context also considers structural information (e.g. d_{min} and d_{max} values, which are described in the next subsection).



(a) Example document tree

d_{min}	d_{max}	eType	docID	parent_d_{min}
1	12	'bioml'	0	NULL
2	11	'organism'	0	1
3	6	'organelle'	0	2
4	5	'label'	0	3
7	10	'organelle'	0	2
8	0	'label'	0	7

(b) corresponding table

Fig. 5. XASR

eXtended Access Support Relations. An extended access support relation (XASR) is an index that preserves the relationships between the nodes. This is done by labeling the nodes of an XML document tree by depth-first traversal (see Figure 5). We assign each node a d_{min} value (when we enter the node for the first time) and a d_{max} value (when we finally leave the node). For each node in the tree we store a row in an XASR table with information on d_{min}, d_{max}, the name of the element tag, the document ID, and the d_{min} value of the parent node (see Figure 5).

XASR is combined with regular full text indexes that supply the node numbers of nodes containing words we are searching for. That means, if we are looking for nodes containing specific words or nodes of a certain type in a path, we also join these nodes to the nodes fetched by XASR.

A path in a query is translated into a sequence of joins on an XASR table (one join for each location step). Let x_i and x_{i+1} be two nodes in the path. The first part of the join predicate checks if both nodes are in the same document, so we have to test for equality of x_i.docID and x_{i+1}.docID. Depending on the location step we also have to introduce one of the following predicates:

x_i/child :: x_{i+1}	x_i.dmin = x_{i+1}.parentID
x_i/parent :: x_{i+1}	x_i.parentID = x_{i+1}.dmin
x_i/descendant :: x_{i+1}	x_i.dmin < x_{i+1}.dmin $\wedge$ x_i.dmax > x_{i+1}.dmax
x_i/descendant-or-self :: x_{i+1}	x_i.dmin $\leq$ x_{i+1}.dmin $\wedge$ x_i.dmax $\geq$ x_{i+1}.dmax
x_i/ancestor :: x_{i+1}	x_i.dmin > x_{i+1}.dmin $\wedge$ x_i.dmax < x_{i+1}.dmax
x_i/ancestor-or-self :: x_{i+1}	x_i.dmin $\geq$ x_{i+1}.dmin $\wedge$ x_i.dmax $\leq$ x_{i+1}.dmax
x_i/preceding :: x_{i+1}	x_i.dmin > x_{i+1}.dmax
x_i/following :: x_{i+1}	x_i.dmax < x_{i+1}.dmin
x_i/preceding-sibling :: x_{i+1}	x_i.dmin > x_{i+1}.dmax $\wedge$ x_i.parentID = x_{i+1}.parentID
x_i/following-sibling :: x_{i+1}	x_i.dmax < x_{i+1}.dmin $\wedge$ x_i.parentID = x_{i+1}.parentID

For additional details on query processing with XASRs see [12].

4 Transaction Management

Enterprise-level data management is impossible without a transaction concept. The majority of advanced concepts for versioning, workflow and distributed processing depends on primitives based on the proven foundation of *atomic*, *durable* and *serializable* transactions.

Consequently, to be an effective tool for enterprise-level applications, Natix has to provide transaction management for XML documents with the above-mentioned properties. The transaction components supporting transaction-oriented programming in Natix are the subject of this section. The two areas covered are recovery and isolation, in this order.

For recovery, we adapt the ARIES protocol [24]. We further introduce the novel techniques of *subsidiary logging*, *annihilator undo*, and *selective redo* to exploit certain opportunities to improve logging and recovery performance which prove — although present in many environments — especially effective when

large records with a variable structure are managed. This kind of record occurs when XML subtrees are clustered in records as in Natix' storage format.

For synchronization, an S2PL-based scheduler is introduced that provides lock modes and a protocol that are suitable for typical access patterns occurring for tree-structured documents. The main novelties are that (1) granularity hierarchies of arbitrary depth are supported and (2) contrary to existing tree locking protocols, jumps into the tree do not violate serializability.

4.1 Architecture

During system design, we paid special attention to a recovery architecture that treats separate issues (among them page-level recovery, logical undo, and metadata recovery) in separate classes and modules. Although this is not possible in many cases, we made an effort to separate the concepts as much as possible, to keep the system maintainable and extendible.

Although most components need to be extended to support recovery, in most cases this can be done by inheritance and by extension of base classes, allowing for the recovery-independent code to be separate from the recovery-related code of the storage manager.

4.2 Recovery Components

We will not explain the ARIES protocol here, but concentrate on extensions and design issues related to Natix and XML. A description of ARIES can be found in the original ARIES paper [24] and in books on transaction processing (e.g. [15,27]). In the following we give a brief description of the recovery components in Natix.

Log records. Natix writes a recovery log describing the actions of all update transactions using *log records*.

Natix log records consist of fields that describe the involved transaction, log record type and operation code information, some flags, and the IDs of involved objects and their before and/or after images to redo and/or undo the operation. The log records of a transaction are linked together into a pointer chain. Special care is taken to set up this pointer chain to enable partial rollbacks and idempotent system restart.

Segments. From the view of the recovery subsystem, the segment classes comprise the main interaction layer between the storage subsystem and the application program. As part of their regular operations, application programs issue requests to modify or access the persistent data structures managed by the segments.

The segments map operations on their data structures — which can be larger than a page — to sequences of operations on single pages. The page interpreters deal with logging and recovery for operations on pages. This means that the code for multi-page data structures is the same for recoverable and nonrecoverable

variants of the data structure, it only has to instantiate different page interpreter versions in the recoverable case. This is a significant improvement in terms of maintainability of the system, because less code is necessary, and recovery-related code is separate from recovery-independent code. Only some high-concurrency multi-page data structures require special recovery code in the segments. These include B-Trees and metadata storage.

Page interpreters. The page interpreter classes are responsible for page-level logging and recovery. They create and process all page-level (i.e. the majority of) log records. The page-level log records use physical addressing of the affected page, logical addressing within the page, and logical specification of the performed update operation. This is called *physiological logging* [15].

For every page type in the page interpreter hierarchy that has to be recoverable, there exists a derived page interpreter class with an identical interface that, in addition to the regular update operations, logs all performed operations on the page and is able to interpret the records during redo and undo.

Buffer manager. The buffer manager is controlling the transfer of pages between main and secondary memory. Although ARIES is independent of the replacement strategy used when caching pages [24], the buffer manager enables adherence to the ARIES protocol by notifying other components about page transfers between main and secondary memory and by logging information about the buffer contents during checkpoints.

Recovery Manager. The recovery manager orchestrates system activity during undo processing, redo processing and checkpointing. It is stateless and serves as a collection of the recovery-related top-level algorithms for restart and transaction undo. During redo and undo, it performs log scans using the log manager (see below) and forwards the log records to the responsible objects (e.g. segments and page interpreters) for further processing.

Log manager. The log manager provides the routines to write and read log records, synchronizing access of several threads that create and access log records in parallel.

It keeps a part of the log buffered in main memory (using the Log Buffer as explained below) and employs special partitions, log partitions, to store log records.

Transaction manager. The transaction manager maintains the data structures for active transactions and is used by the application programs to group their operations into transactions.

Each transaction is associated with a control block that includes recovery-related per-transaction information.

4.3 Subsidiary Logging

Existing logging-based recovery systems follow the principle that every modification operation is immediately preceded or followed by the creation of a log record

for that operation. An *operation* is a single update primitive (like insert, delete, modify a record or parts of a record). *Immediately* usually means before the operation returns to the caller. In the following, we explain how Natix reduces log size and increases concurrency, boosting overall performance, by relaxing both constraints.

Suppose a given record is updated multiple times by the same transaction. This occurs frequently when using a storage layout that clusters subtrees into records, for example, when a subtree is added node by node. It is desirable that a composite update operation is logged as one big operation, for example by logging the complete subtree insertion as one operation: merging the log records avoids the overhead of log record headers for each node (which can be as much as 100% for small nodes), and reduces the number of serialized calls to the log manager, increasing concurrency.

In the following, we sketch how Natix' recovery architecture supports such optimizations, especially in the case of XML data.

In Natix, physiological logging is completely delegated to the page interpreters: How the page interpreters create and interpret log records is up to them. Each page interpreter has its own state, which it can use to collect logging information for the associated page without actually transferring them to the log manager, thus keeping a private, *subsidiary log*. The interpreter may reorder, modify, or use some optimized representation for these private log entries before they are published to the log manager. A set of rules guarantees recoverability by controlling when the subsidiary logs must be flushed to the system log manager.

The example above refers to a typical kind of update operations performed by applications using Natix, the insertion of a subtree of nodes into a document. Often applications insert a subtree by inserting single nodes. To avoid excessive logging in such situations, Natix employs subsidiary logs for XML data page modifications.

With subsidiary logging, larger log records describing the effects of many operations are generated. To avoid CPU-intensive copying and dynamic memory management, the contents of the subsidiary log records are represented using pointers into the page contents in the buffer manager. The page interpreters use the global log manager to create persistent log records only when necessary for recoverability. If the newly created subtree(s) fit into the buffer, the log volume created is nearly equal to the size of the data. This is possible as only the "final" state of the subtrees is logged upon commit, and only a few log record headers are created (one for each subtree record), amortizing the logging overhead for the large number of small objects.

4.4 Annihilator Undo

Transaction undo often wastes CPU resources, because more operations than necessary are executed to recreate the desired result of a rollback. For example, any update operations to a record that has been created by the same transaction need not be undone when the transaction is aborted, as the record is going to be deleted as a result of transaction rollback anyway. Refer to Fig. 6 which shows a

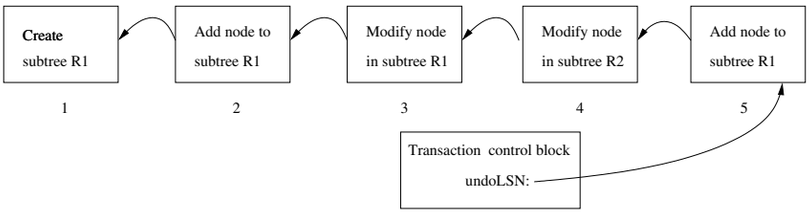


Fig. 6. Log records for an XML update transaction

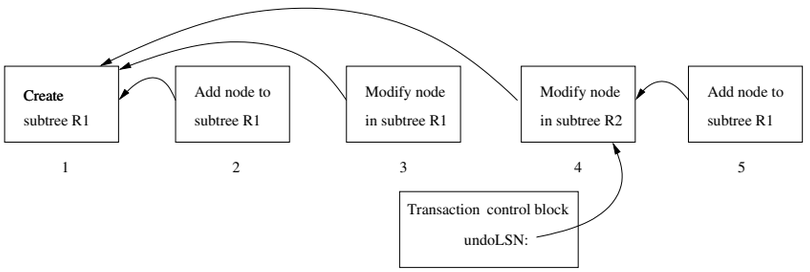


Fig. 7. Chaining with check for annihilators

transaction control block pointing to its chain of log records. During undo, the records would be processed in the sequence $5, 4, 3, 2, 1$. Looking at the operations' semantics, undo of records 4 and 1 would be sufficient, as undo of 1 would delete record R1, implicitly undoing all changes to R1.

For our XML storage organization, creating a record and performing a series of updates to the contained subtree afterwards is a typical update pattern for which we want to avoid unnecessary overhead in case of undo. And since the abort probability of transactions can be much higher than in traditional database applications, undos occur more frequently. For example, online shoppers often fill their shopping carts and then decide not to go to the cashier.

Natix implements a technique based on *annihilators*. Annihilators are operations whose undo already implies the undo of a sequence of other operations. The creation of R1 above is such an annihilator. By storing information about annihilators, Natix is able to link log records in a way that avoids unnecessary undo operations (Fig.7).

4.5 Selective Redo and Selective Undo

The ARIES protocol is designed around the redo-history paradigm, meaning that the complete state of the cached database is restored after a crash, including updates of loser transactions. After the redo pass has accomplished this, the

following undo pass may unconditionally undo all changes of loser transactions in the log.

In the presence of fine-granularity locking, when multiple transactions may access the same page concurrently, the redo-history method is necessary for proper recovery, together with writing log records that describe actions taken during undo (compensation log records, or CLRs). Unfortunately, this may cause pages that only contain updates by loser transactions to be loaded and modified during restart, although their on-disk version (without updates) already reflects their desired state as far as restart recovery is concerned.

If a large buffer is employed and concurrent access to the same page by different transactions is rare, ARIES' restart performance is less than optimal, as it is likely that all uncommitted updates were only in the buffer at the time of the crash, and thus no redo and undo of loser transactions would be necessary.

There exists an approach to address this problem [25]. However, it is only applicable to systems with page-level locking, and requires extra fields in the log record headers. Natix employs a similar technique that also works with record-level locking, and avoids to blow up log records for the sake of the special case of restart acceleration.

4.6 Synchronization Components

Since XML documents are semi-structured, we cannot apply synchronization mechanisms used in traditional, structured relational databases. XML's tree structure suggests using tree locking protocols as described e.g. in [2,27]. However, these protocols fail in the case of typical XML applications, as they expect a transaction to always lock nodes in a tree in a top-down fashion. Navigation in XML documents often involves jumps right into the tree by following an IDREF or an index entry. This jeopardizes serializability of traditional tree locking protocols. Another objection to tree locking protocols is the lack of lock escalation. Lock escalation is a proven remedy for reducing the number of locks held at a certain point in time. A straightforward approach taken in some commercial products is to lock whole XML documents, limiting concurrency in an unsatisfactory manner. In order to achieve a high level of concurrency, one might consider locking at the level of XML nodes, but this results in a vast amount of locks. We strive for a balanced solution with a moderate number of locks while still preserving concurrent updates on a single document.

Although a traditional lock manager [15] supporting multi granularity locking (MGL) and strict two-phase locking (S2PL) can be used as a basis for the locking primitives in Natix, we need several modifications to guarantee the correct and efficient synchronization of XML data. This involves an MGL hierarchy with an arbitrary number of levels and the handling of IDREF and index jumps into the tree. More information about the protocol and its implementation can be found in [26].

5 Natix Query Execution Engine

A query is typically processed in two steps: the query compiler translates a query into an optimized query evaluation plan, and then the query execution engine interprets the plan. We describe Natix' query execution engine. The query compiler is far out of the scope of this paper.

5.1 Overview

While designing the Natix Query Execution Engine (NQE) we had three design goals in mind: efficiency, expressiveness, and flexibility. Of course, we wanted our query execution engine to be efficient. For example, special measures are taken to avoid unnecessary copying. Expressiveness means that the query execution engine is able to execute all queries expressible in a typical XML query language like XQuery [4]. Flexibility means that the algebraic operators implemented in the Natix Physical Algebra (NPA)—the first major component of NQE—are powerful and versatile. This is necessary to keep the number of operators as small as possible. Let us illustrate this point. The result of a query can be an XML document or fragment. It can be represented as text, as a DOM tree [18], or as a SAX event stream [21]. Since we did not want to implement different algebraic operators to perform the implied different result constructions, we needed a way to parameterize our algebraic operators in a very flexible way. We use small programs which are then interpreted by the Natix Virtual Machine (NVM)—the second major component of NQE.

Let us now give a rough picture of NPA and NVM. NPA works on sequences of tuples. A tuple consists of a sequence of attribute values. Each value can be a number, a string, or a node handle.

NPA operators are implemented as *iterators* [14]. They usually take several parameters which are passed to the constructor. The most important parameters are programs for the Natix Virtual Machine (NVM). Take for example the classical `Select` operator. Its predicate is expressed as an NVM program. The `Map` operator takes as parameter a program that computes some function(s) and stores the result(s) in some attribute. Other operators may take more than one program. For example, a typical algebraic operator used for result construction takes three NVM programs.

The rest of the section is organized as follows. We first introduce the Natix Virtual Machine. Then we describe the Natix Physical Algebra.

5.2 Natix Virtual Machine

The Natix Virtual Machine interprets commands on register sets. Each register set is capable of holding a tuple, e.g. one register holds one attribute value. At any time, an NVM program is able to access several register sets. There always exists a global register set which contains information that is global to but specific for the current plan execution. It contains information about partitions, segments, lookup tables, and the like. It is also used for intermediate results or to pass

information down for nested query execution. Between operators, the tuples are stored in the register sets Z and Y where Y is only available for binary operators. In case of a join operator, Z contains an outer and Y an inner tuple.

It is database lore that during query execution most of the time is spent on copying data around. We have been very careful to avoid unnecessary copying in NQE. Let us briefly describe our approach here. In order to avoid unnecessary copying, pointers to registers sets are passed among different NPA operators. If there is no pipeline breaker in a plan, only one Z register set is allocated and its address is passed down the tree. The general rule for Z registers used by pipelined operators is the following: memory is passed from top to bottom and content from bottom to top.

The XML specific part of NVM contains about 150 commands. Among these are simple operations that copy a node handle from one register to another, compare two handles, print the XML fragment rooted at a handle with or without markup and the like. The main portion of the XML specific commands consists of navigation operations roughly corresponding to the axes in XPath. These commands retrieve the attributes of an element node, its children, or its descendants. Further details about NVM are available in [9].

5.3 Natix Physical Algebra

Query languages for XML (for example XQuery) often provide a three-step approach to query specification. The first part (`let` and `for` in XQuery) specifies the generation of variable bindings. The second part (`where` in XQuery) specifies how these bindings are to be combined and which combinations are to be selected for the result construction. The final part (`return` in XQuery) specifies how a sequence of XML fragments is to be generated from the combined and selected variable bindings.

Reflecting this three step approach, NPA operators exist to support each of these steps. The middle step—binding combination and selection—can be performed by standard algebraic operators borrowed from the relational context. Those provided in NPA are a select, map, several join and grouping operations, and a sort operator. We concentrate on the XML specific operations for variable binding generation and XML result construction.

Variable binding generation. At the bottom of every plan are scan operations. The simplest is an expression scan (`ExpressionScan`) which generates tuples by evaluating a given expression. It is used to generate a single tuple containing the root of a document identified by its name. The second scan operator scans a collection of documents and provides for every document a tuple containing its root. Index scans complement the collection of scan operations.

Besides the scan operations `UnnestMap` is used to generate variable bindings for XPath expressions. An XPath expression can be translated into a sequence of `UnnestMap` operations. Consider for example the XPath expression `/a//b/c`. It can be translated into

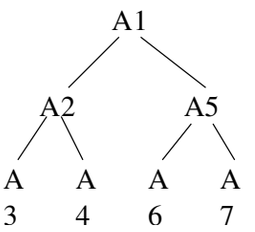


Fig. 8. A Sample XML Document

$$\text{UnnestMap}_{\$4=child(\$3,c)}(\text{UnnestMap}_{\$3=desc(\$2,b)}(\text{UnnestMap}_{\$2=child(\$1,a)}([\$1])))$$

However, one has to be careful: not all XPath expressions can be translated straightforwardly into a sequence of `UnnestMap` operations. Often relatively complex transformations are needed to guarantee a correct and efficient pipelined evaluation of XPath expressions. Consider for example the document in Fig. 8 which contains only `A` elements which are numbered for convenience. When evaluating the XPath expression `/descendant-or-self::A/descendant-or-self::A` in a straightforward manner by evaluating each axis on every input node separately, the result will contain duplicates. Since XPath does not allow duplicates, duplicate elimination has to be performed. However, things get even worse. During naive evaluation `A2` and all its descendants are generated three times. Hence, we not only have to perform a single duplicate elimination after naively evaluating the path expression but also do a lot of redundant work. We can attenuate the problem by performing a duplicate elimination after every step that may produce duplicates. An even better approach is to build a plan that never produces duplicates. Details on such a method can be found in [16,17].

Result construction. For XML result construction NPA provides the `BA-Map`, `FL-Map`, `Groupify-GroupApply`, and `NGroupify-NGroupApply` operators. The interfaces of these operators are shown in Fig. 9. The `BA-Map` and `FL-Map` operators are simple enhancements of the traditional Map operator. They take three NVM programs as parameters. The program called `each` is called on every input tuple. The programs `before` and `after` of the `BA-Map` operator are called before the first and after the last tuple, respectively. The programs `first` and `last` of the `FL-Map` operator are called on the first and last tuple, respectively. In general `BA-Map` is more efficient (`FL-Map` needs to buffer the current tuple) and should be used whenever applicable.

The `Groupify` and `GroupApply` pair of operators detects group boundaries and executes a subplan contained between them for every group. The `Groupify` operator has a set of attributes as parameters. These attributes are used to detect groups of tuples. On every first tuple of a group the program `first` is

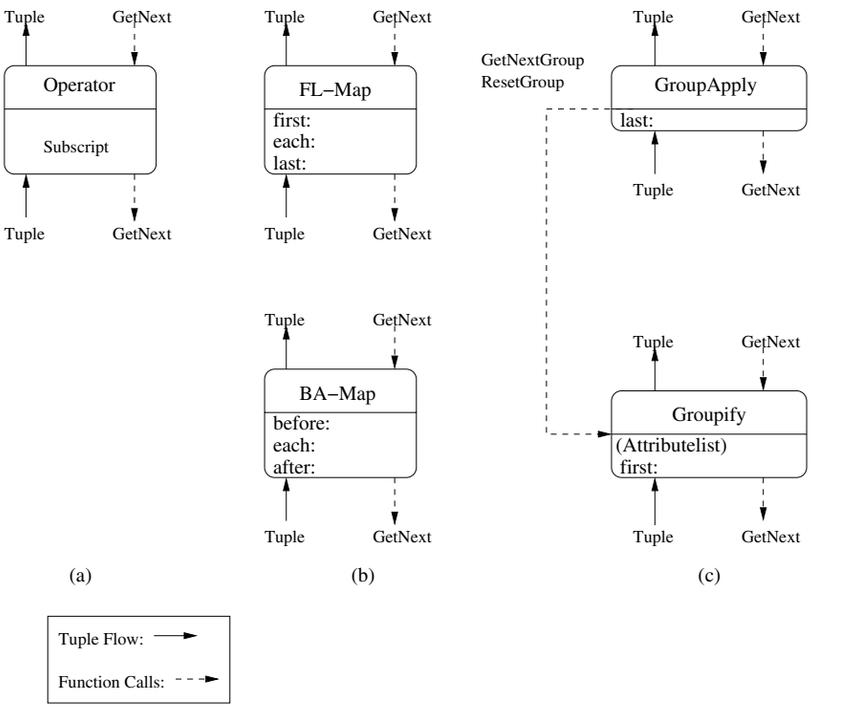


Fig. 9. Interfaces of construction operators

executed. Whenever one attribute's value changes, it signals an end of stream by returning `false` on the `next` call. The `GroupApply` operator then applies the `last` program on the last tuple of the group. It then asks the `Groupify` operator to return the tuples of the next group by calling `GetNextGroup`. `ResetGroup` allows to reread a group. The `NGroupify` and `NGroupApply` pair of operators allows multiple subplans to occur between them. More details about the operators and the generation and optimization of construction plans can be found in [10, 11].

6 Conclusion

Exemplified by storage management, recovery, multi-user synchronization, and query processing, we illustrated that the challenges of adapting database management systems to handling XML are not limited to schema design for relational database management systems.

We believe that sooner or later a paradigm shift in the way XML documents are processed will take place. As the usage of XML and its storage in DBMSs spreads further, applications working on huge XML document collections will be the rule. These applications will reach the limits of XML-enhanced traditional

DBMSs with regard to performance and application development effectiveness. Our contribution is to prepare for the shift in processing XML documents by describing how efficient, native XML base management systems can actually be built.

Acknowledgments. The authors thank Simone Seeger for her help in preparing the manuscript. Soeren Goeckel, Andreas Gruenhagen, Alexander Hollmann, Oliver Moers, Thomas Neumann, Frank Ueltzhoeffer, and Norman May provided invaluable assistance in implementing the system.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
2. P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley, Reading, Massachusetts, USA, 1987.
3. Alexandros Biliris. An efficient database storage structure for large dynamic objects. In *Proceedings of the International Conference on Data Engineering*, pages 301–308, 1992.
4. S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML query language. Technical report, World Wide Web Consortium, 2002. W3C Working Draft 30 April 2002.
5. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible markup language (xml) 1.0 (second edition). Technical report, World Wide Web Consortium (W3C), 2000.
6. Michael J. Carey, David J. DeWitt, Joel E. Richardson, and Eugene J. Shekita. Object and file management in the EXODUS extensible database system. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 91–100, Los Altos, California, USA, 1986.
7. James Clark and Steve DeRose. XML path language (XPath) version 1.0. Technical report, World Wide Web Consortium (W3C) Recommendation, 1999.
8. data ex machina. NatixFS technology demonstration, 2001. available at `http://www.data-ex-machina.de/download.html`.
9. T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a Native XML Base Management System. *VLDB Journal*, to appear.
10. T. Fiebig and G. Moerkotte. Algebraic XML construction and its optimization in Natix. *WWW Journal*, 4(3):167–187, 2001.
11. T. Fiebig and G. Moerkotte. Algebraic XML construction in Natix. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'01)*, pages 212–221, Kyoto, Japan, December 2001. IEEE Computer Society.
12. T. Fiebig and G. Moerkotte. Evaluating queries on structure with extended access support relations. In *The World Wide Web and Databases, Third International Workshop WebDB 2000, Dallas, Texas, USA, May 18–19, 2000, Selected Papers*, volume 1997 of *Lecture Notes in Computer Science*. Springer, 2001.
13. Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. Http extensions for distributed authoring – webdav. Technical Report RFC2518, Internet Engineering Task Force, February 1999.

14. Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.
15. Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, USA, 2000.
16. S. Helmer, C.-C. Kanne, and G. Moerkotte. Optimized translation of xpath expressions into algebraic expressions parameterized by programs containing navigational primitives. In *Proc. Int. Conf. on Web Information Systems Engineering (WISE)*, 2002. to appear.
17. S. Helmer, C.-C. Kanne, and G. Moerkotte. Optimized translation of XPath expressions into algebraic expressions parameterized by programs containing navigational primitives. Technical Report 11, University of Mannheim, 2002.
18. Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (DOM) level 2 core specification. Technical report, World Wide Web Consortium (W3C), 2000.
19. Tobin J. Lehman and Bruce G. Lindsay. The Starburst long field manager. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 375–383, Amsterdam, The Netherlands, August 1989.
20. Mark L. McAuliffe, Michael J. Carey, and Marvin H. Solomon. Towards effective and efficient free space management. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 389–400, Montreal, Canada, June 1996.
21. David Megginson. SAX: A simple API for XML. Technical report, Megginson Technologies, 2001.
22. Julia Mildenberger. A generic approach for document indexing: Design, implementation, and evaluation. Master's thesis, University of Mannheim, Mannheim, Germany, November 2001. (in German).
23. C. Mohan and D. Haderle. Algorithms for flexible space management in transaction systems supporting fine-granularity locking. *Lecture Notes in Computer Science*, 779:131–144, 1994.
24. C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, March 1992.
25. C. Mohan and Hamid Pirahesh. Aries-rrh: Restricted repeating of history in the aries transaction recovery method. In *Proceedings of the Seventh International Conference on Data Engineering, April 8–12, 1991, Kobe, Japan*, pages 718–727. IEEE Computer Society, 1991.
26. Robert Schiele. NatiXync: Synchronisation for XML database systems. Master's thesis, University of Mannheim, Mannheim, Germany, September 2001. (in German).
27. Gerhard Weikum and Gottfried Vossen. *Transactional information systems : theory, algorithms and the practice of concurrency control and recovery*. Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, USA, 2002.
28. Gio Wiederhold. *File organization for database design*. McGraw-Hill computer science series; McGraw-Hill series in computer organization and architecture; McGraw-Hill series in supercomputing and artificial intelligence; McGraw-Hill series in artificial intelligence. McGraw-Hill, New York, NY, USA, 1987.
29. I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, San Francisco, CA 94104-3205, USA, 1999.

Intelligent Support for Selection of COTS Products

Günther Ruhe

University of Calgary
ruhe@ucalgary.ca
http://sern.ucalgary.ca/~ruhe/

Abstract. Intelligent Decision Support is considered in unstructured decision situations characterized by one or more of the following factors: complexity, uncertainty, multiple groups with a stake in the decision outcome (multiple stakeholders), a large amount of information (especially company data), and/or rapid change in information. Support here means to provide access to information that would otherwise be unavailable or difficult to obtain; to facilitate generation and evaluation of solution alternatives, and to prioritize alternatives by using explicit models that provide structure for particular decisions.

Integration of commercial off the shelf (COTS) products as elements of larger systems is a promising new paradigm. In this paper, we focus on the selection of COTS products. This is characterized as a problem with a high degree of inherent uncertainty, incompleteness of information, dynamic changes and involvement of conflicting stakeholder interests. A semi-formal problem description is given. We derive requirements on Decision Support Systems for COTS selection and discuss ten existing approaches from the perspective of those requirements. As a result, we propose an integrated framework called COTS-DSS combining techniques and tools from knowledge management and artificial intelligence, simulation and decision analysis.

Keywords: Components, Reuse, COTS selection, requirements, decision support, integrated framework.

1 Introduction

While software is of paramount importance for market success in all high-tech and service domains, Software Engineering practice does not yet live up to this challenge and requires tremendous improvement efforts. There is a great variety of techniques, methods, and tools. But not so much is known about their appropriate use under specific goals and constraints. Even worse, decisions to be made to achieve predefined goals are mostly done ad hoc, without reference to reliable models, best knowledge and sound methodology. Intelligent support is needed to make the best decision according to the characteristics of the environment.

Software Engineering Decision Support as a new paradigm for learning software organizations is exactly addressing this issue [18]. Computerized decision support

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 34–45, 2003.

should be considered in unstructured decision situations characterized by one or more of the following factors: complexity, uncertainty, multiple groups with a stake in the decision outcome (multiple stakeholders), a large amount of information (especially company data), and/or rapid change in information. Support here means to provide access to information that would otherwise be unavailable or difficult to obtain; to facilitate generation and evaluation of solution alternatives, and to prioritize alternatives by using explicit models that provide structure for particular decisions. Software Engineering decision support systems (SE-DSS) couple the intellectual resources of individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of proposed decisions. Some systems are primarily tools for individuals working more or less alone on decision tasks; others serve primarily to support communication among people and stakeholders.

The need for decision support covers the complete software life cycle. For the requirements, analysis, design, construction, testing and evolution phases, decision makers need support to describe, evaluate, sort, rank, select or reject candidate products, processes or tools. In this paper, focus will be on COTS based software development, especially on the selection of COTS products. However, many of the questions discussed here are (slightly modified) applicable also for other type of decision problems.

Integration of commercial off the shelf (COTS) products as elements of larger systems is a promising new paradigm. COTS-based development allows organizations to focus on their key business area, and to rely on outside sources for other functions. Another fundamental motivation for use of COTS software is its expected impact on timely maintenance as well as on flexibility and on ease of modernization of computer-based systems. From a system perspective, COTS products are black boxes providing operational functionality. However, not so much is known about the interior of the COTS product and how does is fit with original requirements and with the remaining software system.

Systems that are built from existing COTS components are potentially faced with incompleteness and uncertainty of information as well as with a large number of criteria and constraints. Evaluation and selection of most appropriate COTS products is a decision problem of tremendous impact on the subsequent processes and products of software development and evolution. COTS selection has to be done in close interaction with requirements analysis and system design and strongly taking into account the impact on subsequent integration, validation and maintenance activities.

The paper is subdivided into five parts. Following this introduction is a description of Software Engineering Decision Support Systems (SE-DSS) and its relationship to the Experience Factory and Learning Software Organization approaches. A semi-formal problem statement of COTS selection and a characterization of the problem in terms of its inherent complexity are presented in part 3. A brief summary of existing approaches for COTS selection and an evaluation from the perspective of Software Engineering Decision Support is given in part 4. Finally, the architecture of an

integrated framework for intelligent decision support in COTS selection called COTS-DSS is discussed in chapter 5.

2 Software Engineering Decision Support Systems

The Experience Factory (EF) [4] supports organizational learning in the areas of software development, evolution, and application. Objects of learning are all kinds of models, knowledge and lessons learned related to the different processes, products, tools, techniques, and methods applied during the different stages of the software development process.

Learning extends knowledge and enables decision making for individuals as well as for groups and entire organizations. Learning Software Organization (LSO) extends the EF approach so as to accelerate the learning processes supported by the LSO, to extend the scope of knowledge management to all directly or non-directly relevant processes, products, methods, techniques, tools, and behavior in the context of software development, and to extend the knowledge management approach to handle the tacit knowledge available within an organization [19].

In the context of LSO knowledge management and learning approaches are complementary views on knowledge handling processes. The knowledge management literature usually deals with the mechanisms of knowledge handling, while learning approaches address the process how to gain knowledge. This can be done on an individual, group, or organizational level.

Decision Support Systems (DSS) are defined as a broad category of analytical management information systems [21]. They help decision makers use communications technologies, data, documents, knowledge and/or models to identify and solve problems and make decisions. DSS are based on a suite of analytical and modeling tools, on simulation capabilities as well as on intelligence-based capabilities for reasoning, retrieval and navigation. Nowadays, DSS are typically based on internet technologies. They have graphical user-interfaces and a high degree of interactions with a distributed group of staff.

There are a great variety of DSS types and they serve different specific functions. The general function of all DSS is to support decision-making tasks, activities and processes. Some specific DSS focus more on information gathering, analysis and performance monitoring, other DSS help conduct "What-if?" analyses and support comparison of specific decision alternatives. Some DSS provide expertise and knowledge to augment and supplement the abilities and skills of the decision makers. Other DSS encourage and support communication, collaboration and/or group decision-making.

Software Engineering Decision Support Systems can be seen as an extension and continuation of the Software Engineering Experience Factory and Learning Software Engineering approaches [18]. In addition to collecting, retrieving and maintaining models, knowledge, and experience in the form of lessons learned, SE-DSS generates new insights from on-line investigations in a virtual (model-based) world, from

offering facilities to better structure the problem as well in ranking and selecting alternatives. For that purpose, sound modeling and knowledge management is combined with a variety of techniques of analysis, reasoning, simulation, and decision-making.

3 COTS Product Selection

3.1 Problem Statement: COTS-Select

To determine requirements on Decision Support for COTS product selection, we give a semi-formalized description of the problem called COTS-Select.

Given:

- A (imprecise and incomplete) description of the functional and non-functional requirements of a product to be developed;
- A set of stakeholders with different opinions and preferences;
- An underlying (dynamically changing) process of (COTS-based) software development;
- Knowledge and experience on (COTS-based) software development;
- A set Ω of candidate COTS products (pre-selected for the domain and the functionality of the overall system to be developed).

To be determined: A subset Ω^* of COTS products that contributes most to

- 'optimal' functionality of the final product,
- better quality (non-functional requirements) of the final software system,
- more efficient development of this product (saving of effort), and
- shorter time-to-market (saving of time) of the final product.

3.2 Problem Characterization

COTS-Selection is characterized by uncertainty, dynamic changes of the environment, explicit and implicit criteria and constraints and involvement of different stakeholders.

- **Uncertainty**: Uncertainty about requirements and uncertainty about the impact of COTS products on reliability, performance, (maintenance) effort and time of the overall product life-cycle.
- **Dynamic changes:** The problem is dynamic because of changing variables over time. This concerns both the candidate COTS with their characteristics (products are evolving) as well as the requirements of the system to be developed (evolutionary understanding).
- **Problem complexity:** Potentially, the number of candidate COTS is large. However, assuming a kind of pre-screening (focusing on the domain and

functionality), the problem becomes of small to medium size in the number of candidate COTS products.

- **Objectives:** Selection of COTS includes a variety of (non-comparable) objectives such as: cost-benefit ratio in effort and time, covered functionality, implications on maintenance, availability of service, reputation and credibility of the provider.
- **Constraints:** There are constraints on fitness with existing architecture and non-functional constraints such as reliability, performance or cost. Unfortunately, from the black-box character of COTS it is hard to evaluate them in this respect, i.e., constraints are more qualitative and uncertain in nature.
- **Stakeholder:** There might be conflicting (stakeholder) interests in terms of the functionality and the provider of the COTS products.

4 Decision Support for COTS Selection

4.1 Requirements on Decision Support for COTS-Selection

Taking the above problem statement and problem characteristics, we can derive some set of "idealized" requirements on support systems that combine the intellectual resources of individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of COTS selection. In dependence of the usage scenario of the COTS-Selection DSS (on-line versus off-line support, individual versus group-based decision support), different aspects will become more important than others.

- (R1) **Knowledge, model and experience management** for the existing body of knowledge of COTS-based software development (in the respective organization).
- (R2) **Integration** into existing organizational information systems (e.g., ERP systems).
- (R3) **Process orientation** of decision support, i.e., consider the process how decisions are made, and how they impact development and business processes.
- (R4) **Process modeling and simulation component** to plan, describe, monitor, control and simulate ("what-if" analysis) the underlying COTS-based development processes and to track changes in its parameters and dependencies.
- (R5) **Negotiation component** to incrementally select COTS products having a 'best' overall fit with given functional and non-functional requirements, architecture and to resolve conflicts between stakeholders.
- (R6) **Presentation and explanation component** to present and explain generated knowledge and solution alternatives in various customized ways to increase transparency.

- (R7) **Analysis and decision component** consisting of a portfolio of methods and techniques to evaluate and prioritize generated solution alternatives and to find trade-offs between the conflicting objectives and stakeholder interests.
- (R8) **Intelligence component** for intelligent knowledge retrieval, knowledge discovery and approximate reasoning.
- (R9) **Group facilities** to support electronic communication, scheduling, document sharing, and to allow access to expert opinions.
- (R10) (Web-based) **Graphical User Interface** to facilitate interaction between (remote) users and the different components of the DSS in such a way that the user has a flexible choice and sequence of the decision supporting activities.

4.2 Existing Approaches

There are an increasing number of approaches addressing COTS-Selection. Most of the existing approaches reduce problem complexity by neglecting some of the inherent difficulties of the problem. We will briefly discuss the main contributions of ten methods.

4.2.1 OTSO

OTSO (Off-the-Shelf-Option) [11],[12] method starts from the definition of hierarchical evaluation criteria that takes several influencing decision factors into account. The factors include requirement specification, organizational infrastructure, application architecture, project objectives and availability of libraries. All alternatives are compared with respect to the cost and value they provide and AHP [20] is used to consolidate the evaluation results for decision-making. OTSO assumes that requirements do already exist and that they are fixed.

4.2.2 PORE

PORE (Procurement-Oriented Requirements Engineering) method [14] is a template-based approach based on an iterative process of requirements acquisition and product evaluation. It includes knowledge engineering techniques, feature analysis technique to aid when scoring the compliance of COTS to requirements, and multi-criteria decision making techniques to aid decision making during the complex product ranking. PORE proposes a model of product-requirement compliance to provide a theoretical basis for technique selection. The COTS selection is made by rejection.

4.2.3 CEP

CEP (Comparative Evaluation Process) [17] is an instance of the Decision Analysis and Resolution (DAR) process area the Capability Maturity Model Integration^{SM} (CMMI^{SM}) and is based on a structured decision-making process. CEP is made up of five top-level activities: (i) scoping evaluation effort, (ii) search and screening candidate components, (iii) definition of evaluation criteria, (iv) evaluation of candidate components and (v) analysis of evaluation results. The categories of evaluation criteria include basic, management, architecture, strategic, and functional goals. One feature of this method is the credibility scoring of data source. The CEP decision model use weighted averages to calculate evaluation results based on the criteria value and credibility ratings.

4.2.4 CAP

CAP (COTS Acquisition Process) [16] is a repeatable and systematic method for performing COTS selection. It is strictly measurement-oriented, allowing for optimization of the evaluation towards cost-efficiency, systematic changes of evaluation depth and spiral model-like enactment. Its three main components are CAP Initialization Component, CAP Execution Component, and cap reuse Component. The main feature of this method is the estimation of measurement effort. One crucial assumption for applying CAP is the existence of full system requirement specification prior to the decision to search for COTS software.

4.2.5 CRE

CRE (COTS-Based Requirements Engineering) [2] is an iterative process to guide the evaluation and selection of COTS. The selection of COTS is made by rejection. The goal-oriented approach has four iterative phases: Identification, Description, Evaluation, and Acceptance. During selection phase, four dimensions including domain coverage, time restriction, cost rating and vendor guaranties are considered. A key issue supported by this method is the definition and analysis of non-functional requirements.

4.2.6 QESTA

QUESTA [10] is mainly concerned with the actual process of evaluation, excluding such phases as developing a plan and choosing the alternative entities to be evaluated. The heart of the evaluation process consists of five steps: (i) Qualification of qualities into metrics, (ii) Examination of the entities to find their values for those metrics, (iii) Specification of transforms based on the metrics, (iv) Transformation of those values according to the transforms to produce factors, and (v) Aggregation of the factors to generate decision data. It is based on the quantitative evaluation of each candidate products. This method considers the requirement from different stakeholders perspective. Stakeholders and expert together specify the value of each metric.

4.2.7 Storyboard

Storyboard [9] takes advantage of use case modeling and screen captures to combine traditional requirements products into a single product that provide a clear, unambiguous understanding of the system to be developed. The goal of the storyboard process is to align user interface and user interaction requirements to the capabilities inherent in the COTS products, thus minimizing customization and integration code. Storyboards provide a means for all to communicate requirements more clearly and concisely. It also provides an excellent means for managing user expectations and overcoming the myth that COTS integration is plug-and-play. Non-functional requirements are not addressed in the selection process. There isn't a formal COTS evaluation process in this methodology.

4.2.8 Combined Selection of COTS Components

Combined selection of COTS components as described in [6] is a process model that consists of two levels. The global level is responsible of negotiating the overall process of combined selection, by firing individual selection processes for each area, supervising their evolution, controlling the viability of their results, and finding the best combination in the resulting proposals. The local level selection could follow some of the currently existing analysis and decision-making methods. The approach

considers the incomplete requirements by using cycles between phases and interactions between global level and local level. It is mainly intended to organizations that are highly specific, oriented to satisfy the product or service particularities within their vertical industries and markets.

4.2.9 PECA

PECA [8] consists of four phases: (i) Planning the evaluation, (ii) Establishing the criteria, (iii) Collecting the data, and (iv) Analyzing the data. This is a high level process that has to be tailored for each specific project. It is also a good guideline for any COTS evaluation process as this process elaborates all aspects that influence evaluation results and all activities in COTS evaluation. The decision itself is not considered as part of the evaluation process – the aim of the process is to provide all the information necessary for a decision to be made.

4.2.10 STACE

STACE (Socio-Technical Approach to COTS Evaluation) [13] considers the whole COTS evaluation and selection process and emphasizes customer participation during evaluation. Specifically, this approach studies how to define evaluation criteria. There are three underlying principles of STACE: (i) Support for a systematic approach to COTS evaluation and selection, (ii) Support for both evaluation of COTS products and the underlying technology, and (iii) Use of socio-technical techniques to improve the COST selection process. This method supports the negotiation between requirement elicitation and COTS evaluation.

4.3 Evaluation

There is a great variety of methods and techniques to sole COTS-Select. None of the ten approaches discussed in part 4.2 was primarily designed to become a global and integrated DSS for COTS-Selection. Not surprisingly, none of the approaches fulfills all or even most of the formulated requirements. However, most of the requirements are addressed by at least one of the approaches. In general, there is a focus on applying analysis and decision techniques such as multi-attributive utility theory, multi-criteria decision aid, weighted score method, and the analytic hierarchy process (AHP). The limitations of the most of these techniques are discussed in [15].

Some of the methods (e.g., PORE, CEP, STACE) are process oriented and offer (intelligent) support for negotiation to determine a ‘good’ fit between evolving requirements, candidate COTS, and architectures. Most of the methods have a ‘closed world’ assumption. That means, support is not provided for interaction between different stakeholders or for interaction with the computer to conduct simulation-based scenarios to estimate the impact of certain decisions. Group facilities to support electronic communication, scheduling, document sharing and to allow access to expert opinions is not considered by any of the approaches discussed. Synchronization with existing business processes and integration into existing organizational information systems is only addressed by the Comparative Evaluation Process CEP and the ’Combined Selection of COTS’-method. User interface and user interaction requirements are the special focus of the storyboard technique. CEP is the only method explicitly taking into account non-functional requirements.

5 COTS-DSS – An Integrated Framework for Decision Support in COTS Selection

From the problem characterization presented in part 3.2 it became clear that COTS-Selection is a highly complex problem with a huge amount of uncertainties. Its solution needs more than just the local application of analysis and decision-making techniques. Instead, we propose a global and integrated, knowledge-based approach with build-in analysis and negotiation components. It encompasses intelligent knowledge retrieval and knowledge discovery facilities as well as components for modeling and simulation.

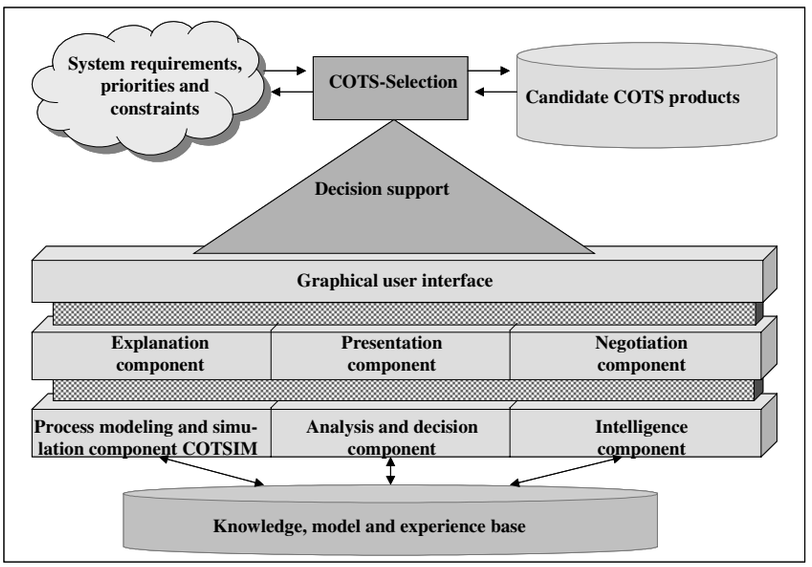


Fig. 1. Architecture for Decision Support in COTS Selection

The conceptual architecture of this framework is described in Figure 1. The main difference to the approaches discussed in chapter 4 is that we consider COTS-Selection as part of the complete decision-making, business and software development processes. We try to provide and to retrieve as much knowledge and information from these processes as possible. However, it needs further implementation and evaluation to really demonstrate the potential advantages of this integrated framework called COTS-DSS.

COTS-DSS comprises three connected layers with different components and link to a common knowledge, model, and experience base. The components are designed to communicate with each other on the same level and between levels. Whenever

applicable, we rely on existing solutions for their implementation. Explanation capabilities are mainly intended to justify the proposed solution alternatives. This increases transparency of decision-making. In conjunction with presentation capabilities, it supports the negotiation process between different stakeholder interests.

For the underlying knowledge, model, and experience base we initially use the lessons learned repository [4] developed and offered for public domain by CeBASE project [7]. In its current stage, the repository is seeded with an initial set of about 70 lessons learned extracted from the literature. The interface to the repository supports search and retrieval based on text search over all attributes. Links to the source of each lesson are also provided.

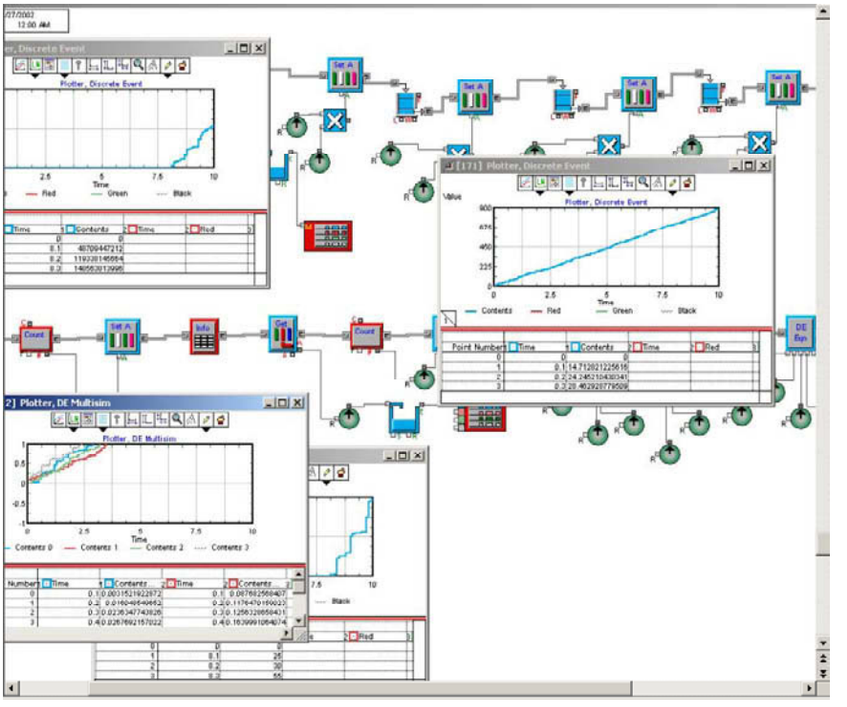


Fig. 2. Part of the COTSIM interface with underlying process model and resulting effort functions.

Another essential component of COTS-DSS is devoted to modeling and simulation. COTSIM [22] is a simulation-based approach to estimate project effort for software development using COTS products. Although inherently difficult, estimation of the impact on effort of reusing COTS is studied using the COCOTS [1] process model in combination with discrete event simulation. For that purpose, the commercial tool Extend™ was implemented.

COTSIM contributes to COTS-DSS by providing effort estimates for the different scenarios. Following COCOTS, COTSIM includes sub-models to estimate filtering, assessment, tailoring and glue code effort. As a kind of snapshoot, Figure 2 shows part of the model and resulting effort curves. COTSIM improves COCOTS estimation model in terms of its inherent flexibility to conduct simulation runs with varying problem parameters. Coping with uncertainty, some stochastic parameters are introduced into the model. The output of a series of simulation runs is a distribution instead of one single value.

Acknowledgement. The author would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE) for their financial support of this research. Many thanks also to Qun Zhou and Jinfang Sheng for evaluation of current selection techniques.

References

[1] C. M. Abts, B. Boehm: COCOTS (Constructive COTS) Software Integration Cost Model: An Overview. USC Center for Software Engineering, University of Southern California. August 1998.

[2] C. Alves, J. Castro: CRE: A Systematic Method for COTS Components Selection. XV Brazilian Symposium on Software Engineering (SBES) Rio de Janeiro, Brazil, October 2001.

[3] V.R. Basili, B. Boehm: COTS-Based Systems Top 10 List, IEEE Software May 2001, pp. 91–93.

[4] V.R Basili, G. Caldiera, D. Rombach: Experience Factory. In: J. Marciniak: Encyclopedia of Software Engineering, Volume 1, 2001, pp. 511–519.

[5] V.R. Basili, M. Lindvall, I. Rus, I., C. Seaman, and B. Boehm: Lessons-Learned Repository for COTS-Based SW Development, *Software Technology Newsletter*, vol. 5, no. 3, September 2002.

[6] X. Burgués, C. Estay, X. Franch, J.A. Pastor, and C. Quer: Combined selection of COTS components, Proceedings of ICCBSS, February, Orlando, Florida USA, 2002, pp. 54–64.

[7] CeBASE. NSF Center for Empirically Base Software Engineering, see http://www.cebase.org/www/home/index.htm

[8] S. Comella-Dorda, J. C. Dean, E. Morris, and P. Oberndorf, A process for COTS Software Product Evaluation, Proceedings of ICCBSS, February 2002, Orlando, Florida USA, pp. 86–92.

[9] S. Gregor, J. Hutson, and C. Oresky: Storyboard Process to Assist in Requirements Verification and Adaptation to Capabilities Inherent in COTS, Proceedings of ICCBSS, February 2002, Orlando, Florida USA, 2002, pp. 132–141.

[10] W. J. Hansen: A Generic Process and Terminology. URL: http:// www.sei.cmu.edu/cbs/tools99/generic/generic.html

[11] J. Kontio, OTSO: A Systematic Process for Reusable Software Component Selection CS-TR-3478, 1995. University of Maryland Technical Reports.

[12] J. Kontio: A Case Study in Applying a Systematic Method for COTS Selection. In: Proceedings of the 18th International Conference on Software Engineering ICSE'1996, Berlin, pp. 201–209.

[13] D. Kunda and L. Brooks: Applying Social-Technique approach for COTS selection, Proceedings of 4th UKAIS Conference, University of York, McGraw Hill, April 1999.
[14] N. Maiden, C. Ncube: Acquiring COTS Software Selection Requirements. IEEE Software, March/April 1998, pp. 46–56.
[15] C. Ncube & J.C. Dean: The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Products. Proceedings of ICCBSS, February 2002, Orlando, Florida USA, 2002, pp. 176–187.
[16] M. Ochs, D. Pfahl, G. Chrobok-Diening, B. Nothhelfer-Kolb: A Method for Efficient Measurement-based COTS Assessment and Selection–Method Description and Evaluation Results. Software Metrics Symposium, pp. 285–296, 2001
[17] B.C. Phillips and S. M. Polen: Add Decision Analysis to Your COTS Selection Process. Software Technology Support Center Crosstalk, April 2002.
[18] G. Ruhe: Software Engineering Decision Support – A New Paradigm for Learning Software Organizations, appears in: Proceedings of the 4th Workshop on Learning Software Organizations, Chicago, Springer 2003.
[19] G. Ruhe and F. Bomarius (eds.): Learning Software Organization – Methodology and Applications, Lecture Notes in Computer Science, Volume 1756, Springer 2000.
[20] T.L. Saaty, The Analytic Hierarchy Process, Wiley, New York 1980.
[21] E. Turban, J.E. Aronson: Decision Support Systems and Intelligent Systems, Prentice Hall, 2001.
[22] Q. Zhou: COTSIM – An effort estimation method for COTS-based software development using discrete event simulation. Manuscript. University of Calgary, Department of Electrical and Computer Engineering, 2002.

DAML Enabled Web Services and Agents in the Semantic Web

M. Montebello and C. Abela

CSAI Department
University of Malta
Malta
mmont@cs.um.edu.mt
abcharl@maltanet.net

Abstract. Academic and industrial bodies are considering the issue of Web Services as being the next step forward. A number of efforts have been made and are evolving to define specifications and architectures for the spreading of this new breed of web applications. One such work revolves around the Semantic Web. Lead researches are trying to combine the semantic advantages that a Semantic Web can provide to Web Services. The research started with the now standardized RDF (Resource Description Framework) and continued with the creation of DAML+OIL (DARPA Agent Markup Language and Ontology Inference Layer) and its branches, particularly DAML-S (where S stands for Services) [1].

The Semantic Web's point of view, being considered in this paper presents a rich environment where the advantages of incorporating semantics in searching for Web Services can be fully expressed. This paper aims to describe an environment called DASD (**D**AML **A**gents for **S**ervice **D**iscovery) where Web Service requesters and providers can discover each other with the intermediary action of a Matchmaking service.

Keywords: Semantic Web, Web Services, Ontologies, Matchmaking

1 Introduction

As the demand for Web Services is increasing, this is producing a situation whereby entities searching for a particular Web Service are faced with the burden of deciding which one is the best to use.

DASD is intended to produce an environment where technologies revolving around the Semantic Web initiative are integrated together to present a solution that will help interested parties in making such a decision. For this purpose the application we created is divided into two main parts, the Service Requester/Provider agent and the Matchmaker. Though the Requester and Provider have different roles, their implementation is similar, and for this reason the same packages are used in their

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 46–58, 2003.

implementation. As their name suggests, the Requester is the application, which the user can utilize to request a service, while the Provider is the application, which is used to advertise a service. On the other hand the Matchmaker is the application, which helps in making all this happen, since it helps in the discovery of Service Providers by Service Requesters.

The aims of the project can be summarised as follows:

- The agents involved have to use a common ACL, Agents Communication Language.
- The agents have to make use of the ACL to:
    - Create requests
    - Create replies
    - Reason on requests and replies when necessary
- The Requester agent should be able to request and invoke a Web Service.
- The Provider Agent should be able to submit advertisements of Web Services.
- The Matchmaker Agent should be able to store advertisements and match requests with advertisements.

To accomplish the above-mentioned aims we had to devise several hybrid solution that involved the integration of different languages and technologies. As the underlying technology for the ACL we used DAML+OIL (DARPA Agent Markup Language) [2] which is a language based on the RDF (Resource Description Framework) [3] standard. We also combined RDF with WSDL (Web Services Description Language) [4] to create the grounding ontologies that are needed by the Requester to automatically invoke a Web Service. As regards the reasoning capabilities of the agents, we integrated an FOL (First Order Logic) reasoner, called JTP (Java Theorem Prover) [5], into the agent's control module to make it possible for the agent to extract and manipulate facts originating from the various ontologies. This reasoner also plays a very important part in the matching process of the MatchMaker. The whole process of requesting and invoking Web Services is made transparent to the user. To achieve this, several mapping classes are used to transform the inputs presented by the user into the format that the agents can understand and to which they can react.

We first motivate our efforts by giving some background information on the areas of research that we mention in our work. In the central part of the paper we describe the system that we developed by focussing on the main components. We then present some evaluation information by describing some tests that we carried out using the system. We finally end the paper with some concluding statements.

2 Background

According to Tim Berners-Lee, Director of the World Wide Web Consortium (W3C) [6], the Semantic Web is a web of data, which in some way resembles a global database. In his paper *Semantic Web Road Map* [7] Tim Berners-Lee states that a goal of the Semantic Web is that it should be useful not only for human-human

communication, but also that machines would be able to process and interpret its contents. A major obstacle to this view is the fact that most information on the Web is designed for human consumption and the structure of the data is not evident to a robot browsing the web. In the paper *The Semantic Web* [8] the authors mention four basic components that are necessary for the evolution of the Semantic Web;

- ***Expressing meaning***: The Semantic Web is aimed at bringing structure and adding semantics to the content of web pages, hence creating an environment where software agents can roam from page to page, carrying out sophisticated tasks for the users.
- ***Access to knowledge representations***: The Semantic Web is aimed at resolving the limitations of traditional Knowledge Representations by creating a rules language that is expressive enough to allow the Web to reason as widely as desired.
- ***Ontologies***: Ontologies in the Semantic Web have a fundamental role, since they provide machines with the possibility to manipulate terms more effectively. With Web content referring to ontologies, various terminology problems can be solved.
- ***Agents***: The real power of the Semantic Web is realized when agents that are capable of handling semantic content are used to collect and process Web information and exchange the results with other agents. Issues like exchange of proofs and digital signatures will ensure that the results exchanged between agents are valid and can be trusted.

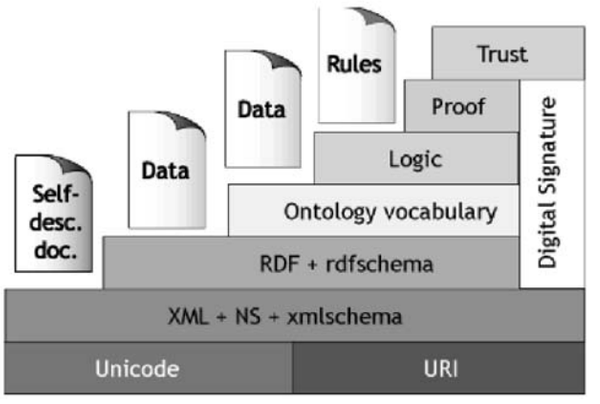


Fig. 1. Architecture for the Semantic Web

The architecture for the evolution of the Semantic Web as depicted by Tim-Berners Lee is shown in Figure 1 where the upper levels depend on the levels beneath. XML is at the base of this architecture, but it can provide only structure to documents. More expressive power can be obtained with RDF and RDFS. These though are not expressive enough and languages such as DAML+OIL and the newly proposed OWL are aimed at achieving such expressivity. Ontologies play an important role in this architecture and their coupling with semantic languages will enable software agents to understand and manipulate web content. There is still a lot to be done though, especially for the realisation of the upper three levels since only minimal work has been done on these levels.

The increased expressivity through the use of semantics is not only an advantage for agents that are roaming the Web, but can also be utilised with similar benefits in other web-related areas such as in the area of Web Services. Several research activities are infact being conducted in this respect. One of these research groups has drafted a set of ontologies that can be used in the process of discovery, invocation and possibly execution of Web Services.

Academic and industrial bodies consider Web Services as being at the heart of the next generation of distributed systems. The W3C is also involved in this research and has a dedicated site at [9]. A Web Service can be defined as a collection of functions that are packaged as a single entity and published to the network for use by other programs. Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide. A Web service can aggregate with other Web services to provide a higher-level set of features.

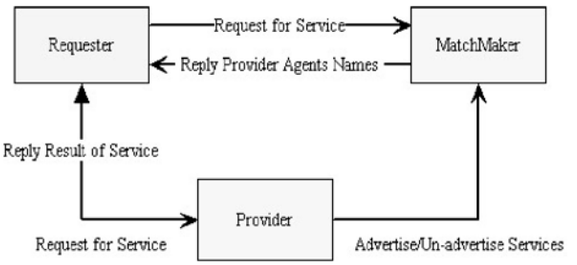


Fig. 2. Matchmaking Architecture

The main actors involved in the advertisement, discovery, invocation and execution processes of Web Services are the ***Service Provider***, which advertises a particular Web Service, the ***Service Requester***, which requests and invokes a Web Service and the ***Matchmaker*** or ***Broker*** whose role as an intermediary is important in the discovery process between Provider and Requester [10].

From the industrial aspect, Microsoft, IBM and Ariba defined a three level architecture for web services. This includes UDDI (Universal Discovery Description Integration) [11]. WSDL (Web Services Description Language) [12] and SOAP (Simple Object Access Protocol) [13]. On the other front, research on the Semantic Web has lead researches to combine the semantic advantages that the Semantic Web offers with Web services. DAML-S is one such result, which leverages on DAML+OIL ontologies for the description of Web Services.

A limitation that surrounds XML based standards, such as UDDI and WSDL is their lack of explicit semantics by which two identical XML descriptions could mean totally different things, depending on the context in which they are used. This limits the capability of matching Web services. This is important because a requester for a Web service does not know which services are available at a certain point in time and so semantic knowledge would help in the identification of the most suitable service for a particular task.

Matchmaking is the process of pruning the space of possible matches among compatible offers and requests. A matchmaking service requires rich and flexible metadata as well as matching algorithms. UDDI is a registry of Web Services, which a Service Requester can query to find a required Web Service. But since it does not in itself support semantic descriptions of services, UDDI depends on the functionality offered by a content language such as WSDL. DAML-S on the other hand defines three ontologies for service description, invocation and execution [14]. The ***Service Profile*** ontology is used to describe and advertise the Web Service, the ***Service Model*** is used to describe the process model and execution of the service and the ***Service Grounding*** is used to describe how to access the service. The Service Profile ontology is the most complete of the three, while the Service Model has some important issues still unresolved, such as the mapping of input/output concepts to the actual implementation details. The Service Grounding has still to be defined and only lately has there been some initial ideas about its definition.

The matching solution that UDDI utilizes is built on a rigid format for descriptions and restricts the query mechanism. Matching in UDDI is based on exact pattern matching and the process depends highly on categorization of the Web Services. Though this is effective, a matching service based on different levels of specificity and complexity would be more effective.

As stated in [15] a service description is a self-consistent collection of restrictions over named properties of a service. The DAML-S profile offers a more complete description and hence an advantage in the matching process since it can be viewed as a list of functional attributes and descriptions. Concept matching involves the matching of two service descriptions based on the properties that each service has. In [16] an advertisement is considered as a suitable match for a request when the advertisement is sufficiently similar to the request. For *"sufficiently similar"* matches, the algorithm used by the matching engine should be flexible enough so that various degrees of similarity between advertisements and requests are recognized. These degrees of matching as described in the above mentioned paper mainly revolves around the concept of ***subsumption.***

3 DASD (DAML Agents for Service Discovery)

The DASD API consists of a number of Java packages that permits the user to create the various components that make up such an environment as shown in Figure 3 below.

This API is made up of various packages:

- ***Agent package***: takes care of the main GUI and the various viewers necessary to monitor the process of advertising, requesting and invoking a Web Service.
- ***Control package***: controls the agent's process and reasoning component.
- ***Query package***: maps the user's inputs to DAML+OIL requests.
- ***Communication package***: takes care of wrapping a DAML+OIL request in a SOAP message and send/ receives these messages.

- ***Utilities package***: contains various utility classes.
- ***MatchMaker package***: takes care of the MatchMaker agent's engine, reasoning and reply mechanisms.

Both the Requester and Provider agents need to be able to send and receive requests. The difference is that a Requester sends requests to either get a list of matched Web Services or else to invoke one, while a Provider send requests to advertise a Web Service.

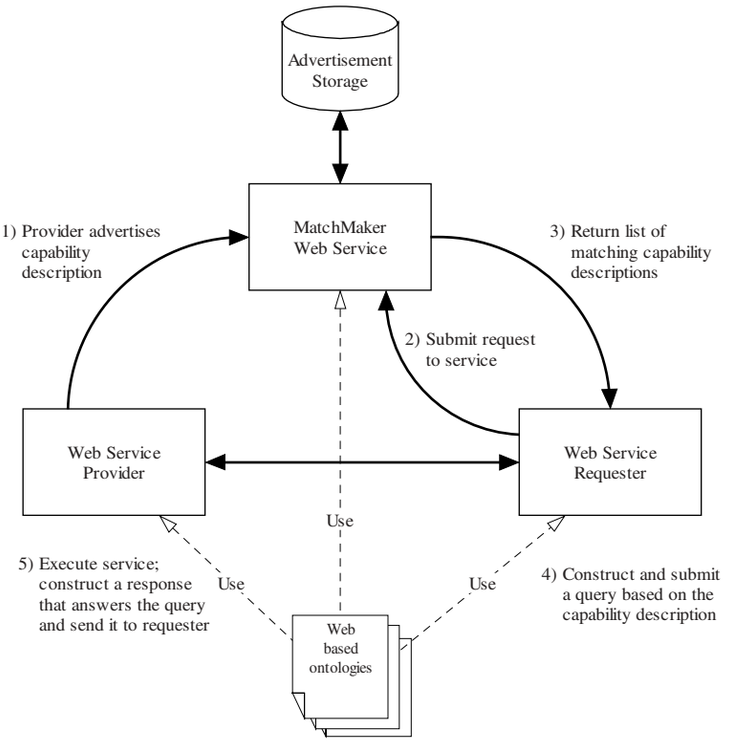


Fig. 3. Overview of the DASD system

3.1 The Requester Agent

A Requester agent runs as a standalone application. It consists of a number of modules working in co-operation with each other. Figure 4 below shows an overview of the Requester's architecture. In our system a Requester can also take the role of a Service Provider, in the sense that after the creation of a particular Web Service

advertisement, the same application offers to the user the facility to submit this new Web Service to the MatchMaker.

The Requester agent waits for the user's inputs to initiate a request. The user makes use of the GUI to enter the information that makes up the request. This information consists of restrictions, such as ***Search By*** (Service Provider, Service Category or Service-Name) and ***Inputs/Outputs*** (required/returned by the Web Service), which are aimed at reducing the search space on the Matchmaker. The Control engine takes care of channelling this input to the correct request builder in the Query module. Here the information is mapped into a DAML+OIL request by making use of templates. This is not the ideal way of creating requests. Ideally the Requester will use the grounding ontology to create the requests automatically. Unfortunately the DAML-S grounding has yet to be defined. Nonetheless we succeeded in producing a hybrid solution to partially solve this problem. This solution consisted in using a mixture of RDF and WSDL. RDF was added to a normal WSDL description by introducing resource definitions for the WSDL constructs.

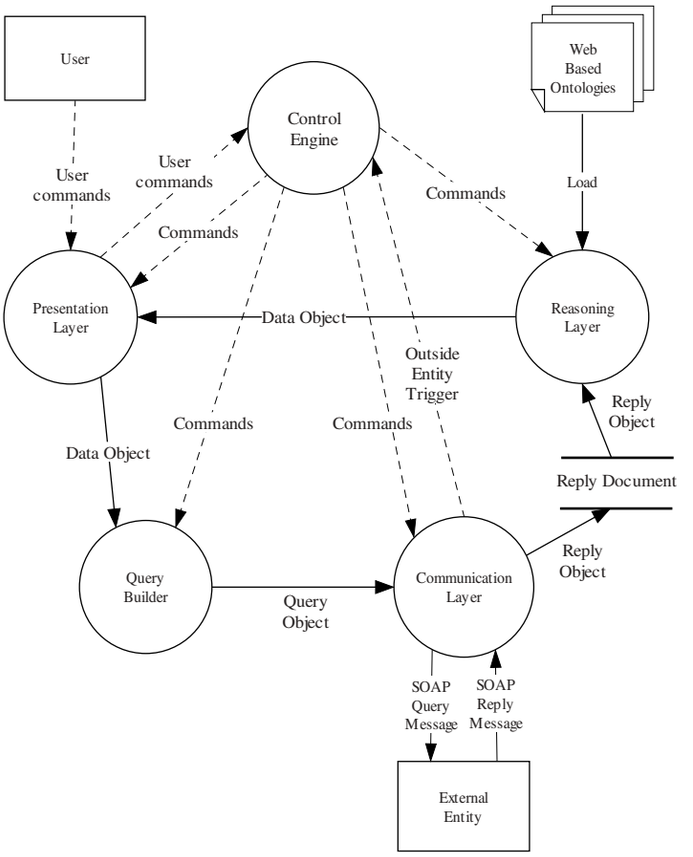


Fig. 4. Overview of the Requester Agent

This made it possible to extract facts from the description by using the reasoner that is integrated in the control module. These facts are of great importance, especially when a invoking Web Services.

The DAML+OIL request is then passed on to the Communication layer where it is wrapped inside a SOAP message and transferred to the appropriate port of the intended agent. SOAP is an ideal protocol, not only because it is being used by the major industrial bodies, but also because it accepts any XML-based content in the "*Body"* of the message.

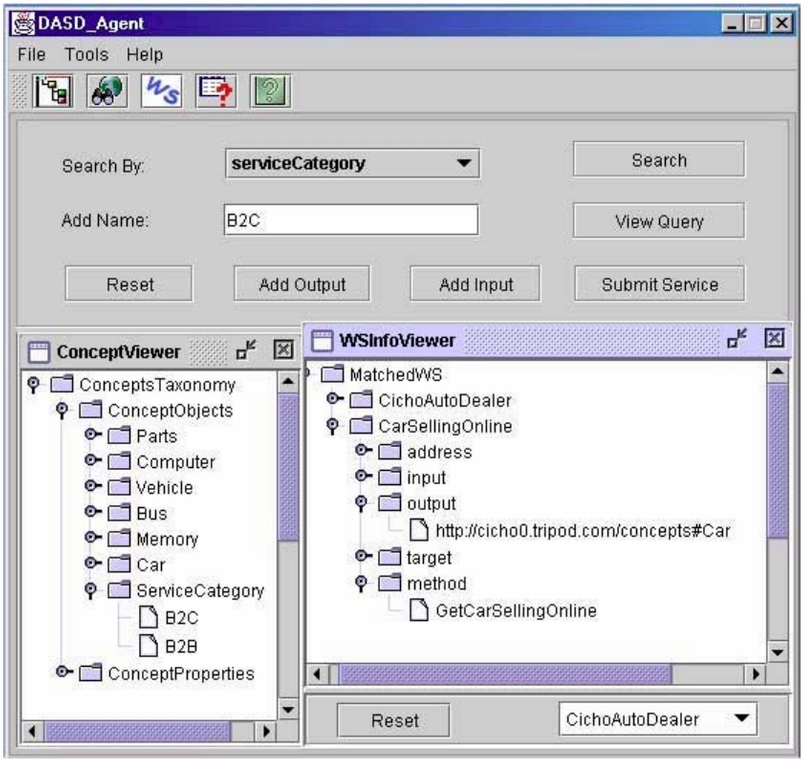


Fig. 5. DASD Requester Application and Web Service Viewers

The Communication module is able to handle the channelling of the various DAML+OIL messages since it is implemented as a proxy client. The information that the reasoner extracts from the RDF/WSDL ontology is used to create different SOAP messages to different Web Services. This information consists of the name, address, target address and inputs of the Web Service, together with the method to invoke on the service.

In the case of a Service Provider, the request is handled in a similar way accept that the DAML+OIL request contains the URL of the service profile that is going to be advertised with the MatchMaker.

The replies are handled in the reverse order of the requests. The Communication module performs some pre-processing on the received message. This consists in the creation of a temporary DAML+OIL file representation of the reply. This is due to the

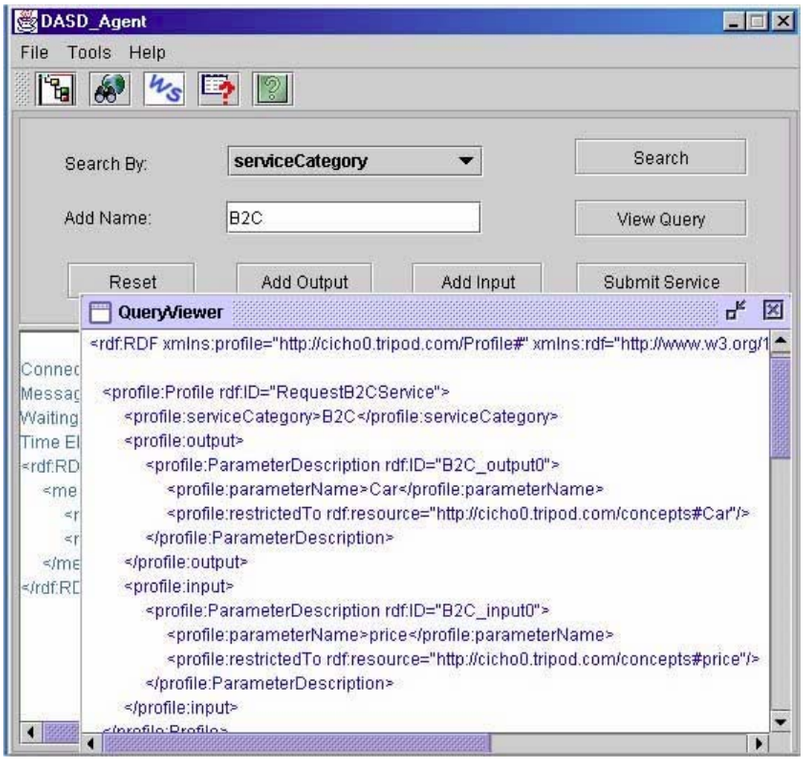


Fig. 6. DASD Requester's Query Viewer

fact that the FOL reasoner is only capable of importing ontologies in this form. To extract the information from the reply message, this file is loaded by the reasoning component and is then transformed into a set of triples, which is stored in memory. Queries are then made on this representation and action is taken according to the results of these queries. Such action might consist in displaying a list of the URLs that belong to the matched Web Services' profiles or displaying the structure of the concepts ontology that is shared by both the MatchMaker and the Requester in the appropriate viewer.

3.2 The MatchMaker Agent

We now turn our attention to the process that goes on in the MatchMaker agent, an overview of which can be seen in Figure 7 below. The basic functionality is similar to that of the Requester and Provider agents since it is able to receive requests and return replies. The additional functionality on the MatchMaker is the process by which it is capable of matching advertisements that are stored in its database to the requests made by Service Requesters. A request for a Web Service is similar to a succinct version of an advertisement.

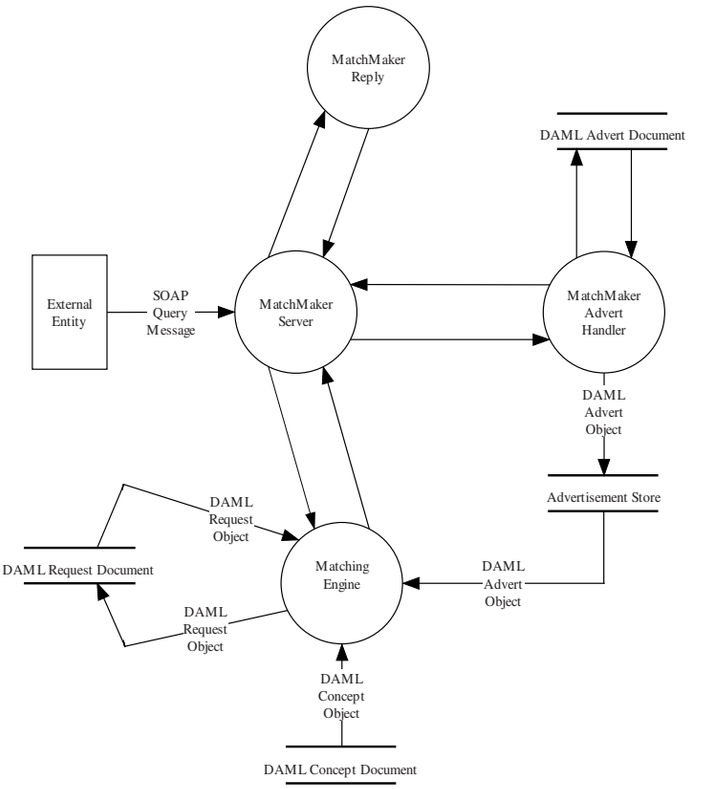


Fig. 7. DASD MatchMaker Architecture

When a message is received its type is verified to identify whether it is a request or an advertisement. We will first consider the process were the MatchMaker handles a request and then the process where an advertisement is handled. In the first case, the request is transformed into a DAML+OIL file representation and loaded into memory by the reasoner as described in the previous paragraphs. Then the matching algorithm

is executed. This algorithm consists of two filtering phases. The first filter consists of reducing the search space by considering the *SearchBy* restriction defined in the request. At this stage the database is searched and all the Web Services that match this criteria are returned. The second filter considers the output concepts defined by the user in the request, and tries to match these with those of a Web Service that was returned by the first filtering phase. This part of the matching process is again carried out in two stages. The first stage takes the two output concepts (one from the request and one from the advertisement) and checks whether they are equal. Equality is considered true if the two output concepts unify. If the first stage fails then a subsumption match is performed, where the two concepts are tested to see whether the request concept is a subclass of the advertisement concept. If any of these two stages are found to be true then this will result in a match. The process is repeated for the other output concepts defined in the request. If no Web Service fully satisfies the request then a failure is detected. In any case, the results are wrapped in a DAML+OIL message and transported in the Body of a SOAP message to the Requester.

If the request was an advertisement, then by using a similar process as described above, the message is transformed into a set of triples and the URL to the advertised Web Service is extracted. When a Provider advertises a Web Service he submits the URL that refers to the Profile-Summary ontology and not to the actual Profile. This summary is used to speedup the process of loading advertisements when matching is performed. The MatchMaker extracts from this summary the necessary information, such as the service provider name, the service category name and the service name, and stores it together with the full profile's URL in different indices in its database. If this process of storing an advertisement is successful then an acknowledgement is returned to the Provider in a DAML+OIL message, announcing this success. In case that the storing of the advertisement fails for some reason, the acknowledgement with a reason for this failure is returned.

4 Evaluation

To test our system we have subjected a number of Requester/Provider agents and our MatchMaker to various requests, making use of the ontologies we created. These ontologies consist of a number of Web Service profiles defined in DAML-S and their relevant groundings are defined by an RDF enhanced version of WSDL definitions. The domains considered are those of vehicles and computer hardware. We created a "*concepts"* ontologies, which is shared by all the agent components, and were we described all the terminology used by the agents in the process of discovery, invocation and execution of the Web Services. We also created a *"rules"* ontology that is used by the agents to identify which actions to take upon being triggered. Some of the results we found are annotated below. These ontologies can be found in the ontologies directory at http://alphatech.mainpage.net/.

One such test that was carried out with the Requester consisted in requesting a search for a Web Service that is in the "*ServiceCategory"* of *"B2C"* (Business to Consumer)

and whose output is a *"Coupe"* type of car. This returned as expected, a list of the two services that were stored in the MatchMaker's database and that had the concept *"Car"* as output (since Coupe was defined as being a subclass of Car in the concepts ontology). This tested the effectiveness of the algorithm in identifying subclass concepts.

In another test we invoked the same service category as above, but asked for a *"MiniBus"* type of vehicle. This returned a *"No Matched Web Service Found"* message since there was no *"B2C"* Web Service available with these restrictions. We then changed the service category to *"B2B"* and a matched Web Service was found as expected. Similar tests were carried out with the other query restrictions ("ServiceProvider" and "ServiceName") and the results were all as expected.

We then tested the invocation of the Provider's Web Service whose output is the concept *"Car"*. This invocation was completed successfully when the correct concepts were presented to the service. For example we supplied the concept *"MiniBus"*, which is a vehicle but is not considered to be a *"Car"* in the concepts ontology, and as expected a message of *"No cars available"* is returned. But when we supplied the concept *"Coupe"* a list of available coupe models specifications was returned.

5 Conclusion

This paper describes the DASD environment as the result of a research project that integrated two areas of research, namely those of the Semantic Web and Web Services. Due to the novelty and limitation of the present technologies we had to resort to a hybrid solution that integrated several technologies together and made the final product possible.

The system in itself presents the user with a searching facility that makes use of semantics to retrieve and eventually invoke Web Services. This process is made as transparent as possible to the user thus relieving him from the necessity of learning new concepts. The creation of the several viewers helps in this process by displaying the generated queries that are to be submitted to the matchmaker, by displaying the information about each of the Web Services retrieved by the matchmaker and by suitably creating on the fly, appropriate dialogue boxes where the user can input the necessary data to invoke the different Web Services. We succeeded in solving several important issues such as the creation of the matching algorithm that handles subsumption matching between concepts in requests and advertisements, the invocation of different Web Services through the same proxy client by making use of the RDF/WSDL grounding ontologies and also to implement some reasoning capabilities within the system that made it possible for the DASD agents to reason about facts found in the ontologies. The transformation of the WSDL by enhancing it with RDF made it possible for us to use the inbuilt reasoner to extract and manipulate facts in the ontology which otherwise would not have been possible, since WSDL does not support any semantic content.

We have obtained some promising results that highlighted the importance and benefits that can be obtained by integrating semantic web languages such as RDF and DAML+OIL to the important area of Web Services and related technologies. In our opinion there should be a combined effort so that further research in these areas focus on such integration of technologies as we described in this paper.

References

1. DAML-S, http://www.daml.org/services/
2. DAML+OIL (March 2001) Reference Description http://www.w3.org/TR/daml+oil-reference
3. Resource Description Language http://www.w3.org/RDF/
4. Annotated WSDL with RDF http://www.w3.org/2001/03/19-annotated-RDF-WSDL-examples
5. JTP user manual http://www.stanford.edu/~gkfrank/jtp/
6. WWW Consortium, http://www.w3.org/
7. Tim Berners-Lee, Semantic Web Road Map, (1998) http://www.w3.org/DesignIssues/Semantic.html
8. Tim Berners-Lee, James Hendler , Ora Lassila, *The Semantic Web*, Scientific American, May 2001, http://www.scientificamerican.com/2001/0501-issue/-0501-berners-lee.html
9. Web Services Activity, http://www.w3.org/2002/ws/
10. The DAML Services Coalition: Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, Katia Sycara, Honglei Zeng. *DAML-S: Semantic Markup For Web Services*. In *Proceedings of the International Semantic Web Workshop*, 2001
11. Universal Discovery Description and Integration Protocol, http://www.uddi.org/
12. Web Services Description Language, http://www.w3.org/TR/wsdl
13. Simple Object Access Protocol, http://www.w3.org/TR/soap12-part0/
14. The DAML Services Coalition: Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara. *DAML-S: Web Service Description for the Semantic Web*. Appeared In *The First International Semantic Web Conference (ISWC)*, 2002
15. J. Gonzalez-Castillo, D. Trastour, C. Bartolini, *Description Logics for MatchMaking of Service*, KI 2001,http://www.hpl.hp.com/techreports/2001/HPL-2001-265.pdf
16. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. *Semantic Matching of Web Services Capabilities*. Appeared in *The First International Semantic Web Conference (ISWC)*, 2002

Building Reliable Web Services Compositions

Paulo F. Pires[1], Mario R.F. Benevides[1,2], and Marta Mattoso[1]

[1] System Engineering and Computer Science Program – COPPE
[2] Institute of Mathematics
Federal University of Rio de Janeiro
P.O. Box 68511, Rio de Janeiro, RJ, 21945-970, Brazil
{pires, mario, marta}@cos.ufrj.br

Abstract. The recent evolution of internet technologies, mainly guided by the Extensible Markup Language (XML) and its related technologies, are extending the role of the World Wide Web from information interaction to service interaction. This next wave of the internet era is being driven by a concept named *Web services*. The Web services technology provides the underpinning to a new business opportunity, i.e., the possibility of providing value-added Web services. However, the building of value-added services on this new environment is not a trivial task. Due to the many singularities of the Web service environment, such as the inherent structural and behavioral heterogeneity of Web services, as well as their strict autonomy, it is not possible to rely on the current models and solutions to build and coordinate compositions of Web services. In this paper, we present a framework for building reliable Web service compositions on top of heterogeneous and autonomous Web services.

1 Introduction

Web services can be defined as modular programs, generally independent and self- describing, that can be discovered and invoked across the Internet or an enterprise intranet. Web services are typically built with XML, SOAP, WSDL, and UDDI specifications [19]. Today, the majority of the software companies are implementing tools based on these new standards [7,10]. Considering how fast implementations of these standards are becoming available, along with the strong commitment of several important software companies, we believe that they will soon be as widely implemented as HTML is today.

According to the scenario just described, an increasing number of on-line services will be published in the Web during the next years. As these services become available in the Web service environment, a new business opportunity is created, i.e., the possibility of providing value-added Web services. Such value-added Web services can be built through the integration and composition of basic Web services available on the Web [3].

Web service composition is the ability of one business to provide value-added services to its customers through the composition of basic Web services, possibly offered by different companies [3,4]. Web service composition shares many

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 59–72, 2003.

requirements with business process management [1]. They both need to coordinate the sequence of service invocation within a composition, to manage the data flow between services, and to manage execution of compositions as transaction units. In addition, they need to provide high availability, reliability, and scalability. However, the task of building Web service compositions is much more difficult due to the degree of *autonomy*, and *heterogeneity* of Web services. Unlike components of traditional business process, Web services are typically provided by different organizations and they were designed not to be dependent of any collective computing entity. Since each organization has its own business rules, Web services must be treated as strictly autonomous units. Heterogeneity manifests itself through structural and semantic differences that may occur between semantically equivalent Web services. In a Web service environment it is likely to be found many different Web services offering the same semantic functionality thus, the task of building compositions has to, somehow, deal with this problem.

This paper introduces a framework, named *WebTransact*, which provides the necessary infrastructure for building reliable Web service compositions. The WebTransact framework is composed of a multilayered architecture, an XML-based language (named *Web Services Transaction Language*), and a transaction model. The multilayered architecture of WebTransact is the main focus of this paper. A detailed discussion on all other components of WebTransact can be found in [11].

The remainder of this paper is organized as follows. In Section 2, we start presenting a general picture of the WebTransact architecture. Next, we explain the components of that architecture. In Section 3, we discuss the related work. Finally, in Section 4, we present our concluding remarks.

2 The WebTransact Architecture

As shown in Fig. 1, WebTransact[1] enables Web service composition by adopting a multilayered architecture of several specialized components [12]. Application programs interact with *composite mediator services* written by composition developers. Such compositions are defined through transaction interaction patterns of *mediator services*. Mediator services provide a homogenized interface of (several) semantically equivalent *remote services*. Remote services integrate Web services providing the necessary mapping information to convert messages from the particular format of the Web service to the mediator format.

The WebTransact architecture encapsulates the message format, content, and transaction support of multiple Web services and provides different levels of value- added services. First, the WebTransact architecture provides the functionality of uniform access to multiple Web services. Remote services resolve conflicts involving the dissimilar semantics and message formats from different Web services, and conflicts due to the mismatch in the content capability of each Web service.[2] Besides resolving structural and content conflicts, remote services also

[1] The architecture of the WebTransact is based on the Mediator technology [20].
[2] These conflicts are better explained in Section 2.3.

provide information on the interface and the transaction semantics supported by Web services. Second, Mediator services integrate semantically equivalent remote services providing a homogenized view on heterogeneous Web services. Finally, transaction interaction patterns are built on top of those mediator services generating composite mediator services that can be used by application programs or exposed as new complex Web services.

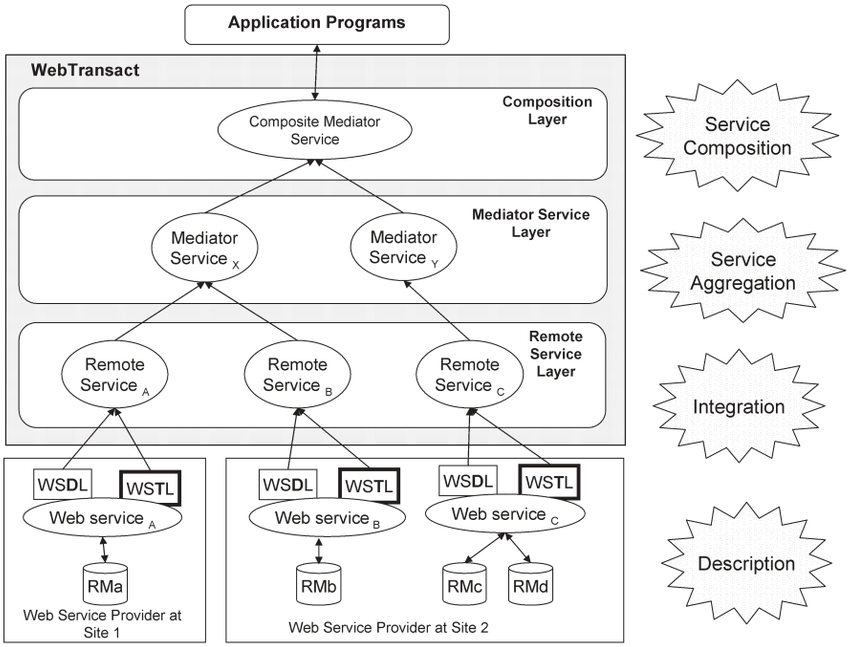


Fig. 1. The multilayered architecture of WebTransact.

WebTransact integrates Web services through two XML-based languages: Web Service Description Language (WSDL) [19], which is the current standard for describing Web service interfaces, and Web Service Transaction Language (WSTL) [11], which is our proposal for enabling the transactional composition of heterogeneous Web services. WSTL is built on top of WSDL extending it with functionalities for enabling the composition of Web services. Through WSDL, a remote service understands how to interact with a Web service. Through WSTL, a remote service knows the transaction support of the Web service. Besides the description of the transaction support of Web services, WSTL is also used to specify other mediator related tasks such as: the specification of mapping information for resolving representation and content dissimilarities, the definition of mediator service interfaces, and the specification of transactional interaction patterns of Web service compositions.

The distributed architecture of WebTransact separates the task of aggregating and homogenizing heterogeneous Web services from the task of specifying transaction interaction patterns, thus providing a general mechanism to deal with the complexity introduced by a large number of Web services.

The design of the WebTransact framework provides novel special features for dealing with the problems of Web service composition. Since mediator services provide a homogenized view of Web services, the composition developer does not have to deal with the heterogeneity nor the distribution of Web services. Another import aspect of the WebTransact architecture is the explicit definition of transaction semantics. Since Web services describe their transaction support through WSTL definition, reliable interaction patterns can be specified through mediator service compositions.

2.1 The Remote Service Layer

Each remote service in WebTransact is a logical unit of work that performs a set of *remote operations* at a particular site. Besides its signature, a remote operation has a well-defined *transaction behavior*. The transaction behavior defines the level of transaction support that a given Web service exposes. There are two levels of transaction support. The first level consists of Web services that cannot be cancelled after being submitted for execution. Therefore, after the execution of such Web service, it will either commit or abort, and if it commits, its effects cannot be undone. The second level consists of Web services that can be aborted or compensated. There are two traditional ways to abort or compensate a previous executed service. One way, named *two-phase commit* (2PC) [8], is based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. Another way, called *compensation*, is based on the idea that a constituent transaction is always allowed to commit, but its effect can be cancelled after it has committed by executing a compensating transaction. In order to accommodate these levels of transaction support, the WebTransact framework defines four types of transaction behavior of remote services, which are: *compensable*, *virtual-compensable*, *retriable*, or *pivot*. A remote operation is *compensable* if, after its execution, its effects can be undone by the execution of another remote operation. Therefore, for each compensable remote operation, it must be specified which remote operation has to be executed in order to undo its effects. The *virtual-compensable* remote operation represents all remote operations whose underlying system supports the standard 2PC protocol. These services are treated like compensable services, but, actually, their effects are not compensated by the execution of another service, instead, they wait in the prepare-to-commit state until the composition reaches a state in which it is safe to commit the remote operation. Therefore, virtual-compensable remote operations reference Web service operations whose underlying system provides (and exposes) some sort of distributed transaction coordination. A remote operation is *retriable*, if it is guaranteed that it will succeed after a finite set of repeated executions. A remote operation is *pivot*, if it is neither retriable nor compensable. For example, consider a simple service for buying flight tickets. Consider

that such service has three operations, one, `reservation`, for making a flight reservation, another, `cancelReserv`, for canceling a reservation, and another, `purchase`, for buying a ticket. The `reservation` operation is compensable since there is an operation for canceling reservations. The `cancelReserv` operation is retriable since it eventually succeeds after a finite set of retries. On the other hand, the `purchase` operation is pivot because it cannot be undone (supposing non refundable ticket purchases).

In WebTransact, the concept of compensable, virtual-compensable, and pivot operation is used to guarantee safe termination of compositions. Due to the existence of dissimilar transaction behavior, some Web service providers may not support transaction functionalities like the two-phase commit interface. In this case, it is not possible to put a remote operation in a wait state like the prepared-to-commit state. Therefore, after a pivot service has been executed, its effects are made persistent and, as there is no compensating service for pivot services, it is not possible to compensate its persistent effects. For that reason, pivot remote operations can be executed only when, after its execution, the composition reaches a state where it is ready to successfully commit. At this state, the system has guaranteed that there will be no need to undo any already committed service.

WSTL Document Example. The following example (Fig. 2, Fig. 3, and Fig. 4) shows a WSDL definition of a simple service providing car reservations, extended by the WSTL framework. The car reservation service supports two operations: `reservation` and `cancelReservation`.

The `reservation` operation returns a reservation code, or an error, in the parameter `reservationResult` of type `string`. The `cancelReservation` operation has only one parameter: `reservationCode` of type `string`. A valid value for this parameter is the value returned in a previous successfully executed request of operation `reservation`. The `cancelReservation` operation returns a value of type `string` indicating the success or failure of the executed request.

The definition of the car reservation service is separated in three documents:[3] data type definitions (Fig. 2), abstract definitions (Fig. 3), and transaction behavior definitions (Fig. 4).

```
<definitions
  targetNamespace="http://example.com/carReservation/Schema/"
  ...
  xmlns:tns="http://example.com/carReservation/Schema/"
  <types>
    ...
    <xsd:element name="reservationResponse">
      <xsd:complexType>
```

[3] Since the concrete WSDL definitions are not used for explaining the WSTL usage, they are not shown.

```
        <xsd:sequence>
         <xsd:element name="reservationResult" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="cancelReservation">
      <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="reservationCode" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="cancelReservationResponse">
      ...
    <xsd:element name="string" type="xsd:string"/>
    ...
  </types>
</definitions>
```

Fig. 2. Types definition for the car reservation service.

```
<definitions
  targetNamespace="http://example.com/carReservation/abstractDef/"
  ...
  xmlns:lxsd="http://example.com/carReservation/Schema/">
  ...
  <message name="reservationSoapIn">
    <part name="parameters" element="lxsd:reservation"/>
  </message>
  <message name="reservationSoapOut">
    <part name="parameters" element="lxsd:reservationResponse"/>
  </message>
  <message name="cancelReservationSoapIn">
    <part name="parameters" element="lxsd:cancelReservation"/>
  </message>
  <message name="cancelReservationSoapOut">
    <part name="parameters"
          element="lxsd:cancelReservationResponse"/>
  </message>
  <portType name="reservationSoap">
    <operation name="reservation">
      <input message="tns:reservationSoapIn"/>
      <output message="tns:reservationSoapOut"/>
    </operation>
```

```
    <operation name="cancelReservation">
      <input message="tns:cancelReservationSoapIn"/>
      <output message="tns:cancelReservationSoapOut"/>
    </operation>
  </portType>
</definitions>
```

Fig. 3. Abstract definitions for the car reservation service.

```
<definition ...
  xmlns:absd="http://example.com/carReservation/abstractDef/"">
  <wstl:transactionDefinitions>
    <wstl:transactionBehavior operationName=" absd:reservation"
                              type="compensable">
      <wstl:activeAction portTypeName="svc:reservationSoap"
                         compensatoryOper="cancelReservation">
        <wstl:paramLink>
          <wstl:sourceParamLink
             msgName="reservationSoapOut"
             param="absd:reservationResponse/@reservationResult"/>
          <wstl:targetParamLink
             msgName="cancelReservationSoapIn"
             param="absd:cancelReservation/@reservationCode"/>
        </wstl:paramLink>
      </wstl:activeAction>
    </wstl:transactionBehavior>
    <wstl:transactionBehavior
                  operationName=" absd:cancelReservation"
                  type="retriable"/>
    </wstl:transactionDefinitions>
</definitions>
```

Fig. 4. Transaction behavior definitions for the car reservation service.

Considering the WSDL concept of abstract and concrete definitions, the transaction behavior describes the transaction semantics of *abstract operations* of Web services. The transaction behavior is a semantic concept related to operations, thus it is independent of network deployment or data format bindings of concrete endpoints and messages. Therefore, the transaction behavior should be inserted as a child element of a port type operation that describes the abstract portion of message exchanges. However, WSDL does not allow extensibility elements inside port type operations. The only WSDL element that supports extensibility and is located in the context of the abstract portion of a WSDL

document is the definitions element. For this reason, WSTL defines its root element, `transactionDefinitions`,[4] as a direct child of the `wsdl:definitions` element.

In the car reservation service (Fig. 4), the `transactionDefinitions` element has two child `transactionBehavior` elements each containing transaction semantics information on the operations supported by that Web service.

The first `transactionBehavior` element has `"absd:reservation"` as the value of attribute `operationName`. The prefix `absd:` references the namespace `http://example.com/carReservation/abstractDef/`, which is the namespace for the abstract definitions of the car reservation service. The value `"absd:reservation"` is a `QNAME` that references the `wsdl:operation` element named `reservation`, which is defined in the WSDL document of Fig. 3. Therefore, the first `transactionBehavior` element in Fig. 4 defines the transaction behavior of the `reservation` operation of the car reservation service. The value `"compensable"` of attribute `type` indicates that `reservation` operation is *compensable*, as defined in Section 2.1. The compensatory operation of the compensable operation `reservation` is defined by the `activeAction` element that has `"absd:reservationSoap"` and `"absd:cancelReservation"` as the values of attributes `portTypeName` and `compensatoryOper`, respectively. These values define the `cancelReservation` operation of port type `reservationSoap` from the car reservation service (prefix `absd:`) as the compensatory operation for the `reservation` operation.

The `paramLink` element, which is a child element of the `activeAction` element, represents a data link involving a message part of the compensable operation `reservation` to a message part of its compensatory operation `cancelReservation`. This data link specifies a data flow from the compensable operation `reservation` to its compensatory operation `cancelReservation`, prescribing how the input parameters of the operation `cancelReservation` is constructed from the output parameters of the `reservation` operation. The `paramLink` element contains two child elements: the `sourceParamLink` and `targetParamLink` elements. The `sourceParamLink` element defines the origin of the data flow, while the `targetParamLink` element defines the destination of the data flow. In the example, the `sourceParamLink` element has `"absd:reservationSoapOut"` as its `msgName` attribute and the XPath expression `"absd:reservationResponse/@reservationResult"` as the value for its `param` attribute, while The `targetParamLink` element has `"absd:cancelReservationSoapIn"` as its `msgName` attribute and the XPath expression `"absd:cancelReservation/@reservationCode"` as the value for its `param` attribute. These elements link the result value of the operation `reservation` must to the input parameter of the operation `cancelReservation`.

The data link defined above is used by WebTransact to construct the input message of the compensatory operation `cancelReservation` when invoking it to compensate the work done by the operation `reservation` during an execution of a given transaction.

[4] The complete specification can be found in [11].

The second `transactionBehavior` element has `"absd:cancelReservation"` as the value of attribute `operationName`. This value indicates that this `transactionBehavior` element defines the transaction behavior of the `cancelReservation` operation of the car reservation service. The value `"retriable"` of attribute `type` indicates that the `cancelReservation` operation is *retriable*, as defined in Section 2.1. Note that there is no child element for this `transactionBehavior` element. The reason is that retriable operations are not compensable, thus there is no other necessary information on the operation transaction behavior to be provided by the Web service. Only *compensable* and *virtual-compensable* operations need further information besides the information available in the set of attributes of the `transactionBehavior` element.

In this section, we have only shown a simple example of the WSTL elements for describing *compensable* and *retriable* Web service operations. Other examples, including examples of virtual-compensable operations, can be found in [11].

2.2 The Mediator Service Layer

Mediator services aggregate semantically equivalent remote services, thus providing a homogenized view of heterogeneous remote services. Semantically equivalent remote services are services that integrate Web services exposing different WSDL interfaces but providing the same semantic functionality. Unlike remote services, which are logical units of work that perform remote operations at a particular site, mediator services are *virtual* services responsible for delegating its operations' execution to one or more remote services. This delegation is done over a set of semantically equivalent remote services aggregated by the mediator service. Like remote operations, mediator service operations have a signature and a well-defined *transaction behavior*, which can be either *compensable*, *retriable*, or *pivot*.

The transaction behavior of one mediator service operation is based on the transaction behavior of its aggregated remote operations. If all aggregated remote operations have the same type of transaction behavior, e.g. *compensable*, then the transaction behavior of mediator service operation will have the same value, i.e., *compensable*. On the other hand, if the mediator service operation aggregates remote operations with different transaction behaviors, then its transaction behavior will be the *least restrictive* transaction behavior among the transaction behaviors of its aggregated remote operations. The most restrictive transaction behavior is the pivot, followed by the retriable transaction behavior, while both the compensable and virtual-compensable transaction behaviors are least restrictive transaction behaviors. The concept of least/most restrictive transaction behavior defines whether a mediator service operation can participate in a given composition execution. A mediator service operation, which aggregates at least one remote operation that is compensable (or virtual-compensable), can participate in any composition execution. On the other hand, a mediator service operation that aggregates only pivot (or retriable) remote operations can participate only in compositions that call this mediator service operation after it reaches a state where it is ready to successfully commit. Recall from Section 2.1

that after a pivot remote operation has been executed, the composition has to enter a state where it is ready to successfully terminate. Thus, it is guaranteed that there will be no need to undo any already submitted pivot (or retriable) remote operation. Therefore, mediator service operations that aggregate only pivot (or retriable) remote service operations can only participate in compositions that call this mediator service operation when it is ready to successfully terminate.

Mediator service operations expose the same types of transaction behavior as remote service operations, except for the absence of the virtual-compensable operation. The specific transaction behavior of virtual-compensable remote operations is isolated by the mediator service operation that aggregates them. Mediator service operations that aggregate virtual-compensable remote operations expose the same interface of mediator service operations that aggregate only real compensable remote operations. This simplifies the protocol that coordinates the composition execution, since both compensable and virtual-compensable remote service operations are treated, at the composition level, as a single operation type.

2.3 Resolving Semantic and Content Dissimilarities of Web Services

In order to provide a homogenized layer of services, each mediator service exposes a single interface that is used by composition specifications. As mediator services aggregate semantically equivalent remote services, which possibly have different interfaces, it is necessary to provide mapping information between the interface supported by the mediator service and each one of the interfaces supported by its aggregated remote services. In WebTransact, each WSDL port type is imported as a new remote service. Since the WSDL port type element defines the syntax for calling a set of remote operations, i.e., a specific supported interface, each WSDL port type definition is considered as a separated remote service. A remote service links a mediator service to a WSDL port type element and it provides *mapping information* between mediator service operations and port type operations and specifies the *content description* of the remote service. The mapping information prescribes how the input parameters of a remote service operation are constructed from the input parameters of its related mediator service operation, as well as how the output parameters or fault messages received from that remote service operation are mapped to the output or fault messages of its related mediator service operation. The mapping information also defines the *transaction outcome* of a given remote service operation. In order to define messages signalizing an unsuccessful terminating state, all that is needed is the definition of mapping information linking fault messages and/or specific return value of output messages of a remote service operation to a fault message of a mediator service operation. The content description specifies whether a remote service is able to execute a particular service invocation. For example, consider remote services rm_1 and rm_2 providing car reservations. Remote service rm_1 can make car reservations world wide, while remote service rm_2 accepts only car reservations in Brazil. Now, consider that mediator service ms_1 aggregates

rm_1 and rm_2. If ms_1 receives a request to make a car reservation inside USA then, ms_1 will invoke only the remote service rm_1. The mediator service ms_1 knows, using the remote service content description, that rm_2 is not able to make car reservation outside Brazil. Mediator services and remote services are both specified through WSTL definitions.

2.4 The Composition Layer

A composite mediator service describes transaction interaction patterns from a set of cooperating mediator service operations necessary to accomplish a task. Such interaction pattern defines the execution sequence of mediator service operations as well as the level of atomicity and reliability of a given composition. In WebTransact, a composition is specified using WSTL elements. The WSTL elements for specifying composite mediator service as well as their operational semantics are described in [11].

WSTL models compositions as *composite tasks*. A composite task is represented by a labeled directed graph in which nodes represent steps of execution and edges represent the flow of control and data among different steps. Each step of a composite task is either an *atomic task* or another composite task. An atomic task is a unit of work that is executed by a mediator service operation. Therefore, atomic tasks have a mediator service operation assigned to it, which is invoked when the task is executed.

Tasks are identified by a name and have a *signature*, a set of *execution dependencies*, a set of *data links*, and, optionally, a set of *rules*.

The *signature* of an atomic task is related to the input, output, and fault messages of the mediator service operation that is used as the implementation of the task. The *signature* of a composite task is related to the input, output, and fault messages of the component tasks used as the implementation of the task. A component task can be an atomic task or another composite task.

Execution dependencies are based on the execution state of component tasks defining the first kind of edges in the graph that represent a composite task. An execution dependency is defined among related component tasks and it is a constraint on the temporal occurrence of the start and termination events of them. Execution dependencies define the order in which tasks must be executed, i.e., the composition control flow. An execution dependency is always specified based on the *execution state* of one component task.

Data links are mappings between messages belonging to the signatures of related tasks to allow the exchange of information between these tasks. Data links are the second kind of edges in the graph that represents a composite task.

Rules specify the conditions under which certain events will happen. Rules can be associated to dependencies or to data links. A dependency that has a rule will evaluate to true if both the dependency *and* the rule evaluate to true. A data link that has a rule will evaluate to true if the rule evaluates to true. Data links without explicit rules always evaluate to true.

Besides the components described above, composite tasks have a set of *mandatory tasks*. This set specifies the component tasks that must commit in

order to commit the composite task. A user can specify a composite task aggregating tasks that are desirable, but not essential, to accomplish the target task. The set of mandatory tasks allow the distinction between *desirable* and *mandatory* tasks, providing more flexibility while specifying a composition. This flexibility increases the composition robustness, since the set of tasks required to commit, in order to commit the composition, are formed only by that tasks that are essential to accomplish the composition target. Thus, the composition will successfully terminate even if a subset of its component tasks fails, as long as all its mandatory tasks successfully commit.

3 Related Work

The WebTransact framework is a multidisciplinary work that is related with many other areas such as e-service composition, transactional process coordination, workflow management systems, and distributed computing systems.

There has been extensive research in transaction support for distributed computing systems [5,14], transactional process coordination [2,15], and in workflow management systems [6]. While these projects address the support of distributed transactions, they do not consider the coordination of service capabilities of autonomous remote service providers. Since Web services support dissimilar capabilities with respect to its transaction behavior, this is a significant difference. Thus, implementing transaction semantics across the Web services becomes much more difficult when compared to a scenario where all distributed components support identical transaction behavior. Moreover, these works were conceived before the current stage of the World Wide Web. Therefore, they do not consider the XML- based standards that enable the Web service technology.

The existent works in the area of e-service composition (WSFL [9], XLANG [16], and WSCL [18]) are concentrated in defining primitives for composing services and automating service coordination. The majority of these works consider XML-based standards for Web service technology. However, the primitives for composition proposed by these works do not directly address the problems associated with the necessary *homogenization* of Web services. In addition, the transaction support proposed in this area does not consider the coordination and mediation of Web services with dissimilar transaction support.

More recently, some specific transaction protocols for the Web have being discussed, such as the W3C tentative hold protocol (THP) [17], and the OASIS Business Transaction Protocol (BTP) [13]. These protocols define a model for coordinating the transaction execution of Web services based on a predefined set of transaction messages. Still, they propose a transaction model based on a relaxed notion of the all-or-nothing property of conventional transactions. While these works provide support for distributed transactions on the Web environment through enforcing a predefined set of transaction messages, WSTL provides a mean of coordinating distributed transactions on the Web without enforcing all participants (Web services) to support a unique and standard transaction

protocol. In this sense, WSTL provides more flexibility while preserving the autonomy of Web services.

4 Conclusions

The Web services technology provides the underpinnings to a new business opportunity where one company can offer value-added services to its customers through the composition of basic Web services. However, the current Web services technology solves only part of the problem of building Web services compositions. Building reliable Web service compositions requires much more than just addressing interoperability between client programs and Web services. Besides interoperability, building Web services compositions requires mechanisms for: describing the dissimilar transaction support of Web services, resolving the semantic and content heterogeneity of semantically equivalent Web services, specifying the transaction interaction patterns among Web services, and coordinating such interaction patterns. Still, due to this novel set of requirements, posed by the Web services environment, existent business process frameworks cannot be directly applied to develop Web service compositions. Therefore, there is a need of new frameworks, specifically developed for addressing the new requirements of the Web service environment. In this paper, we have introduced one of such frameworks, the WebTransact framework.

WebTransact treats the problem of building composition in an integrated way, providing mechanisms for describing the dissimilar transaction behavior of Web services, for aggregating semantically equivalent Web services and for resolving their heterogeneities, for specifying reliable interaction patterns of Web services, and for coordinating such interaction patterns in a transactional way. To the best of our knowledge, there is no other works on integrated frameworks for Web service composition addressing the requirements encompassed by WebTransact.

References

1. Alonso, G., Fiedler, U., Hagen, C., et al.: WISE: Business to Business E-Commerce. In: Proceedings of RIDE. Sydney, Australia, (1999).
2. Alonso, G., Schuldt, H., Schek, H.: Concurrency Control and Recovery in Transactional Process Management. In: Proceedings of the Symposium on Principles of Database Systems. Philadelphia, (1999) 316–26
3. Casati, F., Ilnicki, S., Jin, L., et al.: Adaptive and Dynamic Service Composition in eFlow. In: Proceedings of CaiSE 2000. Stockholm, (2000) 13–31
4. Casati, F., Shan, M.: Dynamic and Adaptive Composition of E-services. Information Systems, Vol. 26 **1**. (2001) 143–163
5. Elmagarmid, A. K. (ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann. (1992)
6. Georgakopoulos, D., Hornick, M., Sheth, A.: An overview of workflow management: from process modeling to workflow automation infrastructure. Intl. Journal on distributed and parallel databases, Vol. 3 **2** (1995) 119–153

7. IBM White Paper: The IBM WebSphere software platform and patterns for e-business – invaluable tools for IT architects of the new economy. (2000) [http://www4.ibm.com/software/info/websphere/ docs/wswhitepaper.pdf].
8. Lampson, B. W.: Atomic Transactions. In: Goos, G., Hartmanis, J. (eds.), Distributed Systems – Architecture and Implementation: An Advanced course. Springer-Verlag (1981) 246–265
9. Leymann, F.: Web Services Flow Language (WSFL 1.0). (2001) [http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf]
10. Microsoft White Paper: A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform. (2000) [http://www.microsoft.com/commerceserver/techres/whitepapers.asp].
11. Pires, P. F., Benevides, R. F. M., Mattoso, M.: WebTransact: A Framework for Specifying and Coordinating Reliable Web Service Compositions. In: Technical Report ES-578/02, PESC/COPPE, Federal University of Rio de Janeiro. April (2002) [http://www.cos.ufrj.br/~pires/webTransact.html]
12. Pires, P.F., Raschid, L.: MedTransact: Transaction Support for Mediation with Remote Service Providers. In: Proceedings of the 3rd International Conference on Telecommunications and Electronic Commerce. Dallas, USA (2000).
13. Potts, M., Cox, B., Pope, B.: Business Transaction Protocol Primer. OASIS Committee Supporting Document. [https://www.oasis-open.org/committees/business-transactions/documents/primer/Primerhtml/BTP
14. Ramamritham, K., Chrysanthis, P. K. (eds.): Advances in Concurrency Control and Transaction Processing. IEEE Computer Society Press, CA (1997)
15. Schuldt, H., Schek, H. J., Alonso, G.: Transactional Coordination Agents for Composite Systems. In: Proceedings of the International Database Engineering and Applications Symposium (IDEAS 1999). Montreal, Canada (1999) 321–331
16. Thatte, S.: XLANG: Web Services for Business Process Design. Microsoft Corporation (2001) [http://www.gotdotnet.
17. W3C (World Wide Web Consortium) Note: Tentative Hold Protocol Part 1: White Paper. (2001) [http://www.w3.org/TR/tenthold-1/]
18. W3C (World Wide Web Consortium) Note,: Web Services Conversation Language. (2001) (WSCL) 1.0. [http://www.w3.org/TR/2002/NOTE-wscl10-20020314/]
19. W3C (World Wide Web Consortium) Note: Web Services Description Language (WSDL) 1.1. (2001) [http://www.w3.org/TR/2001/NOTE-wsdl-20010315]
20. Wiederhold, G.: Mediation in Information Systems. ACM Computing Surveys, Vol. 27 **2** 1995 265–267

NSPF: Designing a Notification Service Provider Framework for Web Services

Bahman Kalali, Paulo Alencar, and Donald Cowan

University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
School of Computer Science
Computer Systems Group
{bkalali,palencar,dcowan}@csg.uwaterloo.ca

Abstract. In this paper we extend current typical Web service architectures by providing a Notification Service Provider Framework (NSPF). Besides the three standard roles found in current frameworks (i.e., the service provider, the service requestor, and the service registry), our approach introduces an additional role that we call the service notifier. The framework is designed in four layers: the Proxy Layer, the Web Server Layer, the Application Notification Server Layer, and the Application Worker Layer. Since the NSPF itself is a service provider, this framework is reflective in the sense that it checks and notifies itself about changes. The framework is documented using design patterns. The set of patterns applied in the framework design includes the following patterns: the singleton, the delegation, the factory method, the observer, the mediator, the notifier, which is a combination of the mediator and the observer, the item description, and the proxy. The notifier pattern is in fact a publish-subscribe pattern with push semantics. The framework uses a requestor profile to support notifications related to a category of events related to changes, failures, and version control problems of Web services.

Keywords: Web Service, Event Notification, Profile, Framework, Design Patterns, Software Change, Failures, Version Control.

1 Introduction

As the Internet becomes the preferred media for businesses transactions, the number of business transactions is increasing every moment and the Internet is flooded with goods and services. For example, buyers enjoy locating the service providers and information sources globally from their desktop computers conveniently at real-time. In many cases, sellers should provide their goods and services to the buyers 24 hours a day everyday.

If a business service provider wants to implement a Web service, it first needs to find the abstract part of its description represented in a Web Service Description Language (WSDL). Once the business service provider implements the service modules related to the abstract part, these concrete descriptions are added as a second part of the WSDL file. The mechanism to publish and discover

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 73–90, 2003.

these descriptions is based on Universal Description, Discovery, and Integration (UDDI) technology.

However, UDDI only addresses the problem of finding WSDL files in a static way. If a service provider modifies its underlying Web service implementation and in turn its WSDL file, the UDDI can be useful if the service provider has updated UDDI with a link to the latest version of the WSDL file. This is due to the fact that the UDDI specification (e.g., UDDI Version 2.0) only supports very basic operations such as publishing and subscribing APIs. Service requestors who used old versions of the WSDL and are currently bound to a service provider might fail to operate if this service provider changes its WSDL content by changing its Web service implementation.

Service requestors fail to operate because they were not notified about the change of the WSDL. For example, if a company that is currently providing a service to calculate the cost of shipping a package of certain size decides to change its Web service description (WSDL), then all the partners of this company that use this service must be notified of this change. As another example, once the end-point location of a Web service provider is modified in a WSDL file, the service requestors must be notified about this change. Service providers should not only update their UDDI registries with a link to the latest version of a WSDL file, but they should also send an event notification to service requestors using a notification service about any content and version changes in the WSDL file in a dynamic way.

In this paper, we extend the common Web Service architecture that is supported by IBM [3], Microsoft [8], IONA [10], and HP [12] by designing a Notification Service Provider Framework (NSPF). Besides the three standard roles found in current frameworks (i.e., service provider, service requestor, and the service registry), our approach introduces an additional role that we call service notifier. Since the NSPF itself is a service provider, this framework is reflective in the sense that it checks and notifies itself about changes. The framework is documented using design patterns.

This paper is organized as follows. In Section 2 we provide some motivation for our work and propose a solution to the problems we have identified in current Web service architectures. Section 3 discusses background and related work. Section 4 presents a conceptual architecture of the NSPF. Section 5 presents the design of the proposed NSPF framework and describes its typical use case scenarios, the patterns applied in this design and a hierarchy of Web service events. Section 6 summarizes the paper, concluding with future work.

2 Motivation and Research Approach

2.1 Problem Statement

Web services go through the usual lifecycle of newer versions and upgrades. Users of a Web service should be aware of these changes. It is assumed that Web services requestors can adapt themselves to WSDL interface changes by downloading the latest WSDL interfaces and adapting their proxy Web services based

on the latest WSDL. However, service providers change their WSDL interface without informing service requestors.

The implication of this problem is that aggregation of Web services might fail as a result of interface mismatches. For example, in the aggregation of Web services, a new service provider Z can be created as an aggregation of service providers Y and X as shown in Figure 1. The operations of service provider Y and service provider X are hidden from the service requestor W. The service requestor W communicates only with service provider Z in this scenario. If a business process model is implemented as shown in Figure 1, then changing the Web service interface of any service provider Y and X without notifying service provider Z can result in a business transaction failure between W and Z. In other words, the service provider Z must automatically be notified about content and version changes of the WSDL of X and Y.

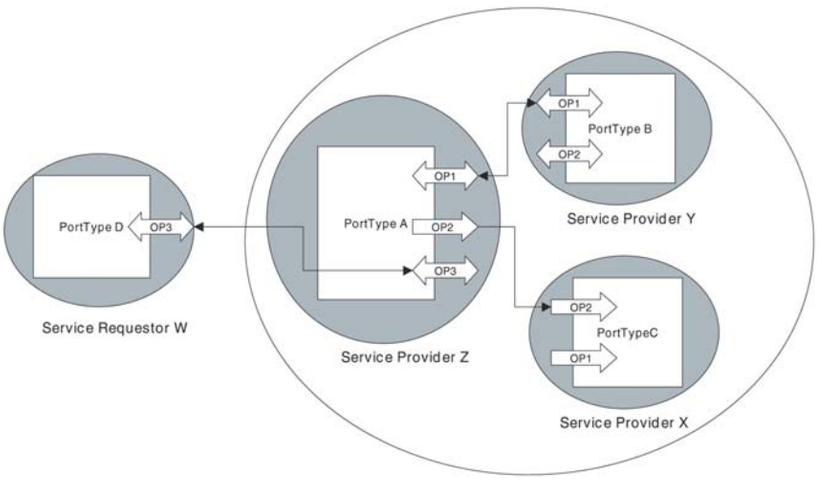


Fig. 1. Web Service Aggregation

A second problem of the current Web service architectures is that it uses the UDDI registry directly to find Web service providers. Further, anyone can have a UDDI account in public registries such as IBM [3], Microsoft [8], HP [12] and register a business without even providing services. For example, we tried to search for a business entity that provides a Currency Exchange Service. We have discovered that out of twelve businesses registered as providers of such a service, only two services were in operation and the other ten were either disabled or their access-point links were broken. In addition, if any changes happen to a WSDL interface, the UDDI will not notify users of a Web service about this change.

A third problem is related to what happens when a Web service becomes disabled. What would happen to service requestors that are currently using this

Web service? Who should inform the service requestors about possible alternative WSDL interfaces that they can use? No mechanism exists to deal with this problem so far in current Web service architectures to the best of our knowledge.

A fourth problem is related to that what happens if a new service provider offers a better service. Who should notify service requestors about better quality services offered by new service providers that offer the same services?

Finally, from a software engineering point of view, solving these problems in an ad-hoc fashion will not lead to a solution that promotes reusability, modularity and extensibility. For this reason, we provide a systematic solution to the previous problems using the concept of object-oriented frameworks [7]. In addition, the lack of appropriate framework documentation has been a significant problem that can prevent frameworks to be effectively (re-)used. However, our framework is documented using design patterns.

2.2 Proposed Solution

We propose that a new role be added to the current Web service architectural framework. We call this role the service notifier, a role that is realized by NSPF services. With this new role, a service provider sends its WSDL interface to be stored in the NSPF's XML database. The NSPF checks if a UDDI registry has a pointer to a WSDL file; in the case that the UDDI does not have such a pointer, the NSPF enforces the UDDI registry to register the WSDL file. A service requestor can search for a service provider through the NSPF. A service requestor can also be notified when any changes happen in the interface of a service provider.

Our assumption is that a service provider finds the abstract part of a Web service description and then implements a service. Once a service is implemented, the service provider fills in the second part of the Web service description (i.e., the implementation part). As a result, the service description is then completed and ready to be registered in a UDDI registry and stored in NSPF. A UDDI registry only has a link to a WSDL, but NSPF stores a copy of a Web service description into its native XML database. Each WSDL file is uniquely identified in the native XML database of NSPF.

As another aspect of the communication, a service requestor may search the UDDI registry for a service provider. Once the service requestor finds the Web service provider (i.e., a link to a WSDL file) it will subscribe its profile to the NSPF along with that WSDL. The profile of a notification consists of the type of events that a service requestor wants to be notified about. The NSPF establishes an association between the service requestor, the WSDL and Web service provider. This Web service interface will be stored in the native XML database. If the Web service provider had already stored its WSDL in the NSPF, then the NSPF establishes an association between the Web service requestor, the WSDL and service provider as well as version controlling the WSDL that is sent by the provider. In the following paragraphs we describe some features of the WNSF.

- **Notification of Changing Web Service Interface**

If the service provider changes one of its Web service interfaces, it will send a new version of the interface to the NSPF. The new service interface version will be kept through version control along with the old version in the NSPF. Therefore, the NSPF will always keep the latest Web service interface file as well as the old versions. Once the NSPF receives a new version of the Web service interface, it will notify all the Web service requestors about this new interface version.

- **Notification of Inactive Web Service Provider and Replacement**

NSPF has a copy of all the Web service interface providers who have previously stored their WSDL file in its native XML database. NSPF will then periodically bind to each service provider that has its WSDL file in the NSPF. By doing this, NSPF tries to sort out active from inactive services. This management task has two main benefits. First, if a service is inactive, the NSPF will notify all the Web service requestors who have been using this service. Second, it will try to find a replacement for the inactive service and notify all the requestors who were using this inactive service.

- **Notification of a New Available Web Service Interface**

Once a new Web service interface is produced and sent by a Web service provider to the NSPF, the NSPF will search its internal native XML database for all the requestors who have used similar service interfaces before. The NSPF will then notify all the requestors of this new Web service interface. In the case if a service requestor does not respond, the NSPF periodically tries to contact the requestor until a threshold time is reached. After expiration, the NSPF will remove the profile of the service requestor from its native XML database.

- **Notification of Best Web Service Interfaces to Web Service Requestors**

NSPF periodically searches for each service requestor in its local native database in order to find associated WSDL files of the service providers. It then reads the abstract part of WSDL file and searches for *portType* element of the abstract part. Once it finds the *portType* element name, it will search for all service providers who have the implemented part of the WSDL file corresponding to the same *portType* name. The result will be a list of all service providers that implemented the same Web service interface. Based on this list, the NSPF will select one or more service provider interfaces that have the highest number of users. All Web service requestors might be notified about these top-ranked interfaces.

3 Background and Related Work

3.1 Web Service Architecture

To identify existing problems with current Web service architectures, we have analyzed some Web service frameworks.

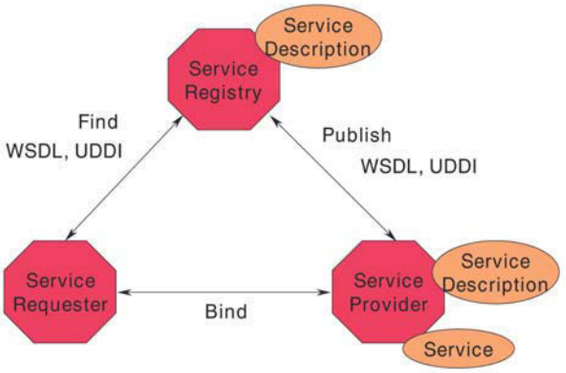


Fig. 2. Typical Web Service Architecture

A typical Web service architecture such as the IBM Web Service Conceptual Architecture [3] is based upon the interactions between three roles: the service provider, the service registry and the service requestor. The interactions between these roles are defined as publishing, finding and binding a Web service. These roles and operations act upon the Web service software module and its Web service description. In a typical scenario, a service provider implements a software module and hosts this module. Then, the service provider defines a Web service description called WSDL and publishes it to a service requestor or service registry such as UDDI registry via a UDDI operator (e.g., the Microsoft UDDI operator). The service requestor uses a find operation to retrieve the service description locally or from the service registry such as UDDI registry and uses the service description to bind with the service provider. Figure 2 illustrates these roles, operations and their interactions [3,8,10,13].

3.2 Web Service Enabling Technologies

We have also identified relevant enabling technologies in the area of Web services by analyzing Web service stacks provided by IBM [3], Microsoft [8], IONA [10], and HP [12]. Figure 3 illustrates the ia typical Web Service Stack.

The Web service standard languages and protocol applied at each layer of stack consists of the following elements:

- **SOAP**

SOAP is an extensible XML messaging protocol that provides the foundation for Web services. It is a general-purpose technology for sending messages between endpoints, and may be used for RPC or XML document transfer [1]. SOAP defines the use of XML and HTTP to access services, objects and servers in a platform-independent manner. SOAP messages are presented using XML and can be sent over any transport layer. HTTP is the most common transport layer protocol used to send SOAP messages. SOAP messages can also be transferred

over other protocols such as Simple Mail Transport Protocol (SMTP), Java Messaging Service (JMS) and IBM MQSeries. SOAP messages consist of three parts: an envelope that defines a framework for describing what is in a message, a set of encoding rules for expressing instances of application defined data types and a convention for representing remote procedure calls [1].

- **WSDL**

The Web Service Description Language (WSDL) is an XML-based language that describes Web services functionality. WSDL describes the operations a Web service can support, parameters each operation accepts and return values. It is through the service description that the service provider communicates all the specifications for invoking the Web service requestor. A basic service description is often divided into two parts: the service interface and the service implementation. A service interface definition is an abstract definition that can be instantiated and referenced by multiple service implementation definitions. Once a WSDL file is instantiated and published into a directory such as UDDI, the other SOAP clients can find the service and bind to it.

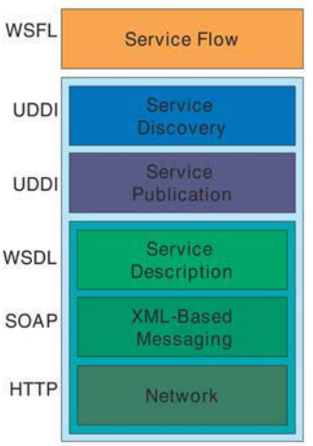


Fig. 3. A Typical Web Service Stack

One of our main goals of designing the NSPF is to address problems associated with the WSDL files and the content changes of these WSDL files. A WSDL file contains six main XML tags that describe how and where a Web service provider can be accessed. Usually a WSDL file can be automatically created by a tool that is supported by a Web service platform such as Glue [9]. We were motivated to investigate problems related to interface changes, service failures, and version control after we tried to create a simple Web service using Glue. We began by publishing the WSDL file of our service provider to a Web server and by trying to bind to this service from a client machine. To

this end, we first retrieved the WSDL file from the Web server and then we created a proxy to the Web service provider, which was running on the server machine, on the client machine. The binding operation between service provider and requestor was successful. However, when we changed the service provider interface at the server machine and, in turn, the WSDL file associated with that Web service and then tried to bind again to the same service provider on the server machine using the old WSDL file, the binding failed. We in fact changed a Java interface name on the server machine, then implemented this interface again and finally published a new WSDL file to the Web server. The binding failure was due to the fact the content of the WSDL file was changed by the service provider, but service requestor had no knowledge about that change. In this scenario, the end user of the service provider was a human and the effects of the interface mismatch were not so serious. However, these effects could be critical if a real-time application requestor used the interface.

- **UDDI**

Universal Description, Discovery, and Integration (UDDI) [2] is a standard that allows information about businesses and services to be electronically published and queried. Published information is stored into one or more UDDI registries, which can be accessed through a Web browser or via SOAP messages. UDDI provides an API for publishing and retrieving information about Web services. UDDI allows us to store and manipulate four main types of entities: *Business, Service, Binding Template* and *TModel*.

A *Business* represents an owner of Web services. It has a name, a unique key, zero or more services, an optional set of contacts, descriptions, and categories as well as identifies. The categories and identifiers can be used to specify attributes such as the business's NAICS (North Standard Products and Services Classification) and the DUNS (Data Universal Numbering System) codes, which can be very useful when performing searches.

A *Service* represents a group of one or more Web services. It has a name, a unique key, one Binding Template per Web service, an optional set of descriptions and categories and the Business that owns the service.

A *Binding Template* represents a single Web service, and contains all the information that is needed to locate and invoke the service. It has a unique key, an access point that indicates the URL of the Web service, an optional description, and a *TModel* key for each WSDL type that the Web service implements.

A *TModel* represents a concept. This concept can be concrete such as a WSDL file or abstract such as the NAICS categorization scheme. A *TModel* has a name, a unique key, an overview URL that points to data associated with the TModel and optional set of descriptions. UDDI uses *TModels* for several different proposes, but their main use is for representing WSDL interface types. By allocating a unique *TModel* to each WSDL type, UDDI allows Web services to be located based on the set of operations that they provide.

• **WSFL**

The Web Services Flow Language is an XML language for the description of Web service composition as part of a business process definition. The WSFL relies and complements existing specifications such as SOAP, WSDL and UDDI.

WSFL considers two types of Web service composition: a Flow Model and a Global Model. In the Flow model, the unit of work in WSFL is an activity and activities are defined as nodes in a linked graph. The *dataLink* and *controlLink* represent the data flow and the control flow between these activities. A *dataLink* specifies that its source activity pass data to the flow engine as part of the process instance context, which in turn has to pass this data to the target activity of the *dataLink*. Data always flows along *controlLink*. However, the *controlLink* path does not have to be direct and can comprise multiple activities. The *dataLink* enables the specification of a mapping between a source and a target document if necessary [4].

The main goal of WSFL is to enable Web services as implementations for activities of business processes. Each activity is associated with a service provider responsible for the execution of the process step. This relationship defines the association between activities which participate in the control flow and operations offered by the service provider. Activities correspond to nodes in a graph. Thus, an activity can have an input message, an output message, and multiple fault messages. This is how the message flow is specified. Each message can have multiple parts, and each part is further defined in some type system. This is the binding between the message flow and the data flow [4].

In the Global Model, WSFL provides a facility to model interactions between business partners. A global model can specify a new service provider type. This service provider type can be used in turn in a composition through a flow model or another global model. Figure 1 illustrates a Web service global model.

4 Conceptual Architecture of the NSPF

Our proposed notification service is designed to notify other service requestors about content and version changes of the WSDL files. Other purposes of the notification service include: to test the operation of a service provider, to discover similar service providers when a service provider fails to operate and notify its related service requestors, to notify service requestors about better service providers, to maintain all versions of the WSDL files, to allow service requestors to search for service providers from UDDI registries and to store service requestor profiles (i.e., the Web service profile notification - WSPN). Figure 4 illustrates the conceptual architecture of the framework.

Once a service requestor finds the WSDL of a service provider through the NSPF based on providing a set of search criteria, the NSPF will require a service requestor to subscribe for interested event types to the NSPF. This operation requires a service requestor to provide its Web Service Profile Notification (WSPN) that is stored in the NSPF's native XML database.

If the WSDL of a service provider is not stored in the NSPF's native XML database, then the NSPF will contact the UDDI registry, get a copy of the WSDL and store it into its native XML database.

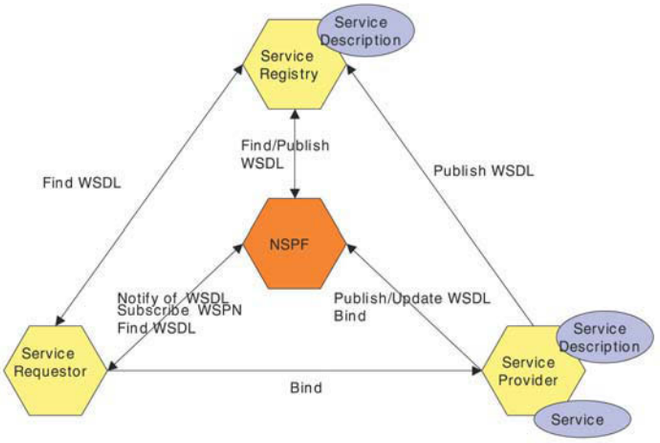


Fig. 4. NSPF Conceptural Framework

5 Design of the NSPF

To provide enhanced flexibility, performance and scalability, our proposed framework is based on a multi-tier architecture and design patterns. The NSPF is designed into four layers: the Proxy Layer, the Web Server Layer, the Application Notification Server Layer and the Application Worker Layer. Because the NSPF framework addresses mainly notifications related to Web service interfaces and their changes, and not database changes, we did not mention a fifth layer, which is a Database Layer.

Service providers and service requestors that intend to use operations provided by NSPF must first access the WSDL of the NSPF. Once they find this WSDL, they can implement a proxy to the NSPF service in order to use *findWSDL, newVersionWSDL, storeWSDL*, and *storeWSPN*. At this stage, these are the four basic operations that can be used directly by service providers and requestors.

All message exchange between the Proxy Layer and the event notification system is based on the SOAP messaging protocol. Therefore, the second layer, which we call the Web Server Layer, has *HTTPServer* and *SOAPProcessor* mechanisms. The event notification service of NSPF, which has a push semantic, is designed at the third layer -the Application Notification Server Layer. All the events are published to the event service through a *WManager* object. The *WMangager* object is also in charge of contacting its worker objects for storing the WSDL and

the WSPN files. In addition, it contacts service requestors in case events need to be notified and contacts the UDDI registry in case a WSDL file of a service provider cannot be found in the local XML database. If the UDDI registry does not have a pointer to the WSDL file, then the *WManager* makes sure the UDDI registry has such a pointer. The *WManager* also supports the version control of the WSDL files. All worker objects operate at the Application Worker Layer, which is the fourth layer. Figure 5 illustrates the design of this framework.

5.1 Use Case Scenarios

In the following use cases we describe some dynamic behaviors of the NSPF.

- **UseCase 1: Storing the First Version of WSDL to NSPF**

By assumption, a service provider that is supposed to store its WSDL to the NSPF's database first needs to find WSDL of the NSPF provider in a UDDI registry. Once a service provider finds the NSPF's WSDL, the service provider can use it to store its WSDL in the NSPF's native XML database using the *storeWSDL* operation.

The *WManager* object delegates the responsibility of storing a WSDL file into a native XML database to the *NativeXMLDatabaseWorker* object. A WSDL file is annotated with an identity (Wij, where "i" the identity of a provider and "j" is the identity of an interface). The *WManager* object also creates an *InterfaceProvider* object via the *OODatabaseWorker* object. Each *InterfaceProvider* object is stored into an OO database. Each *InterfaceProvider* object has an identity and a WSDL identity attribute. The identity of a WSDL uniquely identifies the producer of an interface.

- **UseCase 2: Storing Subsequent WSDLs to NSPF**

When a new version of the *InterfaceProvider* object is created using the *newVersionWSDL* operation, the *OODatabaseWorker* object sends a message to the OO database in order to retrieve all the *InterfaceRequstor* objects that have used previous versions of the interface. At this moment, the *OODatabaseWorker* object knows which requestors have used which providers' interfaces. The *OODatabaseWorker* object can then update the version number of the interface that is going to be used by a service requestor. Once the *OODatabaseWorker* object completes its operation, it will inform the *WManager* obect to notify requestors about the new version of the *InterfaceProvider* object. The *WManager* object will contact the *NativeXMLDatabaseWorker* object in order to get the WSPNs of the requestors. Once the *WManager* receives these WSPNs, it will forward them to the *WSDLPublisher* object. The *WSDLPublisher* object will then publish the *InterfaceChangeEvent* object to the Event Service. The *WSDLPublisher* only publishes this event to the Event Service if the requestor had registered for this type of notification in its WSPN.

Example:

Assume that the service providers P1 and P2 have previously produced the interfaces (W1, W12, W13) and (W21, W22) respectively:

$$\text{P1} \Longrightarrow \text{(produced) (W11, W12, W13)}$$
$$\text{P2} \Longrightarrow \text{(produced) (W21, W22)}$$

Therefore, we have five objects of type *InterfaceProvider* stored into an OO database via the *OODatabaseWorker* object as follows: PO1 [P1, W11]; PO2 [P1,W12]; PO3 [P1,W13]; PO4 [P2,W21]; PO5 [P2,W22].

Then, assume that the service requestors R4, R5 and R6 have previously used interfaces W11, W12 and W21 respectively:

$$\text{R4} \Longrightarrow \text{(used) PO1 [P1, W11]}$$
$$\text{R5} \Longrightarrow \text{(used) PO1 [P1, W12]}$$
$$\text{R6} \Longrightarrow \text{(used) PO2 [P2, W21]}$$

This means we have three *InterfaceRequstor* objects in the OO Database as follows:

RO1 [R4, PO1 [P1, W11]]
RO2 [R5, PO1 [P1, W12]]
RO3 [R6, PO2 [P2, W21]]

Later, let's say a new interface is produced by P1 (i.e., W14). In deed, this is the forth WSDL file that is produced by P1 and submitted to the NSPF.

$$\text{P1} \Longrightarrow \text{(produced) (W14)}$$

After the WSDL file is stored into the native XML database of the NSPF, an *InterfaceProvider* object is created (i.e., PO6 [P1, W14] and stored into the OO database. Consequently, the *OODatabaseWorker* object sends a query message to the OO database that provides the identity of the service provider (i.e., P1). The *OODatabaseWorker* object will receive the following results:

RO1 [R4, PO1 [P1, W11]]
RO2 [R5, PO1 [P1, W12]]

In this way, the *OODatabaseWorker* object knows that R4 and R5 have used W11, the interface produced by P1. This *OODatabaseWorker* object also knows that P1 has just produced a new interface (i.e., W14).

Thus, the *OODatabaseWorker* object sends a message to the *WManager* object to notify R4 and R5 about W14 that is produced by P1. The *WManager* object will forward the message to the *NativeXMLDatabaseWorker* object in order to get the WSPN of R4 and R5. Once it receives the profiles of R4 (i.e.,

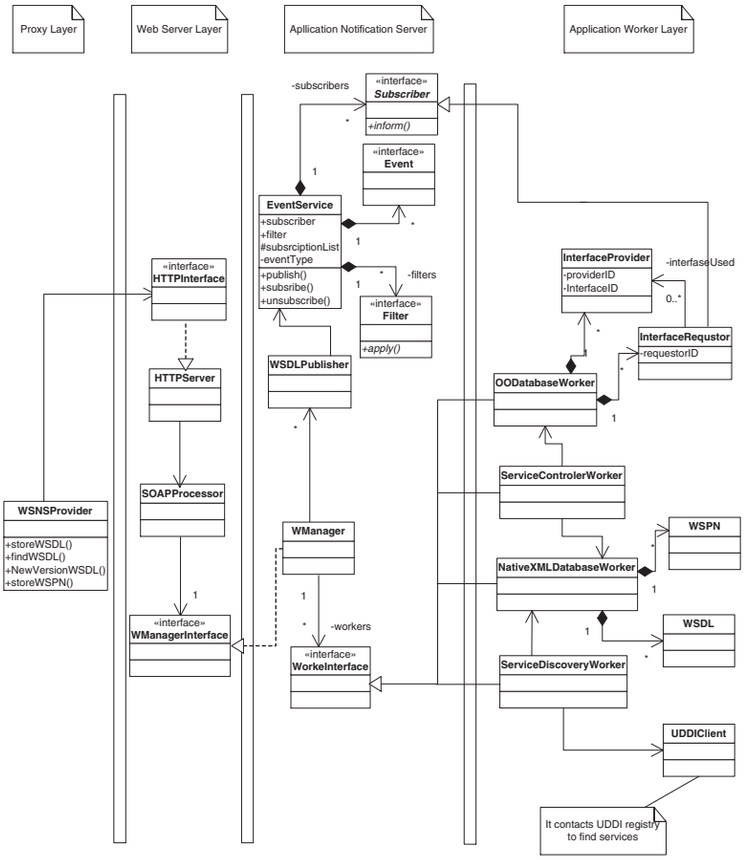


Fig. 5. Design of the NSPF

WSPN4) and R5 (i.e., WSPN5), it will send these profiles to the *WSDLPublisher* to publish the *InterfaceChangeEvent* to the Event Service. Finally, the Event Service will notify R4 and R5 of this interface change and the *DatabaseWorker* object will update the attributes of requestor objects with the latest interface version identity.

$$\text{RO1 [R4, PO1 [P1, W11]]} \Longrightarrow \text{RO1 [R4, PO1 [P1, W14]]}$$
$$\text{RO2 [R5, PO1 [P1, W12]]} \Longrightarrow \text{RO2 [R5, PO1 [P1, W14]]}$$

- **UseCase 3: Checking for an Inactive Service**

The *WManager* object periodically sends a message to its *ServiceControler-Worker* object to check whether a service is active or not. The *ServiceControler-Worker* object contacts *OODatabaseWorker* object to get an *InterfaceProvider* object whose interface identity is the latest one in the OO database. Once the

ServiceControlerWorker object receives the *InterfaceProvider* object, it will send its identity to the *NativeXMLDatabaseWorker* object in order to get the WSDL of the interface provider. The *NativeXMLDatabaseWorker* object sends back the WSDL file to the *ServiceControlerWorker* object. The *ServiceControlerWorker* object uses the WSDL file to contact the service provider in order to test each operation that is supported in the WSDL file. If all the operations that are being tested terminate successfully, then the *WManager* object will receive a message notifying about this success; otherwise, the *ServiceControlerWorker* object contacts *NativeXMLDatabaseWorker* object in order to get the WSPN of all the requestors that were using this WSDL and pass them back to the *WManager* object. Once the *WManager* object receives the WSPN, it will forward it to *WSDLPublisher* object. The *WSDLPublisher* object will then publish the *ServiceEnactiveEvent* object to Event Service. The *WSDLPublisher* object only publishes this event to the Event Service if the requestor had registered for this type of notification in its WSPN.

5.2 Patterns Applied to the Design of NSPF

We have applied the following design patterns in the design of the NSPF: the singleton, the delegation, the factory method, the observer, the mediator, the notifier, which is a combination of the mediator and the observer, the item description, and the proxy. Figure 6 illustrates how the interaction between two adjacent layers of the NSPF can be realized through design patterns.

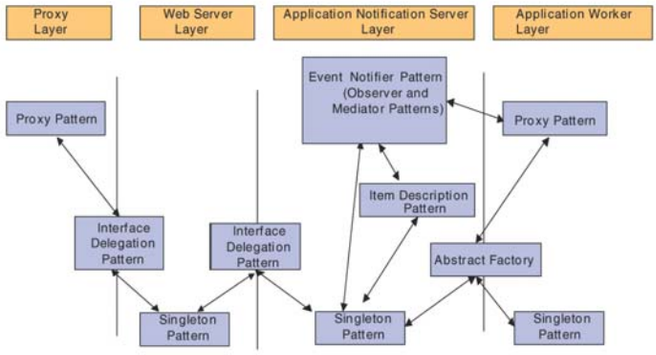


Fig. 6. Design patterns of NSPF and their interaction

- **Singleton Pattern**

This singleton pattern [6,11,13] ensures a class only has one instance, and provides a global point of access to it. A singleton class can ensure that no other instance of the class can be created. It can provide a way to access the instance

of the class. We have used this pattern at the Web Server Layer, the Application Notification Server Layer and at the Application Worker Layer within the NSPF.

• **Delegation Pattern**
The delegation pattern [6,11,13] allows one object to delegate responsibility of performing one task to another object through an interface. In designing the NSPF, the delegation pattern is used between the Proxy Layer and the Web Server Layer as well as between the Web Server Layer and the Application Server Layer.

• **Factory Method Pattern**
The factory pattern [6,11] is used when it is necessary to create one of several polymorphism objects until runtime. The factory method pattern lets subclasses to decide which factory to use. The decision can be made by subclasses of a base class. We have used this pattern to realize the interaction between the Application Notification Layer and Application Worker Layer in the NSPF.

• **Observer Pattern**
The observer pattern [5,6,11,13] allows observer object to watch the subject object. The observer pattern allows the subject and observer to form a publish-subscribe relationship. Through the observer pattern, observer objects can register to receive events from the subject object. When the subject object needs to inform its observer objects of an event, it simply sends the event to each observer object. In designing the notification service of the NSPF, we have used a combination of this pattern with the mediator pattern.

• **Mediator Pattern**
The mediator pattern [5,6,11,13], which is a behavioral pattern, allows modeling a class whose object at run-time is responsible for controlling and coordinating the interactions of a group of other objects. The mediator pattern helps encapsulate collective of behaviors in a separate mediator object. The mediator pattern promotes a loosely coupled interaction between the objects by keeping these objects from referencing one another explicitly. This pattern along with the observer pattern has been used in the notifier pattern, which is described next [5,6,11,13].

• **Notifier Pattern**
The notifier pattern [5,6,11,13] is the combination of the mediator pattern and the observer pattern used to design a publish/subscribe notification service. In the NSPF, the Event Service, as shown in Figure 5, mediates notification so that *WSDLPublisher* objects and *InterfaceRequestor* objects do not need to know about each other. In addition, the Event Service allows us to add or remove subscribers (*InterfaceRequstor* objects) dynamically. In contrast with the observer approach, the addition or removal of subscribers is centralized in the Event Service.

• **Item Description Pattern**
The purpose of this pattern is to put instance variables of a class into a separate class descriptor [6,11,13]. In designing the Web Service Hierarchy of Events, we

have used the item description pattern. In our case, the Event class is an Item and the Property is the ItemDescriptor. Figure 7 illustrates this pattern.

- **Proxy Pattern**

The proxy pattern [6,11,13] describes how to provide a level of indirection access to an object that resides in another expectable environment. In the Web service architecture, a proxy Web service object acts as an interface between the Service requestor and Service providers. We have applied the proxy design pattern at the Proxy Layer of NSPF since service providers and service requestors should first retrieve the WSDL of the NSPF from a UDDI registry and then create a proxy object to the NSPF service provider.

5.3 Web Service Hierarchy of Events

Each service requestor can subscribe to an event type that is going to be published by the service providers to the event service of the NSPF. A natural way of organizing events in object-oriented paradigm is by using inheritance to have a hierarchy of events sharing a common interface or set of attributes on all events, as explained in [5]. For example, the time of occurrence of an event is relevant to all events and therefore should be presented in a base class. Another advantage of such hierarchical organization is for a subscriber to be able to register interest in an event sub-tree and automatically inherit from all the events in that sub-tree [5]. Figure 7 illustrates the hierarchy of events that we adopted for the event type of the NSPF.

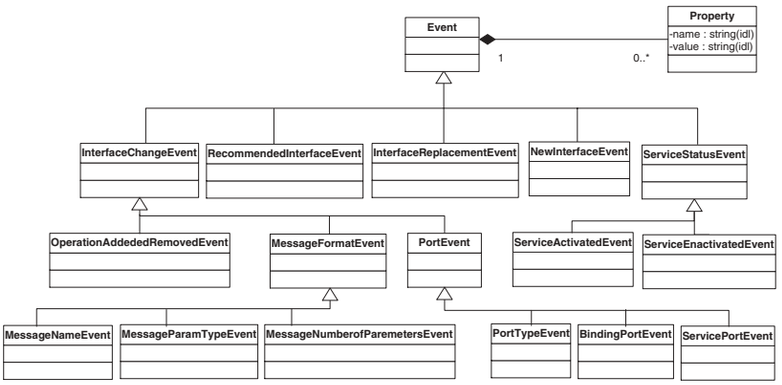


Fig. 7. Web Service Hierarchy of Events

6 Conclusions and Future Work

In this paper we have described an extension of current typical Web service architectures that provides a Notification Service Provider Framework (NSPF).

Besides the three standard roles found in current frameworks (i.e., the service provider, the service requestor, and the service registry), our approach introduces an additional role that we call the service notifier. We are working on a prototype that incorporates these techniques as a platform for experimentation, demonstration, and requirements elicitation.

The framework is designed in four layers: the Proxy Layer, the Web Server Layer, the Application Notification Server Layer, and the Application Worker Layer. Since the NSPF itself is a service provider, this framework is reflective in the sense that it checks and notifies itself about changes. The framework is documented using design patterns. The set of patterns applied in the framework design includes the following patterns the singleton, the delegation, the factory method, the observer, the mediator, the notifier, which is a combination of the mediator and the observer, the item description, and the proxy. The notifier pattern is in fact a publish-subscribe pattern with push semantics [5,6,11,13]. However, other patterns may be included in the framework design and documentation as we experiment and proceed with the framework development. So far, we have only applied object-oriented design patterns in our approach, but we exploring the possibility of having specific service-oriented patterns in our design.

The framework uses a requestor profile (WSPN) to support notifications related to a category of events related to changes, failures, and version control problems of Web services. We are experimenting with these different types of events and their impact in the notification process. In any case, although not complete in any sense, the event set we are currently using constitutes a representative set related to practical circumstances revealed by our conceptual analysis of the Web service application. We are currently working on a more elaborated set of event types and on defining a XML schema for this set. We also plan to improve the design of NSPF's version control features. In addition, we need to improve our design solution to address problems related to the scalability of distributed Native XML databases.

Finally, by providing a more dynamic perspective to Web service architectures based on the introduction of a service notifier, our approach constitutes both research and practical advances in the design, implementation, and maintenance of Web service applications.

References

1. W3C Soap Version 1.2, W3C Working Draft 17, December 2001, http://www.w3.org/TR/2001/WD-soap12-part0-20011217/
2. UDDI. The UDDI Technical White Paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.doc
3. Web Service Conceptual Architecture (WSCA 1.0), IBM Technical White Paper, May 2001, http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
4. Web Services Flow Language (WSFL 1.0), IBM Technical White Paper, May 2001, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

5. Suchitra Gupta, Jeffrey M. Hartkopf, Suresh Ramaswamy, "Event Notifier: A Pattern for event Notification," in Java Report Magazine, July 1998.
6. Wolfgang Pree, "Design Patterns for Object-Oriented Software Development," Addison-Wesley, 1997.
7. Mohamed E. Fayad, Ralph E. Johnson, "Domain-Specific Application Frameworks," Wiley Computer Publishing, 2000.
8. Web Service Description and Discovery Using UDDI, Microsoft Corporation Technical White Paper, October 2001.
9. Glass, Graham, "Web Services – Building Blocks for Distributed Systems," Prentice Hall, 2002.
10. IONA Technologies Ltd.: Orbix Programming Guide – Release 3.2, November 1997.
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.
12. HP Web services Platform Architecture – An Overview, http://www.hpmiddleware.com/downloads/pdf/web_services_architecture.pdf
13. Buschmann, F., Meunier, R., Rohnert, H, Sommerlad, P., Stal, M., "Pattern-Oriented Software Architecture," John Wiley and Sons, August 1996.

Active UDDI – An Extension to UDDI for Dynamic and Fault-Tolerant Service Invocation

M. Jeckle and B. Zengler

DaimlerChrysler Research and Technology
Research Information and Communication / Data and Product Management
Ulm, Germany
{mario.jeckle,barbara.zengler}@daimlerchrysler.com

Abstract. UDDI, Universal Description, Discovery, and Integration, represents a directory for the publication and querying of categorized Web services. Publication and query are performed by utilizing UDDI's Application programming interface (API), which employs SOAP as a communication instrument.

By offering an invocation API in addition to two other types for examination, UDDI allows clients to search for and subsequently invoke specific Web services. Failures in invoking already sought and, on the application side, statically cached Web services typically result in re-querying the registry.

However, an application's reaction time in response to changes is limited due to UDDI's replication latency, i.e. the amount of time it takes for changes to entries stored inside the UDDI repository to be propagated to all UDDI nodes.

This paper proposes a mechanism termed *active UDDI*, which allows the extension of UDDI's invocation API in order to enable fault-tolerant and dynamic service invocation.

1 Introduction

UDDI represents a directory for the publication and discovery and location of categorized Web services. The UDDI specifications [IIea01] published by an industry initiative[1] introduce *data structures*, *API specifications* as well as a directory-specific *replication process* and an *operator's specification*. The abstract notion of a *UDDI directory* is implemented by an application termed *UDDI registry*, which is comprised of several data-holding entities called the *UDDI nodes*. All of these nodes which additionally allow publication of service-related information are further rubricated *UDDI operator nodes*.

UDDI offers a set of application programming interfaces (APIs), which can be used to publish or search information stored within the directory.

[1] including Microsoft, SAP, Intel, HP, Compaq, SUN, and Verisign

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 91–99, 2003.

This API defines interfaces utilizing W3C's *XML protocol/SOAP*[GHM+02][2] as a communication instrument. Therefore, the UDDI directory as a whole can be regarded as a publicly available Web service.

Clients desiring to publish or retrieve information send SOAP messages to one of the nodes. After processing, the node sends back the result to the client as a SOAP message. If the processing results in an error, the node uses SOAP to send back a fault message indicating the type of error it encountered.

Queries, search results, and error reports directed to and emitted by the UDDI repository are formulated in a language conforming to W3C's *Extensible Markup Language* (XML) [BPSMM00]. The definition for this language and the UDDI data structures are given in several XML schema files.

The *UDDI API* provides interfaces for three patterns of request: the *browse pattern*, the *drill-down pattern*, and the *inquiry pattern*.

Shared semantics of these patterns is that they solely enable different ways of accessing the data stored inside the nodes of a repository but do not permit restriction of the services queried by other characteristics as defined by the service provider when initially publishing the service to a UDDI node.

Though providing basic support for remote service invocation, UDDI does not support dynamic service invocation within a network of distributed services. Networks based on Internet technology are subject to diverse changes resulting from the network protocol design. Thus, applications using Internet technologies to fulfill their functionality need to be capable of reacting to incessant and immediate changes such as network partitioning and, as a result, temporary unavailability of parts of the network where services reside. For this reason, the network itself virtually stores information about the dynamic state of a service.

A repository capable of dynamic reaction to errors should be able to detect changes in the availability of a utilized service or at least react to it. Additionally, the repository should be capable of transparently replacing the - most likely only temporarily - unavailable service by an alternative one that fulfills the same task. This behavior is referred to as *dynamic service invocation* within this paper.

With dynamic service invocation only one aspect of fault tolerance, UDDI does not at all support handling mechanisms for fault situations commonly encountered in distributed systems[3]. Applications, however, should be supported by the repository in coping with such situations in order to maintain a uniform level of quality of service.

Active UDDI as proposed in this paper is an extension to UDDI, which enables applications to dynamically react to fault situations as outlined in this paper using the example of network error situations.

[2] Originally SOAP served as abbreviation for *Simple Object Access Protocol*, but since W3C took on responsibility for the standardization this is no longer the case.

[3] Such as a decrease in quality of service which could be caused by an increase in response time or repeated erroneous data transfers.

1.1 The UDDI Data Model

For a basic understanding of how UDDI works, a short overview of the data structures utilized in registering information is provided in the following.

UDDI distinguishes five core data structures: *businessEntity*, *businessService*, *bindingTemplate*, *tModel*[4] and *publisherAssertion*.
BusinessEntity embodies the known information about a registered body[5], including descriptive information such as the body's name, contact information for human contacts, both the publisher's name and the custodian's name as well as descriptive and technical information about the *businessServices* it offers.

It is possible to supplement additional body identification information such as tax identifiers to the *businessEntity* or categorizing information such as standardized industry, product or geographical codes. Also, these standardized codes may be validated by UDDI operator nodes.

The *businessService* structure is contained in a businessServices structure and aggregates descriptive information about a specific Web service such as the service's name, description, and its *bindingTemplate*. As with the *businessEntity*, additional categorizing information such as standardized industry, product or geographical codes may be appended. *businessService*s may be utilized as a projection, i.e. an already published service can be reused or shared within several *businessEntity* elements by including the service reference in the *businessEntity*.

*bindingTemplate*s contain information for defining the technical entry point for a specific Web service and describing service-specific technical characteristics including parameter- specific settings. For instance, the information contained may consist of a descriptive text, the service parent's key which relates the bindingTemplate to its providing businessService and either an indication for the service access location or a reference to a different bindingTemplate describing the service in more detail[6]. Additionally, a technical fingerprint[7].

These sets of tModel references can be searched for by an interested party. The search result includes solely those bindingTemplates that match the specific fingerprint. These fingerprints may additionally be registered within the UDDI registry of a service and can be contained within a *bindingTemplate*.

tModel data structures represent abstract structures for the description of data. Consisting of a unique key, a name, and an included and/or external description, they allow not only the specification of concepts but also a technical description of Web services. Therefore, the UDDI specification defines two application domains for tModels, namely compatibility checks utilizing technical fingerprints and namespace references. With the key of a tModel representing its technical fingerprint or namespace reference, a publisher is able to specify

[4] technicalModel
[5] This could be an organization, a company or any other kind of service-providing entity.
[6] This allows the remote hosting of services or a grouping of service descriptions
[7] This technical fingerprint is represented by the combination of the tModel references contained in the bindingTemplate within a *tModelInstanceDetails* element

compliance to a concept by including a reference to the specific tModel in an application. Consequently, client applications may then search for Web services compatible to the specific concept[8].

A *publisherAssertion* structure allows the declaration of relationships between businessEntity structures. Large organizations may publish several businessEntities which represent their subordinate companies. The validity of business relationships has to be agreed on by both the parties involved: both have to mutually define the relationship. A publisherAssertion consists of the keys of the related bodies and a reference to the tModel representing the relationship.

Every UDDI data structure contains its own unique identifier in terms of a UUID as specified in [ISO96]. The three data structures *businessEntity*, *businessService*, and *bindingTemplate* are related to each other via a containment relationship. A *businessEntity* may contain one or more *businessService*s, which again may contain one or more *bindingTemplate*s.

UDDI distinguishes the relations *containment* and *reference*. Whereas a containment implies that a specific data instance may be contained by only one parent at a specific point in time, references may occur in several places simultaneously. References are typically found in lists which are not individual instances themselves. Any key values directly contained in structures that are not one of the five core structure types themselves are references. For example, the *bindingTemplate* references *tModel* data structures, thus allowing a specific tModel instance to be included in several bindingTemplates at a specific point in time.

1.2 The UDDI Replication Process

Publicly available UDDI nodes, similar to the DNS system, together form a service that, while appearing to be virtually a single component, is composed of an arbitrary number of operator nodes. They are called the *UDDI cloud*. An operator node is responsible for the data published at this node: in UDDI terms, it is the custodian of that part of the data.

UDDI data must be kept consistent between the nodes of a UDDI cloud. This goal is achieved by the application of *UDDI's data replication process*[9] within the participating nodes. Furthermore, the UDDI replication process is the only available mechanism allowing cloud- participating nodes to communicate data changes. The process utilizes XML and SOAP for inter-node communication. With regard to the development of applications using UDDI, the possibilities of an application implementing the functionality of a whole UDDI registry or parts of it[10] as well as an application solely acting as a UDDI client must be equally considered.

[8] Applications must be aware of the relation between the abstract concept and its representing tModelKey in order to utilize this mechanism.

[9] The data replication process is specified within the *UDDI Replication Specification* [IIea01].

[10] i.e. participating within the cloud in the role of an operator node

For replication purposes, *change records*[11] consisting of the originating node's UUID, its *update sequence number*[12] and the data payload are transferred between UDDI nodes with each operator node recording the transferred and processed messages.

Each node's replication state is represented by a *highWaterMarkVector* containing the node's identity in terms of an *operatorNodeID* and information about the latest change in terms of an update sequence number. The highWaterMarkVector consists of the same semantic information as the ChangeID in the change record. The UDDI specification does not mention why there are two structures which carry the same information.

Replication is configured in a cloud-wide master configuration file in XML format containing a serial number[13], the time of the last update, contact information, and replication time configuration. Replication time is defined in two intervals in the unit *hour*:

- The interval defining the assumed time of change visibility at all nodes.
- The interval defining the maximum time an individual is to wait before utilizing change requests.

Additionally, the configuration file aggregates information about participating operator nodes including each node's target replication URL *soapReplicationURL* indicating a secure HTTP connection. Therefore, nodes need to be able to act as TLS1.0 servers and clients[14] with at least RC2 and RC4 40-bit encryption and MD5 message authentication algorithms. Replication process message flow is driven by a global communication graph specified within the configuration file. The graph configuration element lists both the participating nodes by their IDs and the message types that are to be controlled. Fallback edges may be installed.

UDDI v2 offers a mechanism to announce the availability of changes utilizing the *notify_changeRecordsAvailable* element. A node employing this mechanism informs other nodes about the availability of changes, providing its ID together with its own replication state and other nodes' states known within the *notify_changeRecordsAvailable* message. Application of this notification instrument can reduce the retrieval time the polling mechanism would take. Nodes receiving such a message are subsequently required to ask for changes within the time interval specified in the configuration file. Implementation of this notification mechanism is purely optional.

This mechanism is an approach toward solving our problem; yet since nodes are required to ask for changes within an interval that can only be defined in the

[11] A change record structure is transmitted between nodes to indicate the kind of change that is to be processed together with the data to be changed.

[12] An increasing sequence number which is generated locally at every node, indicating the most recent change

[13] Similar to the serial number of DNS SOA records, the number is incremented with every update.

[14] Each operator must therefore acquire an X.501 certificate.

unit hour, it is not applicable for highly dynamic change information, e.g. when a network temporarily partitions.
Replication is initiated at the UDDI nodes by the caller, i.e. the receiving node, submitting a `get_changeRecords` request message. Within this message type, nodes provide their current highWaterMarkVectors to communicate their state of replication to other participating nodes. In addition, nodes may try to limit the number of messages they wish to receive.
To accommodate bug detection and processing, the UDDI Replication Specification offers a process that helps in detecting, dealing with, and managing the following replication errors:

- Invalid record validation: A receiving node's UDDI implementation cannot validate the incoming change request due to erroneous validation implementation (such as overly aggressive checking).
- Invalid interim representation: An intermediary node inaccurately handles the change record.
- Invalid generation: The originating node poorly generates the change record.

2 Active UDDI

2.1 Basic Idea and Background

The several UDDI processes and mechanisms discussed in the above cover solely operational aspects of the UDDI cloud, data management, and replication aspects. They are designed and suitable for dealing with explicitly published changes to the registry data, which are typically done by operators or publishers[15].

While these processes can be regarded as an approach to automatically handle changes in the registry, they do not represent a solution for the problem of dynamic service invocation or fault tolerance. This is illustrated in more detail in the following paragraphs.

Management of changes to information stored in one of the UDDI nodes participating can be slightly automated by utilizing the replication processes presented above within a UDDI registry.

Controlled by the *global* configuration file, the registry nodes replicate regularly, thus automatically propagating changes within the registry. This data replication is a process of long duration: the replication configuration allows specification of replication time intervals in hours. This time-consuming endeavor makes it poorly suited for real-time dynamic service invocation.

However, UDDI replication covers exclusively those changes that were explicitly published at the registry. Types of changes covered are the registration of new data, changes in the information registered within the data structures, data deletion, and changes in custody for a given datum.

[15] Operators can allow third parties to publish UDDI data. Therefore, these publishers are explicitly registered by the UDDI operators. They may also be assigned a limited space of unique keys for information registration purposes.

All these changes necessitate publisher intervention since they must be explicitly declared by a UDDI operator or publisher. This could be done automatically by the use of automated facilities for submitting change requests on the publisher or operator sides. Within the UDDI process, publishers and operators are merely capable of changing data they originally published, i.e. data they have the custody for. For example, if the change in availability for a service, i.e. the service has become temporarily unavailable, is to be reported to a UDDI registry, this cannot be done by a party which detects the unavailability, even if this is a trusted party.

Yet the realization of a dynamic infrastructure for applications demands just these aspects. The need for dynamic reaction to immediate changes such as the temporary unavailability of a service must be met, especially in a network environment where Web services are likely to be reside. Applications relying on the functionality of distributed Web services must be able to collaborate dynamically. Therefore, pivotal aspects such as fault tolerance and dynamic reaction must be considered in addition to the basic replication mechanism as provided by UDDI. Applications must be supported in implementing aspects of fault tolerance. Trusted parties[16] must be able to report errors. These errors should be recorded by the registry and taken into consideration when information about an erroneous service is provided.

2.2 Active UDDI as an Extension to UDDI

For the further discussion of active UDDI, it is assumed that the Web services participating in the service network are interwoven and interdependently utilize each other.

Active UDDI's basic approach is an extension of the existing UDDI infrastructure without requiring changes to the data structures or the APIs presented above. Since a UDDI registry represents itself to the outside as an invocable Web service, active UDDI merely adds an additional Web service, subsequently referred to as the active service, to a UDDI registry which serves as a single entrance point for both the inquiry and publishing APIs within the participating network of Web services, the clients. The extension approach leaves the existing UDDI registry untouched: it is not necessary to change existing data structures. Additionally, no changes need be made to the UDDI APIs as additional interfaces allowing services to use its functionality are provided instead. Thus, the active UDDI extension can be applied to any existing UDDI registry.

Active UDDI's functionality is as follows:
The *active service* locally mirrors the UDDI registry's data pool and implements the UDDI API calls. It can therefore be described as a proxy located between the clients and the UDDI registry. While it participates in the UDDI replication process, it does not have custody of any portion of registration data. However,

[16] The term trusted parties here explicitly includes Web services. Trust is provided either by authentication means or in terms of a distributed consensus.

it implements and offers the additional features necessary to allow real-time dynamic service invocation and fault tolerance.

Mirroring the UDDI registry data, the active service maintains a list of available Web services, which is a subset of the services stored within the UDDI registry. Also, the active service is able to process unavailability (or availability as the case may be) messages originating from its clients indicating that a service is temporarily unavailable. Hence, active UDDI introduces state information about the various Web services. Through continuous maintenance of this state information, the active service is able to provide a statement about the current availability of a service.

Upon reception of an unavailability message, the active service blinds out the affected Web service from its list of available Web services: it is temporarily not included in the node's responses to the inquiry API. When the service becomes available again, it is faded in and thus included in the node's responses.

The temporary changes which are recognized by the active service do not propagate throughout the UDDI cloud. The active service locally registers these changes and, in its role as proxy, prevents the affected services from being reported in response to a client's queries. The original data which is held at the UDDI cloud is not impacted by the active service's operation. The active service does, however, accept change queries by clients but forwards them to the node in custody of the specific data portion.

As mentioned before, unavailability and re-availability are indicated by the participating clients by transmitting an appropriate SOAP message to the active service. However, not just any arbitrary Web service is permitted to announce a change in the state of a particular Web service. An active UDDI operator has to set up a policy that specifies when announcements are to be accepted. And this policy needs to consider authentication. An active service may allow specific, trustworthy clients to report changes in the state of another client. Therefore, the reporting client may authenticate itself to the server by the use of a digital certificate which clearly reveals its identity.

Additionally, the active service may require its clients to perform a distributed consensus about the change in state of a specific client. The application of a policy for a distributed consensus solves the problem of network partitioning. While a Web service may be unavailable for a particular client service due to network problems at the client's side, it might well remain available for other client services. Marking the service as unavailable merely because a particular client reports it to be unavailable would be intolerable for some application scenarios.

If a service is marked as erroneous, the active service is able to determine alternate services which offer the same functionality. Services are grouped according to their functionalities by comparing their WSDL descriptions. Thus, the active service can offer an alternative, i.e. a service with the same functionality belonging to the same group, which the client may invoke in order to fulfill its task. Furthermore, the active service allows its clients to participate in a publish-subscribe mechanism if desiring to be informed of changes affecting a Web service or of a category of Web services they wish to monitor. Clients can in-

form the active service about Web services or categories they wish to track. The active service keeps a list of clients and the changes to be tracked. When a listed service or category is affected by either a temporary or permanent change[17] the active service sends the clients an update message indicating the change that occurred.

3 Conclusion

The concept of active UDDI proposed in this paper implements a proxy that is able to actively react to changes and thus supports applications which make use of distributed Web services. It is an approach geared to meeting the requirements of fault tolerance in distributed systems based on Web service technology and is illustrated using the example of how temporary service unavailability is handled. Additionally, the concept offers implementation of a publish-subscribe mechanism. The proposed UDDI extension can be applied to any UDDI implementation by adding an additional Web service that implements its functionality to the existing infrastructure. The benefit is that this infrastructure does not need to be changed.

References

[BPSMM00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler, editors. *Extensible Markup Language (XML) 1.0 (Second Edition).* World Wide Web Consortium, W3C, October 2000. http://www.w3.org/TR/2000/REC-xml-20001006/.

[GHM+02] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, editors. *SOAP Version 1.2 W3C Last Call Working Draft.* World Wide Web Consortium, W3C, June 2002. http://www.w3.org/TR/2002/WD-soap12-part1-20020626/, http://www.w3.org/TR/2002/WD-soap12-part2-20020626/.

[IIea01] IBM, Intel, and Microsoft et al., editors. *UDDI Version 2.0 Specifications.* UDDI.org, June 2001. http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf, http://www.uddi.org/pubs/Operators-V2.00-Open-20010608.pdf, http://www.uddi.org/pubs/Replication-V2.00-Open-20010608.pdf, http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf.

[ISO96] ISO, editor. *ISO/IEC 11578: Information technology - Open Systems Interconnection – Remote Procedure Call (RPC).* ISO, Geneva, CH, 1996.

[17] The active service detects this either by receiving a message from its clients or from changes within the replication process

WS-Specification: Specifying Web Services Using UDDI Improvements

Sven Overhage and Peter Thomas

Darmstadt University of Technology,
Dept. of Application Engineering and
Business Information Systems
Hochschulstr. 1, 64289 Darmstadt, Germany
{overhage, thomas}@bwl.tu-darmstadt.de

Abstract. Web services are interoperable components that can be used in application-integration and component-based application development. In so doing, the appropriate specification of Web services, as the basis for discovery and configuration, becomes a critical success factor. This paper analyses the UDDI specification framework, which is part of the emerging Web service architecture, and proposes a variety of improvements referring both to the provided information and the appropriate formal notations. This leads to a more sophisticated specification framework that is called WS-Specification and provides information referring to different perspectives on Web services. It considers Web service acquisition, architecture, security, performance, conceptual concepts and processes, interface definitions, assertions, and method coordination. WS-Specification thereby maintains backward-compatibility to UDDI and is ordered using a thematic grouping that consists of white, yellow, blue, and green pages.

1 Introduction

Web services promise to mark a step into the new era of application-integration and component-based application development, which was once prophesied to end the so-called software crisis [26] and has yet to come. Principally, both application-integration and component-based application development are based on the same fundamental paradigm (see figure 1): new applications are being developed by browsing component catalogues (repositories), discovering appropriate components, and configuring them [32].

Components (strictly speaking one should say software components) are reusable pieces of software each consisting of different artefacts (e.g. documentation, executable binaries, tests etc.). They are self-contained and combinable with other components, offer their services using one or more previously defined interfaces, and hide their implementation (black-box reuse) [1], [14], [32]. Components can both be fine-grained (elementary) and coarse-grained (compound), thus allowing it to even look upon applications as components, e.g. in an application-integration scenario.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 100–119, 2003.

Although component-based application development (and application-integration, too) contributes a lot of advantages to the field of software engineering, it encountered many obstacles that prevented a break-through up to now. Among those, especially missing specification standards and the management of (unrecognized) heterogeneities between components turned out to be main issues. Heterogeneity reduces the compatibility between components so that an adapter has to be designed in order to enable interaction between them. Appropriate component specifications are essential both in order to assess the applicability of components during discovery ("find the right components") and as a basis for correctly configuring them (the latter includes identifying and mitigating possible heterogeneities based on the specification).

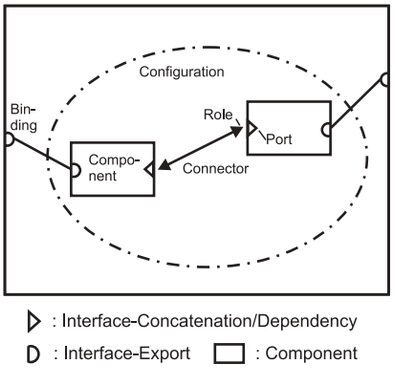


Fig. 1. Conceptual model of component-based application development (based on [31]): Components are configured using connectors. Configurations can be viewed as (compound) components.

The need to avoid heterogeneities and to establish common formats for component specifications is (partially) considered by the emerging Web service architecture [6], which introduces standards that aim at designing platform-independent connectors[1], interoperable (compatible) components, and catalogues containing standardized component specifications. The Web service connector technology is based on established internet-based interaction protocols (like TCP/IP and HTTP) and uses XML as the format for information exchange. Upon those, SOAP (Simple Object Access Protocol [20], [21], [29]) defines a platform-independent remote procedure call for interactions between components. The WSDL (Web Service Description Language [12]) introduces an XML based format to specify the interface(s) of a component that offers its services via SOAP. Ad-

[1] Connectors embody interactions between components and define paths of interaction usually based on interaction protocols [31]. In addition, they eventually contain adapters in order to manage heterogeneities between components.

ditional standards to improve the applicability of the architecture are likely to follow (e.g. WS-Security, WS-Coordination etc. [28]).

Components that make use of these standards are called (XML) Web services. They are (technically) interoperable and can be specified (and catalogued) using the UDDI (Universal Description, Discovery, and Integration [34]) standard, which is also part of the Web service architecture. At its core UDDI contains a framework for the specification of Web services (and their respective publishers) that is recommended to be used in order to implement Web service catalogues (which are called registries). A distributed public UDDI registry has already been launched in September 2000; private registries (e.g. for the use within businesses) are likely to emerge when corresponding UDDI servers will be available (e.g. within the upcoming Microsoft Windows .NET server).

The UDDI specification framework [35] is currently available in version 3.0 and appears to be relatively stable (matured), since only minor changes have been made from the initial version until now. This paper analyses the applicability of the UDDI standard for specifying Web services and (on this basis) elaborates a more sophisticated specification framework that is called WS-Specification (Web Services Specification). The analysis is based on some theoretical requirements for specification frameworks which are introduced in the next chapter. Thereafter the UDDI specification framework is discussed in detail and a variety of insufficiencies are revealed. This leads to improvements that are summarized within the WS-Specification framework. The paper concludes with a perspective on a unified specification framework that is not only suitable for specifying Web services but components in general.

2 Requirements for Web Service Specification Frameworks

Web service specifications (usually stored in Web service catalogues) are the technological basis to both support the discovery, which consists of search, assessment, and selection, as well as the configuration of Web services, which consists of assembly (wiring or integration, respectively) and binding. The scope and format of specifications are determined by a Web service specification framework, which also serves as the basis for the design of Web service catalogues and consequently is an important part of the emerging Web service architecture.

Although the necessity to establish standardized specification frameworks is widely acknowledged to be a critical success factor for component-based application development, it has rarely been discussed in detail up to now [3], [7], [19], [22]. Consequently, there is currently no referential design for specification frameworks, which could be used as a theoretical basis to comment on the UDDI framework. Nevertheless, some general requirements for Web service specification frameworks can be gathered to rate the applicability of UDDI.

A specification framework should contain both **human- and machine-understandable specifications**. Web service discovery and configuration can occur at built-time (by hand or tool-supported, respectively) or run-time (fully

automated): in tightly coupled applications Web services are discovered and completely configured at built-time, in loosely coupled applications the configuration (or at least the binding) is deferred to run-time, and in self-assembling applications both discovery and configuration occur at run-time (ad hoc) [9]. In order to enable support for automated discovery and configuration, specifications denoted using formal notations should be preferred compared to colloquial languages. If helpful, they can be augmented with non-formal specifications and comments to assist human readers.

It ought to be **methodical**, which means that it should precisely determine what is to be specified and which notations have to be used [1]. This is necessary to provide homogeneous information that can be evaluated by CASE tools. The framework should moreover concentrate on supporting the specification formats that are part of the emerging Web service architecture.

It should introduce **different perspectives** on Web services. Web service specifications tend to be relatively complex specifications that focus on different aspects. Introducing different perspectives on a Web service (e.g. specifications regarding the interface, implemented processes etc.) reduces complexity and enhances readability. Further more, a framework that allows different perspectives can easily be augmented by introducing new perspectives if necessary and at the same time maintain backward-compatibility.

It ought to **satisfactorily document the external view** of a Web service. Especially assessment of Web services ("choosing the right Web services") is based on multi-criteria decision-making [23] which requires comprehensive documentation. Generally speaking, a Web service specification should describe the offered services as well as (all the) relevant conditions that apply when invoking them. Relevant conditions refer to the acquisition of a Web service, its implementation (e.g. the underlying architecture), and its interface(s) [32]. In order to both assess the conditions of purchase and the composition efforts that occur when choosing a specific Web service for configuration during application development, the following specifications are identified to be especially valuable:

- **General specifications** provide information about the Web service publisher, the application domain, and the fees that have to be paid when using the Web service. They are used to decide the acquisition of a Web service.
- **Technological specifications** gather information about the underlying architecture of a Web service, e.g. about the platform it uses to offer its services. At a first glimpse, it may sound somewhat strange to claim information about the underlying technology of a Web service, but this is in fact valuable information because even standardized platforms like the Web service architecture are sometimes being modified[2].

  Moreover, technological specifications contain dependencies to other Web services as well as information about the performance (e.g. response time,

[2] By this time, both SOAP and WSDL already exist in two versions that are not completely compatible. This is called "technological heterogeneity" and has eventually to be considered during configuration.

meantime between failure etc.) and the security (e.g. authentication, data encryption etc.).

- **Syntactic specifications** refer to the interface(s) of a Web service and mainly contain technical information about the signatures of interface-methods and used data types (interface definitions). They are necessary to correctly invoke a Web service method and to design adapters if syntactic heterogeneity between Web services occurs (e.g. different data formats for account numbers between e-banking Web services) [1], [32].
- **Semantic specifications** contain information defining the "meaning" of a Web service and consist of both conceptual and technical information. Conceptual specifications list and define domain-specific concepts that are implemented within a Web service interface (e.g. in an e-commerce Web service "price" is defined as "price including VAT"). They are the basis for designing and correctly interpreting data formats for information exchange between Web services. Thus semantic information has significant importance for the design of translators if domain-specific standards are missing[3] and semantic heterogeneity between Web services is likely to occur.

  Technical specifications contain assertions (usually specified as pre- and post-conditions) that apply to the interface-methods of a Web service and support designing applications by contract [27].
- **Pragmatic specifications** provide information referring to the underlying processes that are implemented within a Web service interface. Again, conceptual and technical specifications can be distinguished. Conceptual specifications describe the domain-specific processes, which are synonymously called workflows (e.g. business-processes like "ordering goods" within a supply chain). These specifications can be used to configure Web services using orchestration engines like workflow-management-systems.

  Technical specifications provide information about the chronological order (coordination) that applies when successively calling interface-methods of a Web service (e.g. you have to authenticate to an e-banking Web service using the login-procedure before requesting the current account balance). Moreover they eventually contain coordination constraints referring to "external" interface-methods of other Web services.

3 The UDDI Specification Framework: State and Challenges

Keeping the above-given general requirements in mind, the applicability of the UDDI specification framework is now analyzed in detail. This chapter starts by giving an overview on the structure of UDDI and later on evaluates, whether it

[3] Domain-specific semantic standards for the field of electronic business are for example introduced within the ebXML core components [24], which provide a common vocabulary, and the Universal Business Language [7], which provides common data formats based on [24].

fulfils the general requirements. During the evaluation, detected insufficiencies are marked as challenges and taken up again in chapter 4.

Fundamentally, the UDDI standard [34], [35], [36] contains two specification frameworks which are unfortunately mixed up and actually should be carefully separated from each other: a framework for the specification of businesses (that can be used to build a business directory) and a framework for the specification of Web services (that is the basis for implementing Web service catalogues).

Web services are usually published by exactly one business. However, including the possibility to reference third-party Web services, the business directory and the Web service catalogue are linked together at the rate of n:m (meaning that a business can publish multiple Web services and a Web service can possibly be referenced by multiple businesses).

The business specification framework, which is found under the conceptual name `<businessEntity>` within the corresponding UDDI data model, contains information about companies that can be divided into different perspectives (called "pages" in the UDDI jargon). Figure 2 gives a sample business specification. It firstly contains some general information about companies: a name, contact information (including addresses), one or more business descriptions, unique identifiers (which are contained in the `<identifierBag>`, e.g. a tax code), and a global unique key (GUID). These are referred to as "white pages" in the standardization papers.

Accordingly, the so called "yellow pages" contain classifications of the specified business, which may include the branch of industry and the geographic location. Each classification is based on a standardized taxonomy, such as UNSPSC (Universal Standard Products and Services Classification [37]) or NAICS (North American Industry Classification System [18]). The classifications are contained in the `<categoryBag>` that belongs to the business specification data model.

Finally, the business specification framework contains zero or more Web service specifications, which include the so called "green pages" containing technical information about Web services. Specifications referring to the published Web services are grouped together and tagged with `<businessServices>` (see figure 2). They each conform to the UDDI Web service specification framework which has been especially designed to support both (automated) discovery as well as configuration of Web services and focuses on specifying their respective external view [9], [35]. Therefore, it contains information referring to different perspectives on Web services starting with a few general (non-functional) specifications, which in fact should be called the "white pages" of a Web service. They consist of the service name, a non-formal description, as well as a global unique service key (GUID).

Moreover, the Web service specification framework classifies the field of application (the domain) of the specified Web service by attaching one or more taxonomy-based specification(s). These specifications, which are grouped within the `<categoryBag>`, in fact represent the "yellow pages" of a Web service.

```
<businessEntity businessKey="32E8F1C4-3BC9-470C-A7C4-702D1BF6EB35">
  <name> Oversoft Software </name>
  <description>
    Oversoft Software is specialized in enabling
    XML Web services and component-based application development.
  </description>
  <businessServices>
    <businessService serviceKey="74cebe59-4adb-4919-9f52-8cbbf6ca4c28">
      <name> Oversoft EasyBanking </name>
      <description>
        Get your current account balance over the Internet.
      </description>
      <bindingTemplates>
        <bindingTemplate bindingKey="f5296cc1-8498-4b09-8e84-7ce9a73d112b">
          <description> EasyBanking WebService Binding </description>
          <accessPoint URLType="http">
            http://www.easybanking.oversoft.biz/easybanking.asmx
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo>
              <instanceDetails>
                  ...
                <instanceParms>
                  http://www.easybanking.oversoft.biz/easybanking.asmx?WSDL
                </instanceParms>
              </instanceDetails>
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
      </bindingTemplates>
      <categoryBag>
        <keyedReference
          tModelKey="70a80f61-77bc-4821-a5e2-2a406acc35dd"
          keyName="Internet Business services, n.e.c." keyValue="7399"/>
      </categoryBag>
    </businessService>
      ...
  </businessServices>
  <categoryBag>
    <keyedReference
      tModelKey="70a80f61-77bc-4821-a5e2-2a406acc35dd"
      keyName="Internet Business services, n.e.c." keyValue="7399"/>
  </categoryBag>
</businessEntity>
```

Fig. 2. A sample UDDI business and Web service specification.

Last but not least, the Web service specification framework provides technical information that can be used during the configuration of Web services. Those specifications, which are primarily evaluated by implementers and configuration tools, are (in accordance with the UDDI standard) called the "green pages" and can be found within the `<bindingTemplates>`. They concentrate on documenting the Web service interface (providing information on its location and listing syntactic specifications as mentioned in chapter 2). Usually, specifying the interface is achieved by importing (the location of) a WSDL specification [10].

Analyzing, whether the UDDI specification framework fulfils the general requirements given in chapter 2, some weaknesses in the conceptual design become evident. In fact only the first requirement given in chapter 2, which claims to pro-

vide both human-and machine-understandable specifications, is fulfilled without doubt. UDDI permits to specify the Web services interface(s) using the formal and machine-understandable WSDL notation. In order to support human readers, these specifications can be augmented with comments as needed. In addition, UDDI allows commenting on Web services using a non-formal description denoted in colloquial language.

However, the UDDI specification framework cannot be characterized as a methodical standard (as required in chapter 2) because it does neither enforce the use of WSDL nor even the use of any formal notation to describe the Web service interface(s). Thus, it is left to the publisher which notation to use for specifying (if one is to be used at all). In order to achieve more homogeneous specifications, the UDDI specification framework should enforce the use of WSDL (or at least a pre-determined mix of notations).

Beyond that, the poorly defined separation between the specification of businesses and the specification of Web services should be considered once again. At a first glimpse, UDDI seems to provide a structure that contains different perspectives and is divided into white, yellow, and green pages. However, white and yellow pages refer to the business specification, while only green pages "officially" contain Web service specifications. In fact, clearly separated different perspectives on Web services are hardly provided by UDDI which mixes up different specifications (general, classification, and technical information) in its corresponding data model. Kindly interpreted (and of course deviating from the standardization papers), different perspectives on Web services can be imagined as shown in figure 3. These different perspectives have been elaborated above by again introducing a thematic grouping consisting of white pages (which contain general information), yellow pages containing category bags, and green pages containing binding templates.

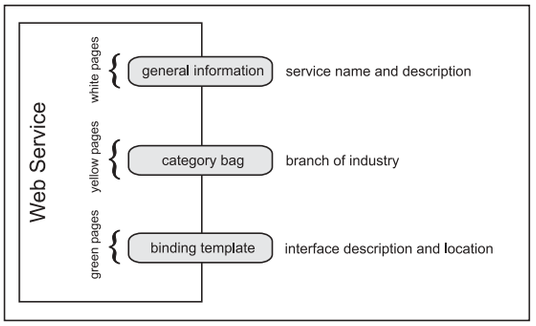


Fig. 3. Visualization of the (modified) UDDI Web service specification framework: It contains three perspectives along with a thematic grouping into white, yellow, and green pages.

Even more important is the fact that UDDI fails to satisfactorily specify the external view of a Web service (the third and most important requirement given in chapter 2). First of all, there is no information referring to the conditions of purchase, which is in fact crucial. Web services are typically designed for physical reuse, i.e. they are being hosted by a provider [33] and can be invoked using a remote procedure call. Consequently, it is likely that different models of pay-per-use will be established (e.g. flat-fees, volume-based rates etc.) which have to be documented appropriately.

Secondly, UDDI should provide additional information in order to better support Web service discovery and configuration (which in fact are its primary fields of application [9], [35]). Specifications referring to the syntax of Web service interface(s) are not sufficient and should be augmented with semantic and pragmatic specifications as well as information about the implementation (as discussed in chapter 2). This leads to a variety of improvements that can be proposed to the UDDI standardization group.

4 WS-Specification: An Improved Specification Framework

This chapter takes into consideration the UDDI insufficiencies and focuses on elaborating a Web service specification framework, which conforms to the general requirements given in chapter 2. It is named "WS-Specification" (which stands for Web Services Specification) and is based on the UDDI specification framework to which it maintains backward-compatibility by augmenting it with new perspectives on Web services. In so doing, the previously-mentioned imperfections are eliminated in order to better support discovery and configuration. The resulting framework is designed to replace a UDDI conformant Web service specification contained within the `<businessService>` tag (see figure 2) and thereby maintain interoperability in contrast to other Web service specification frameworks like for example DAML-S (DARPA Agent Markup Language for Web Services [2]).

WS-Specification is roughly structured by explicitly using the thematic grouping of information referring to Web services that has been introduced in chapter 3: White pages contain general and non-functional specifications referring to the acquisition of a Web service and its implementation. Yellow pages contain classifications of the specified Web service, while green pages hold technical information referring to its interface(s). Besides that, blue pages are being introduced to store conceptual information concerning the interface(s).

In addition, WS-Specification refines this thematic grouping by introducing a total of ten perspectives on Web services as shown in figure 4 to achieve a more structured specification. They are partially taken from a newly introduced specification framework for business components that has been developed by an interdisciplinary standardization group located within the German Society of Informatics [1]. Each perspective contains specialized information using different notations and is separated in the elaborated data model. In the following,

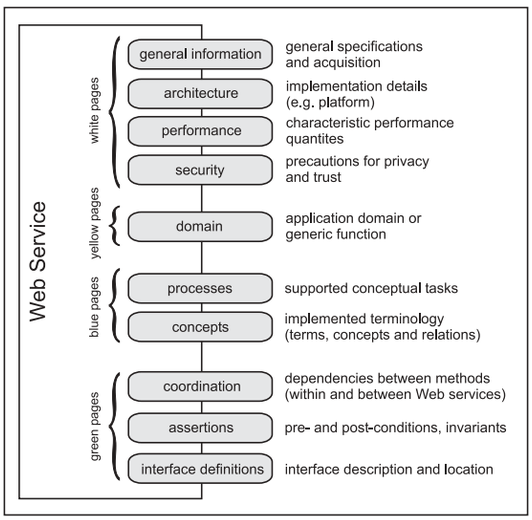


Fig. 4. Visualization of the WS-Specification framework: It contains ten perspectives along with a thematic grouping into white, yellow, blue, and green pages.

WS-Specification is elaborated in detail following the thematic order "white pages", "yellow pages", "blue pages", and "green pages". Each of the previously introduced ten perspectives is discussed within the pages to which it respectively belongs.

4.1 White Pages: General and Technological Information about Web Services

As discussed in chapter 3, the (slightly adopted) UDDI Web service specification framework already contains general information about a Web service (its name, description, and a global unique service key). WS-Specification additionally contributes specifications referring to the acquisition of a Web service by adding information about the license agreement (which contains the conditions of use), the scope of supply, and the distribution channels. Specifications referring to the distribution channels include the prices, accepted forms of payment and thus support different usage rates (e.g. volume-based rates and flat fees). The preferred notation for these specifications is colloquial language (because they are non-functional). WS-Specification supports unmistakable specifications by providing standardized taxonomies (e.g. to specify payment methods like credit card, remittance and so on). Alternatively, ontology-based notations could be an option of future research.

Moreover, information about the architecture of the implementation is provided by the so-called architectural specifications. They contain the platform

that has been used to implement the Web service (e.g. SOAP 1.1) and dependencies to other Web services that eventually exist. A list of platforms is provided by a specialized taxonomy, while dependencies are denoted as an enumeration using a proprietary XML based format.

The white pages are completed by specifications referring to the performance and security of a Web service. Specifications concerning the performance hold information about the quality of service (e.g. expected meantime between failures, maximum response time, maximum data throughput etc.). They are denoted using a notation language that has been borrowed from the complexity theory in computer science and even allows platform-independent temporal specifications (the so-called "O-Notation" [13]) where necessary (in most cases, of course concrete time intervals are to be preferred).

Information referring to the security of a Web service contains message integrity, confidentiality, and authentication which are specified using the WS-Security (Web Services Security [5]) notation. Moreover, a general privacy policy (regarding the transferred data) is conceivable and could be denoted using e.g. P3P (Platform for Privacy Preferences [15]). Figure 5 accordingly augments the Web service specification that was given in chapter 3.

4.2 Yellow Pages: Classifying Web Services

Web service classifications are summarized in the so-called yellow pages. Like UDDI WS-Specification provides a specialized taxonomy to determine the application domain of a Web service (e.g. e-procurement). However, the provided taxonomy is a lot more detailed and supports application domains as well as generic services that affect many domains. Thus, more suitable information is provided to assess the applicability of a Web service during application development.

4.3 Blue Pages: Conceptual Information about Web Services

WS-Specification provides information about the conceptual semantics and pragmatics of a Web service, which is currently not supported by the UDDI specification framework. Because they neither should be added to the white pages (which contain general information) nor to the green pages (which contain technical information), blue pages are introduced to store conceptual information.

The conceptual semantics provide information about the implemented terminology, i.e. the implemented (domain-specific) concepts. A concept is identified by a term and accompanied by a definition (which is called its intension). Moreover, it is possible to sharpen a concept by specifying its extent and to bring it into relation to other concepts. Specifying the implemented terminology can be achieved by providing a lexicon (containing terms and their respective definitions). In so doing definitions have to be denoted in normative (regulated) colloquial language [30] to enable automated processing. This especially means providing plans for the syntactic construction of sentences. Alternatively, ontology-based notations, which provide information about terms and

```
<businessService serviceKey="74cebe59-4adb-4919-9f52-8cbbf6ca4c28">
    ...
  <termsAndConditions>
    Limitation of Liability ...
    Copyright Notices ...
  </termsAndConditions>
  <scopeOfSupply>
    Oversoft EasyBanking can easily be used by downloading the EasyBanking Client
    which is available under http://www.easybanking.oversoft.biz. ...
  </scopeOfSupply>
  <distribution>
    <channel>
      <name> unlimited use (flat rate) monthly payment </name>
      <price currencyKey="811464A4-823F-4a87-9F85-5B69443705B1" name="usd">
        9.90
      </price>
      <acceptedPayments>
        <payment key="3c27116-5493-4931-9411-dd2218e84e11" name="debit advice"/>
        ...
      </acceptedPayments>
    </channel>
    ...
  </distribution>
  <architecture>
    <platform key="D3AAA982-9D28-42af-B94B-C3FDA5EF82AD" name="SOAP1.1"/>
  </architecture>
  <performance>
    <specification key="0F092294-7652-419d-8E58-F58E33F1C6B5" name="mtbf">
      1420.5 days
    </specification>
    ...
  </performance>
  <security>
    <specification key="4141D795-689B-4597-A5FC-12A1BF3DC260" name="ws-sec">
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <SignatureMethod Algorithm=" http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>LyLsF0Pi4wPU...</DigestValue>
        </SignedInfo>
      </Signature>
    </specification>
    ...
  </security>
    ...
</businessService>
```

Fig. 5. A variety of sample specifications illustrating the WS-Specification white pages.

concepts using a machine-readable format, could be an option of future research. Conceptual terms can usually be mapped to data types that are used within the interface (for example, the conceptual term "account" maps to a corresponding data type named "Account"). Explicitly documenting these mappings facilitates the design of semantic translators based on a given conceptual semantic.

In addition to conceptual semantics the blue pages also hold information about the conceptual pragmatics, i.e. about the underlying (domain-specific) processes and workflows (e.g. business processes). Specifications referring to processes focus on documenting conceptual tasks (that are being automated) and their decomposition into sub-tasks. They can be denoted using a formal workflow definition language, e.g. BPML (Business Process Markup Language [4]) or

```
<businessService serviceKey="74cebe59-4adb-4919-9f52-8cbbf6ca4c28">
    ...
  <processes>
    <specification key="5D071356-7D35-40ba-B175-960D7D9063B0" name="bpml">
      <process name="GetBalance" xmlns="http://www.bpmi.org/2002/6/BPML">
        ...
      </process>
    </specification>
    ...
  </processes>
  <concepts>
    <specification key="D93C2858-8164-46df-BA22-37D2A0AF0564" name="lexicon">
      <concept>
        <term> account </term>
        <definition>
          stores information about the current account business which can be
          evaluated by getting an account statement. Accounts can be debited or
          balanced, respectively. ...
        </definition>
      </concept>
    </specification>
    ...
  </concepts>
    ...
</businessService>
```

Fig. 6. A variety of sample specifications illustrating the WS-Specification blue pages.

BPEL (Business Process Execution Language [16]) that is specialized in linking business processes to Web service methods.

Conceptual (sub-) tasks can usually be mapped to a corresponding interface-method. Explicitly documenting these mappings allows automated orchestration of Web services by workflow-management-systems (or other orchestration servers, e.g. EAI servers, respectively). Moreover, conceptual workflows predetermine ordered sequences of method invocations that are important to correctly handle a Web service and will be discussed later on (see section 4.4). Figure 6 shows some sample specifications that belong to the blue pages (mappings of concepts and processes have been omitted for brevity).

4.4 Green Pages: Technical Information about Web Service Interfaces

Following the UDDI specification framework, WS-Specification also supports the specification of the Web service location and its interface(s). This information is the basis to configure and finally invoke a Web service. Interface specifications contain information about the named interface-methods including their signature, named public properties as well as variables, constants, and declarations of specific data types. Moreover, they document possible exceptions (which in the Web Service Description Language are called "faults"). Unlike UDDI, it enforces the use of WSDL as formal notation to denote interface specifications and stores specifications directly in the catalogue repository (as a part of the specification).

Moreover WS-Specification provides some more detailed technical information, because the syntactic interface specification is usually not sufficient to cor-

rectly invoke a Web service or configure it with others [14]: In order to correctly invoke a Web service's methods, semantic information about them is required, which determines the applying prerequisites and the results when invoking an interface-method. On the other hand, Web service methods can typically not be arbitrarily invoked, but only called in well determined orders, in other words there are dependencies between Web service methods. Both are considered by WS-Specification, which consequently introduces two specific perspectives called "assertions" and "coordination".

Information about the (technical) semantics of a Web service is provided by specifying pre- and post-conditions which refer to interface-methods. These so called assertions support designing applications by contract [27], a well-established method of software engineering. Pre-conditions express the constraints under which an invoked method returns correct results. Accordingly, post-conditions describe the respective state resulting from a method's execution and thus guarantee that it will satisfy certain conditions (provided that it was called with the pre-condition satisfied).

In addition to pre- and post-conditions the specification of invariants is sometimes useful to describe global properties which are preserved by all methods. Both pre- and post-conditions as well as invariants can be specified using the Object Constraint Language (OCL), a formal notation provided as part of the Unified Modelling Language (UML [11]). Moreover, formal specifications can optionally be accompanied by comments using colloquial language because they are somewhat difficult to understand for human readers.

Constraints referring to the ordered invocation of Web service methods (the so-called coordination constraints) can occur both within a single Web service (e.g. one has to login to an e-banking service before requesting the current account balance) but also between different Web services (this is already the case if the login-service is provided as autonomous service by a third party, e.g. Microsoft .NET Passport). They can especially be of help when configuring a Web service on the basis of application control flows. Coordination constraints can be denoted using a specialized format like WS-Coordination (Web Services Coordination [8]) or WSFL (Web Services Flow Language [25]). Moreover it is possible to use an extended OCL notation, which contains the temporal operators before, after, sometime, always, until, sometimes_before, sometimes_past, always_past (the corresponding semantics of the operators is described in [1]). As with the specifications that contain assertions, coordination-constraints can optionally be accompanied by comments in colloquial language (or by a UML sequence diagram, which is applicable to specify orders).

Figure 7 shows a sample containing interface definitions, assertions, and coordination constraints. It denotes coordination constraints using the extended OCL notation that has been introduced within this chapter.

4.5 A Tailor-Made Data Model for WS-Specification

Although enabling interoperability between UDDI and WS-Specification by maintaining backward-compatibility to the UDDI data model brings many ad-

```
<businessService serviceKey="74cebe59-4adb-4919-9f52-8cbbf6ca4c28">
    ...
  <coordination>
    <specification key="07793A52-BD60-4961-A03C-DF684DDE8282" name="extocl">
      <formalStatement>
        EasyBanking::GetBalance() pre: sometime_past(Login())
      </formalStatement>
      <description>
        Before retrieving the balance of the account one has to log in.
      </description>
    </specification>
    ...
  </coordination>
  <assertions>
    <specification key="E9D7C918-C8BA-42f0-8559-B1C5B2DB18FE" name="ocl">
      <formalStatement>
        EasyBanking::Login() pre: self.LoginCredentials.length > 0
      </formalStatement>
      <description>
        The login credentials must not be empty.
      </description>
    </specification>
    ...
  </assertions>
  <interfaceDefinitions>
    <specification key="CF53E356-FF7E-4538-AEEE-7B98867F1A9F" name="wsdl1.2">
      <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
        <types>
          ...
      </definitions>
      ...
    </specification>
    ...
  </interfaceDefinitions>
    ...
</businessService>
```

Fig. 7. A variety of sample specifications illustrating the WS-Specification green pages.

vantages, it leads to a somewhat inconsistent XML data model. As seen before, the newly introduced perspectives are each modelled as data groups tagged with a self-describing name (such as `<concepts>`, `<processes>`, `<assertions>`, `<coordination>` etc.). However, this is untrue for the specifications that have been taken from the original UDDI data model. Some of them are not grouped at all (which is in fact true for the general information). Others are tagged with a rather technical name (e.g. `<categoryBag>` tagging classifications or `<bindingTemplate>` tagging interface specifications respectively). Moreover, the thematic grouping into white, yellow, blue, and green pages has not been included into the UDDI data model and remains at the conceptual level. Because Web services specifications usually contain a lot of data, this "design-fault" is worsening readability.

Therefore it is conceivable to abandon backward-compatibility with the UDDI data model and to create a tailor-made data model for WS-Specification in order to obtain a homogeneous data structure that is clearly and completely divided into different perspectives. Its XML data model is shown in figure 8 that

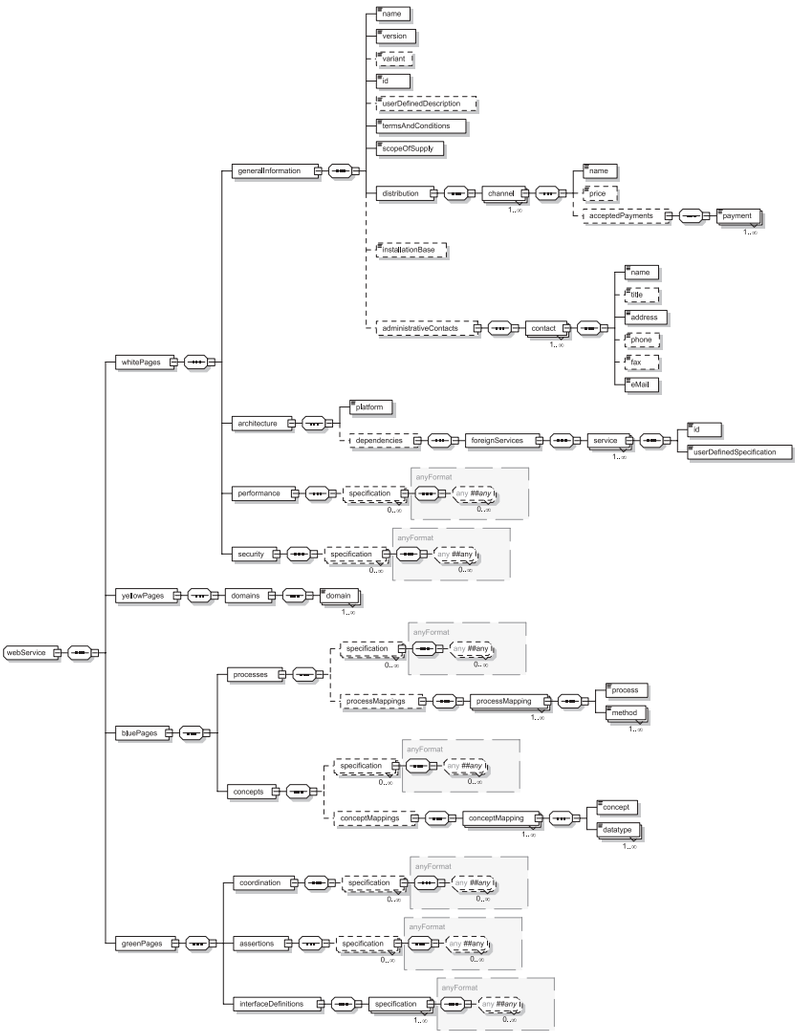


Fig. 8. The WS-Specification data model (denoted as XML Schema).

replaces specific specification formats (which respectively depend on the used notations) with the generic "any format" data type for brevity.

If UDDI was still on the drawing board, this data model would have been an alternative designed to replace the corresponding UDDI data model. In fact working UDDI implementations (listed under www.uddi.org) already exist, so that a tailor-made data model rather implements an alternative designed to compete with the UDDI data model. Because XML data easily can be trans-

formed using XSL(T), it nevertheless is plausible that both data models in fact successfully coexist and gain in significance. WS-Specification can even be used in parallel with existing UDDI implementations since it implements in fact a superset of specifications.

Thus it is conceivable to store common information within UDDI repositories and the additional information within repositories conformant to WS-Specification. Such a repository for WS-Specification could then work as a wrapper that encapsulates a UDDI repository and respectively delivers specifications conforming to UDDI or WS-Specification. In so doing, Web services specified using WS-Specification can automatically be propagated and published to existing UDDI registries.

5 Conclusions and Future Directions

This paper proposes a variety of improvements to the UDDI Web service specification framework in order to ease Web service discovery and configuration. It introduces a detailed specification framework that is divided into ten aspects of documentation and four thematic groups. The commitment to enforce the use of formal notations supports the design of CASE tools to automate the processes of Web service specification, assessment, and configuration. Each of the improvements has been encapsulated using a separate perspective of specification. Thus, the resulting framework WS-Specification is modular and can easily be augmented if necessary in the future.

As it has been discussed in this paper, the emerging Web service architecture frequently discusses problems and issues that not only apply for Web services but component-based application development in general. Because Web services incorporate a special sort of components, many solutions can be transferred to the "conventional" world of components. This is especially true for the here introduced specification framework that can easily be adopted to specify components mainly by augmenting the allowed notations. Thus, a framework for the unified specification of software components could be achieved, which solves many key problems and prepares the ground for future component catalogues and next generation CASE tools.

Considering these possible developments, Web services really mark a step into the new era of application-integration and component-based application development.

References

1. Ackermann, J., Brinkop, F., Conrad, S., Fettke, P., Frick, A., Glistau, E., Jaekel, H., Kotlar, O., Loos, P., Mrech, H., Ortner, E., Overhage, S., Raape, U., Sahm, S., Schmietendorf, A., Teschke, T., Turowski, K.: Standardized Specification of Business Components. German Society of Computer Science (2002) `http://wi2.wiwi.uni-augsburg.de/gi-memorandum.php`

2. Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H.: DAML-S: Semantic Markup for Web Services. In: Proceedings of the International Semantic Web Working Symposium SWWS (2001)
http://www.daml.org/services/SWWS.pdf
3. Apperly, H., Booch, G., Councill, B., Griss, M., Heineman, G. T., Jacobson, I., Latchem, S., McGibbon, B., Norris, D., Poulin, J.: The Near-Term Future of Component-Based Software Engineering. In: Councill, W. T., Heineman, G. T. (eds.): Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Upper Saddle River, New Jersey (2001): 753–774
4. Arkin, A. (ed.): Business Process Modelling Language (BPML). BPMI Working Draft (2001) http://www.bpmi.org
5. Atkinson, B., Della-Libera, G., Hada, S., Hondo, M., Hallam-Baker, P., Kaler, C., Klein, J., LaMacchia, B., Leach, P., Manferdelli, J., Maruyama, H., Nadalin, A., Nagaratnam N., Prafullchandra, H., Shewchuck, J., Simon, D. (eds.): Web Services Security (WS-Security). Public Draft (2002)
http://msdn.microsoft.com/ws/2002/04/Security
6. Austin, D., Barbir, A., Garg, S. (eds.): Web Service Architecture Requirements. W3C Working Draft (2002) http://www.w3.org/TR/wsa-reqs
7. Bosak, J. (ed.): UBL: The Next Step for Global E-Commerce. White Paper (2002)
http://oasis-open.org/committees/ubl/msc/200204/ubl.pdf
8. Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D., Shewchuk, J., Storey, T. (eds.): Web Services Coordination (WS-Coordination). Public Draft (2002) http://msdn.microsoft.com/ws/2002/08/WSCoor
9. Cauldwell, P., Chawla, R., Chopra, V., Damschen, G., Dix, C., Hong, T., Norton, F., Ogbuji, U., Olander, G., Richmann, M. A., Saunders, K., Zaev, Z.: Professional XML Web Services. Wrox Press, Birmingham (2001)
10. Cerami, E.: Web Services Essentials. O'Reilly, Sebastopol, California (2002)
11. Cheesman, J., Daniels, J.: UML Components: A Simple Process for Specifying Component-Based Software. Addison Wesley, Upper Saddle River, New Jersey (2000)
12. Chinnici, R., Gudgin, M., Moreau, J. J., Weerawarana, S. (eds.): Web Services Description Language (WSDL) Version 1.2. W3C Working Draft (2002)
http://www.w3.org/TR/wsdl12
13. Cormen, T. H., Leiserson, C. E., Rivest, R. L.: Introduction to Algorithms. MIT Press, Cambridge, Massachusetts (2001)
14. Councill, B., Heineman, G. T.: Definition of a Software Component and Its Elements. In: Councill, W. T., Heineman, G. T. (eds.): Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Upper Saddle River, New Jersey (2001): 5–19
15. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J. (eds.): The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation (2002) http://www.w3.org/TR/P3P
16. Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., Weerawarana, S. (eds.): Business Process Execution Language for Web Services, Version 1.0. Public Draft (2002)
ftp://www.software.ibm.com/software/developer/library/ws-bpel.pdf
17. Czarnecki, K., Eisenecker, U. W.: Generative Programming: Methods, Tools, and Applications. Addison Wesley, Upper Saddle River, New Jersey (2000)

18. Economic Classifications Policy Committee (ed.): The North American Industry Classification System (NAICS)
http://www.census.gov/epcd/www/pdf/naicsbch.pdf
19. Flynt, J., Desai, M.: The Future of Software Components: Standards and Certification. In: Councill, W. T., Heineman, G. T. (eds.): Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Upper Saddle River, New Jersey (2001): 693–708
20. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J., Nielsen, H. F. (eds.): SOAP Version 1.2 Part 1: Messaging Framework. W3C Working Draft (2002)
http://www.w3.org/TR/soap12-part1
21. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J., Nielsen, H. F. (eds.): SOAP Version 1.2 Part 2: Adjuncts. W3C Working Draft (2002)
http://www.w3.org/TR/soap12-part2
22. Griss, M. L.: CBSE Success Factors: Integrating Architecture, Process, and Organization. In: Councill, W. T., Heineman, G. T. (eds.): Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley, Upper Saddle River, New Jersey (2001): 143–160
23. Konito, J.: A Case Study in Applying a Systematic Method for COTS Selection. In: Proceedings, 18th International Conference on Software Engineering (ICSE). IEEE Computer Society Press (1996) 201–209
24. Kotok, A., Webber, D.: ebXML: The New Global Standard for Doing Business on the Internet. New Riders Publishing, Mass. (2001)
25. Leymann, F. (ed.): The Web Services Flow Language. Public Draft IBM Research (2002)
http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
26. McIlroy, M. D.: Mass Produced Software Components. In: Naur, P., Randell, B. (eds): Software Engineering: Report on a Conference by the NATO Science Committee. NATO Scientific Affairs Division, Brussels (1968) 138–150
27. Meyer, B.: Object-Oriented Software Construction. Prentice Hall, Cambridge (1988)
28. Microsoft Corporation: Web Services Specifications. Public Draft (2002)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsspecsover.asp
29. Mitra, N. (ed.): SOAP Version 1.2 Part 0: Primer. W3C Working Draft (2002)
http://www.w3.org/TR/soap12-part0
30. Ortner, E., Schienmann, B.: Normative Language Approach – A Framework for Understanding. In: Thalheim, B. (ed.): Conceptual Modeling – ER '96, 15th International Conference on Conceptual Modeling, Proceedings, Cottbus 1996. Springer, Berlin (1996) 261–276
31. Shaw, M., Garlan, D.: Software Architecture – Perspectives on an Emerging Discipline. Prentice Hall, Upper Saddle River, New Jersey (1996)
32. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison Wesley, Harlow (1998)
33. Szyperski, C.: Components and Web Services. In: Software Development Magazine 9 (2001) 8 http://www.sdmagazine.com/documents/sdm0108c
34. UDDI Organization (ed.): UDDI Executive White Paper. UDDI Standards Organization Public Draft (2001)
http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf
35. UDDI Organization (ed.): UDDI Technical White Paper. UDDI Standards Organization Public Draft (2000)
http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

36. UDDI Organization (ed.): UDDI Version 3.0. UDDI Open Draft Specification (2002) http://www.uddi.org/pubs/UDDI-V3.00-Open-Draft-20020703.pdf
37. United Nations Organization (ed.): The Universal Standard Products and Services Classification (UNSPSC).
http://www.eccma.org/unspsc/crosswalk.htm

Modeling Web Services Variability with Feature Diagrams

Silva Robak[1] and Bogdan Franczyk[2]

[1] University of Zielona Góra
Institute of Organization and Management
ul. Podgorna 50,
PL-65-246 Zielona Gora, Poland
S.Robak@iiz.uz.zgora.pl
[2] Universität Leipzig
Wirtschaftswissenschaftliche Fakultät
Institut für Software und Systementwicklung
Lehrstuhl für Informationsmanagement
Marschnerstr. 31
D-04109 Leipzig
franczyk@wifa.uni-leipzig.de

Abstract. In the paper the proposal for modeling flexibility of the Web Services with feature diagrams is introduced. Feature diagrams allow presenting the commonality and above all variability of described concept. In the paper the classification of Web Services features from the users' point of view is given. The comparison of Web Services with the orthogonal component concept is also discussed.

1 The Notion of the Web Service

1.1 Properties of Web Services

In [8] the definition of the Web Service is given as: "any process that can be integrated into external systems through valid XML documents over Internet protocols". XML is used to define the payload (i.e. data transferred between processes during execution) of a Web Service.
The definition in [1] is more comprehensive and includes all Web Service significant features: "Web Services are modular, self-describing applications that can be published, located and invoked from just about anywhere on the Web or a local network. The provider and the consumer of the XML Web service do not have to worry about operating system, language, environment, or component model used to create or access the XML Web service, as they are based on ubiquitous and open Internet standards, such as XML, HTTP, and SMTP". Further is stated that XML and SOAP are the base technologies of Web Services architectures. The Web Services communication technology options are summarized in the Fig. 1., containing the communication architecture subcomponents: Consumer, Transport and the Service. The Consumer denotes the entity utilizing the Web Service, the Service is a provider

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 120–128, 2003.

of the Web Service and the Transport defines the means for the communication of Consumer while interacting with a Service. The communication for Web services specifies two layers (i.e. network and transport) in ISO-OSI reference model. The details of the feature diagrams' notation are given in section 2.1.

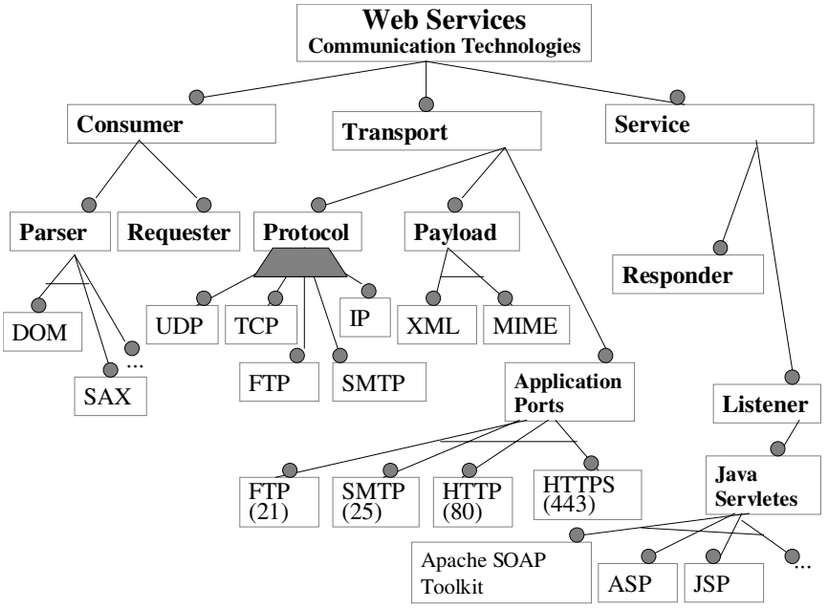


Fig. 1. Web Services Communication Technologies.

The definition of the Web Service is not uniform, caused by the facts, that proprietary descriptions use their own definition of the Web Service, as summarized in [5] with definitions from W3C and WebServices.org and other sources as Intel, IBM, SUN, Microsoft, Hewlett-Packard (HP), Globus Project, Gartner Group. The characteristic features of a Web Service are further that they are: immaterial, transient (i.e. not feasible to stock), position dependent (not viable to transport) and that synchronize producer with consumer immediately [6]. The Web Services are much more then just System Integration because they are providing services (not just connecting systems), they are neutral (i.e. not dependent on a special programming paradigm) and the cooperation proceeds in ad hoc manner (as opposite to long lasting cooperation). WS architecture contains at least three roles: Service Provider, Service Requester and Service Broker, with the responsibilities: publishing (Service Provider to Service Broker: WSDL), finding (Service Requester to Service Broker: UDDI) and bind & execute (between the Service Provider and Service Requester: SOAP).

The WebServices.org defines Web Service as encapsulated, loosely coupled contracted functions offered via standard protocols. The standard-based com-

munication allow accessing the services independent of hardware, operating system or programming environment, i.e. are programming language-, programming model- and system software neutral as underlined by Globus Project and HP. Gartner Group also emphasizes the aspect of Web Service as loosely coupled software components delivered over Internet standard technologies. The IBM definition underscores just-in-time application integration aspect and the possibility of their dynamically changing by the creation of new Web Service. The Intel definition emphasizes the ability of performing 'distributed computation' with the 'best-matched device for the task' and the information delivery on a 'timely basis' in the form needed by the user.

Some definitions refer to the Web Services as "modular and reusable software components" (Hewlett-Packard), or " loosely coupled software components" (Gartner Group). Web Services combine the best aspects of component-based development and the Web (Microsoft) and also define a technique for describing software components (Globus Project). So, Web Services just form the orthogonal concept to the components notion including their key features as providing the functionality as black box with described and published interfaces.

In the paper the aspects comparing the concepts of Web services and components are the topic of section 1.2. In section the sources for variability contained in Web services are given together with an approach for presenting this variability within the feature diagrams. In section 3 we conclude our work.

1.2 Web Services vs. Components

In [3] is contained the opinion, that the main problem is not what Web Service are, but how to use them, with setting the goal as easier and more productive reuse of existing assets. The recent trends include:
- Larger level of granularity,
- Reduced effort required to reuse code,
- Increasing breadth of scope (internet and industrial standards),
- Increased focus on information (use of XML for the payload),
- Transformation between proprietary and industry standards,
- Reduced effort required to use of Web Services,
- New level of interoperability.

Considering this characteristics one can say, that Web Services are the natural consequence in the reuse succession containing classes, components and finally Web Service. The reuse tendency first shifted from objects (i.e. classes) to delivery-and -deployment unit of higher granularity i.e. components [10]. Most reduced effort required to use the units and to reuse code is also in case of Web Service. It is caused not only by larger level of granularity of Web Service, but also through the existence of the appropriate services allowing finding and discovering Web Service (UDDI, WSDL). They offer just much more than only standard interfaces. In [12] are some other trends given, supported by Web services being able to be intelligently process, manipulate and aggregate content:
- Content becomes more dynamic,
- Decreasing costs for bandwidth and storage,
- Increasing importance of pervasive computing.

The Web Services are also capable to enrich the organization business by among others [2]:
- Delivering more affluent e-business applications,
- Run virtually enterprises with more efficient supply chains,
- Grouping products and (software) services.

Loosely coupled services may widely support the B2B collaboration.

Another difference between the component and Web Service is their service-oriented architecture containing three (main) roles: producer, consumer and broker. An operating agent delivers the service to some clients. The runtime environment (with such features as scalability, reliability, security, etc.) should be based on high-performance application server or message broker.
According to [11] the Web Service is a pairing of an operation agent (with it's infrastructure) with software components rather then with arbitrary software. The extreme dynamic nature of Web-based systems includes the specific fact that their parts will evolve separately.

As opposite to Web services the features of a separate software component according to [11] are subsumed in following way:
- It provides functionality through well defined standards,
- It contains a full declaration of its static dependencies,
- Is equipped with explicit configuration mechanisms (required interfaces as opposed to the more traditional provided interfaces),
- Is equipped with version identification.

As stated in the section 1.1 the Web Services just form the orthogonal concept to the components notion with such common properties as providing the encapsulated functionality as a loosely coupled black boxes with described and published interfaces.

2 Describing Web Services with Feature Diagrams

2.1 Feature Diagrams

A *feature* is a visible characteristic of a concept (e.g. system, component, etc.), which is used to describe and distinguish different instances of the concept. The feature model indicates the *intention* of the described concept. The set of instances described by feature model is the *extension* of the concept. This model is particularly important in situations, where a generic description for a range of diverse systems is needed. In case of Web Services a feature diagram may describe a generic concept, which can be further dynamically, customized do profiles' consumers. Upon an established base in form of commonalities and the differences specified to this base, a speedy automated creation of Web Service version for a customer would be possible.

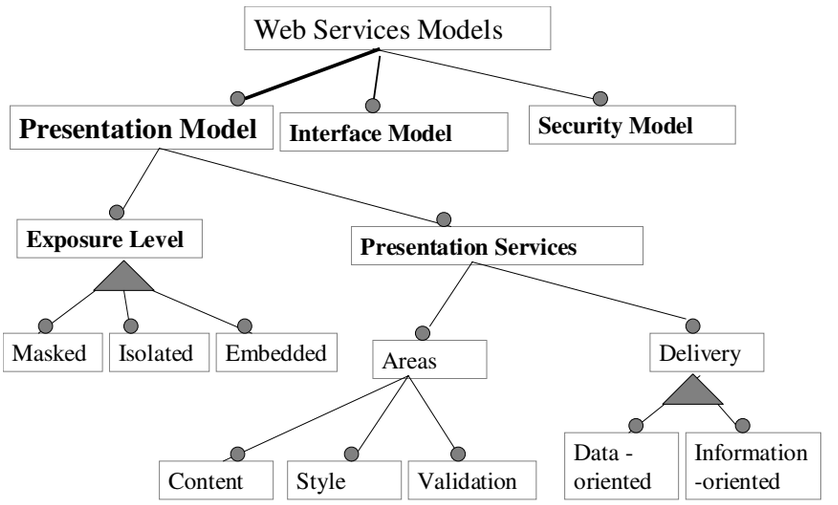


Fig. 2. Web Services Models - Presentation Model.

FODA [7] feature models are the means to describe mandatory, optional, and alternative properties of concepts within a domain. The most significant part of a feature model is a *feature diagram* that forms a tree (graphical AND/OR hierarchy of features) and captures the relationships among features. A root of the tree represents a concept being described and the remaining nodes denote features and their sub-features. Direct features of a concept and *sub-features* (i.e. features having other features as their parents) are distinguished. Direct features of a software system may be mandatory, alternative, or optional with respect to all applications within the domain. A sub-feature may be mandatory, alternative, or optional with respect to only the applications, which also enclose its parent feature. If the parent of the feature is not included in the description of the system, its direct und indirect sub-features are unreachable. Reachable *mandatory* features must be always included in every system instance an *optional* feature (denoted by an empty circle) may be included or not, and an *alternative* feature (denoted by an empty arc) replaces another features when included.

In the GP (Generative Programming) feature diagrams notation [4], that seems to be the present most popular feature diagram presentation, there is an additional (compared to FODA) kind of features introduced i.e. *or-features* (denoted by filled arc) representing multiple choices in the alternative. In the paper the GP-feature diagrams are used to describe the commonality and variability contained in the Web Services technology and service models.

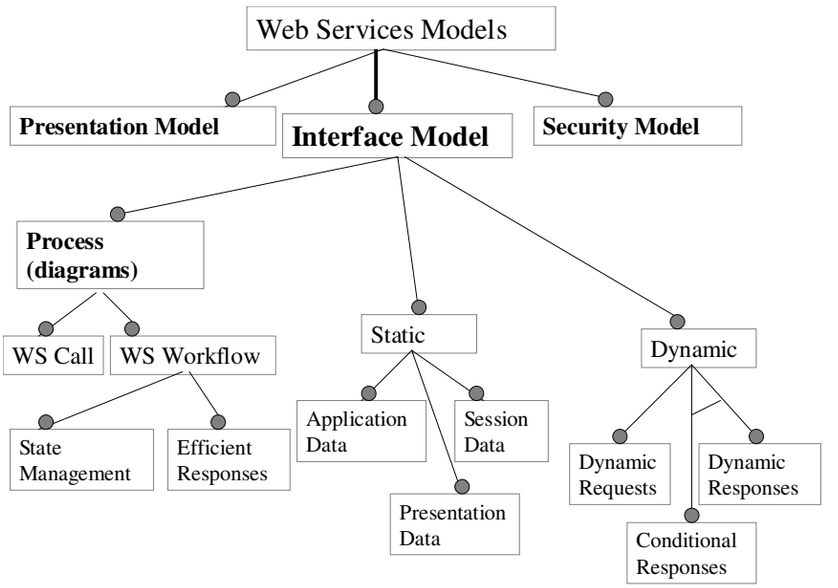


Fig. 3. Web Services Models – Interface Model.

2.2 Variability of Web Services

Web Services may be exposed to a large pool of users, as well mix and match services from other providers to build a variety of applications. The further sources of variability of WS are:

- Number of participants (e.g. in an application chain with multiple partners),
- Participants' level of involvement and
- Technology and platform chosen.

WS application can get very complex, depending e.g. on how many partners integrate their service. The participant's level of contribution includes the responsibility for presenting the application to end-user or providing the functionality for its piece of the process, not delivering the entire application. One more high-level scenario is also possible i.e. the owner-broker model [8]. Selecting the technology and platform may include following aspects as:

- Referencing legacy system - e.g. COM-based,
- Goal platform - e.g. MS-Windows 2000,
- Utilized development platform- e.g. MS Development Platform for COM+,
- Interface middleware - e.g. ASP,
- Maintaining data (consumer accounts and session data) – e.g. MS SQL Server.

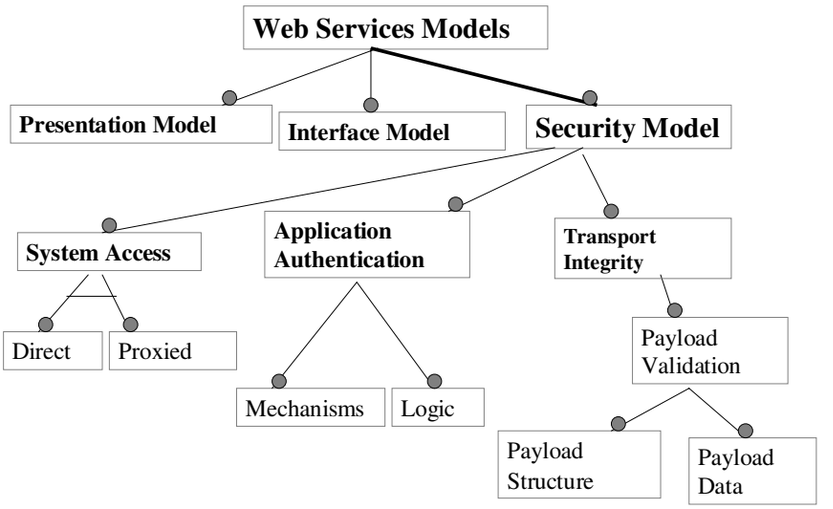


Fig. 4. Web Services Models - Security Model.

The choice between communication technology protocols: TCP (connection oriented) and UDP (connectionless) will be dependent on the current requirements i.e. TCP tries to ensure delivery at cost to overall performance, while UDP focuses on the highest possible performance.

The Web service consumer i.e. direct caller of WS (client in a client-server architecture) provides the widest range of possibilities in functionality. It may be responsible for the presentation to the end user brokering multiple Web services, or simply passing through a Web service call. Furthermore, different consumers on the same Web service may want to use a Web service in completely different ways. One may pass info straight to the client; another may want to use Web service wholly masking his existence (see the subfeature Masked of the Exposure Level in the Presentation Model in Fig. 2).

It has to be emphasized, that Web services themselves do not have a presentation layer. In combined logical and communication architecture for a Web services [8] there are following parts included:

- Presentation (Consumer)
- Integration/Interface (Consumer/Provider)
- Business (Provider)
- Data (Provider)

Presentation layer for a consumer (see Fig. 5) may include the most degree of diversity from an application to application.

Web service models reflecting given business goals and priorities, will further vary because of having different features:

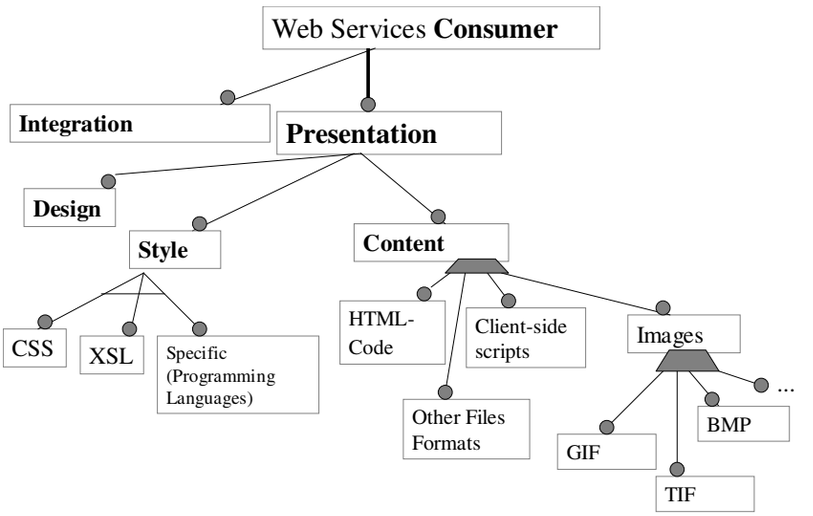


Fig. 5. Web Services Consumer Logical Sublayers.

- Target consumer,
- Functions provided by service,
- Bundled, or directly consumed service,
- Dynamic or static service, etc.

The parts of Web service models i.e. Presentation, Interface and Security (as depicted in feature diagrams in Fig. 2-4) containing different possibilities, which have to be instantiated for an individual Web service. In some situations there are multiple instances of the parts in the model possible – e.g. there may be a need to support more than one security model to accommodate different requirements.

3 Conclusion

There is a need for building taxonomies for Web Services, because the present Web Services architectures are diverse and proprietary for specific manufactures. In the paper different properties of Web Services, regarded from different points of views were presented. Web Services form the orthogonal concept to the components' notion including their key features as providing the encapsulated functionality as loosely coupled black boxes with described and published interfaces. The Web Services may be regarded as services for Web as well as components in the complex workflows.

In the paper the proposal for modeling flexibility of the Web Services with feature diagrams is introduced. The knowledge contained in a feature diagram may be considered as a configuration knowledge, that can be dynamically customized it with

a generator for an individual Web Services profiles' consumers. For the feature diagrams the notation introduced in GP was used. Moreover, the use of UML based notations for feature diagrams (like introduced in [9]) is recommended. In the future the further investigations for the specific Web Services domains are needed.

References

1. Cauldwell, P., Chawla, R., Chopra V., B., Damschen G., Dix, C. et. al: Professional XML Services. Wrox Press Ltd, (2001)
2. Coco, J.: Maximizing the Potential of Web Services. XML-Journal. SYS-CON Media, Inc. (2001). http://www.sys-con.com/xml.
3. Coco, J.: Code Reuse: From Objects to Components to Services. XML-Journal. SYS-CON Media, Inc. (2002). http://www.sys-con.com/xml.
4. Czarnecki, K., Eisenecker U.: Generative Programming Methods, Tools and Applications. –Addison-Wesley, New York (2000).
5. Jeckle, M.: Web Services. Begriffsdefinitionen.(2002). http://www.jeckle.de/webServices
6. Jeckle, M.: Web Service-Architecturen. Presentation slides at "XML in Action 2002". Potsdam (2002).
7. Kang K., Cohen S., Hess J., Nowak W. and Peterson S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990. Pennsylvania
8. Oellermann, W.L, Jr.: Architecting Web Services. Apress. Springer Verlag, New York (2001).
9. Robak, S., Franczyk, B., Politowicz K.: Extending The UML for Modelling Variabilities for System Families, International Journal of Applied Mathematics and Computer Science, 12(2), 2002.
10. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. – New York, 1998. Addison-Wesley
11. Szyperski, C.: Components and Web Services. Software Development Magazine. August 2001.). http://www.sdmagazine.com.
12. Tidwell, D.: Web services – the Web's next revolution. http://ibm.com/developerWorks

A Dependency Markup Language for Web Services

Robert Tolksdorf

Freie Universität Berlin, Institut für Informatik
Netzbasierte Informationssysteme
Takustr. 9, D-14195 Berlin, Germany
research@robert-tolksdorf.de, http://www.robert-tolksdorf.de

Abstract. Current mechanisms for the description of Web Services and their composition are either to coarse – by specifying a functional interface only – or too fine – by specifying a concrete control flow amongst services.
We argue that more adequate specifications can be built on the notion of *dependency* of activities and coordination activities to manage these dependencies. We propose a Dependency Markup Language to capture dependencies amongst activities and generalizations/specializations amongst processes. With that, we can describe composite services at more suited levels of abstraction and have several options to use such descriptions for service coordination, service discovery and service classification.

1 Introduction

Web Services implement the notion of small networked services that can be combined to realize more complex processes or composite services (that we sometimes call processes in the following). With standards for the representation of their interfaces, their composition and the exchange of messages with them, an open market for services becomes possible. They are enabling to compose complex processes from services that are offered and implemented by different organizations.

The goal with Web Services is to allow an *automated* discovery and composition of services. For that, means and languages for machine-readable descriptions of services are necessary. XML is considered to offer the best chances for a wide exchangeability of such descriptions.

Currently, there is a number of proposed XML-based languages existing and emerging for the description of Web Services. We can assume that they will converge in the midterm. Table 1 shows some of the currently discussed standards for the representation of combined services and their main characteristic. [AMS02] gives a brief introduction to most of them.

The proposed languages usually address two aspects that describe a service: its external interface and its internal behavior. The external interface is specified by a functional description of the services provided. Semantic information can be given in the form of contracts, either for the service provider – for example with pre- and postconditions – or the service user – how the services have to be used. The internal behavior can be specified as a service flow which is commonly described by the specification of a control flow.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 129–140, 2003.

Table 1. Current (proto-)standards for combined services

Language	External	Internal
WSFL [Ley01]	functional interface specified	data- and control flows specified
XLANG [Tha01]	interface specified with WSDL	control flow specified, event handling
WSCL [BBB+02]	allowed interactions (conversation) specified by interaction-transition net	not specified
DAML-S [ABH+02]	functional interface specified	control flow specified by imperative constructs
ASDL [ZC02]	allowed usage specified by state-(conditioned)transition net	control flow specified
WSMF [FB02]	functional interface specified with pre- and postconditions	not specified
BPSS [Bus01]	interactions specified by message exchanged	state/transition net
BPML [Agr01]	interactions specified by message exchanged	control flow specified by imperative constructs

In order to identify a service, one searches for a specific interface, perhaps with further requirements on contracts. In the next section, we discuss why this is not always the right kind of service description.

2 How to Describe Processes

[WL95] describes an interesting set of processes, that describe different ways of eating at a restaurant. For example, the visit to a full-service restaurant is a process ordering–cooking–serving–eating–paying. But there are more ways to visit a restaurant (that is, to implement the composite service restaurantVisit), as shown in table 2. To speak consistently from the perspective of the service-provider, we use the terms take order, cook, serve and collect. We also assume that anything serves is eaten, so we leave out eating in the following.

Table 2. Ways to visit restaurants (after [WL95])

Restaurant	Service flow
Full service	take order–cook–serve–collect
Fast food	cook–take order–collect–serve
Buffet	cook–take order–serve–collect
Church supper	collect–take order–cook–serve

All these processes do have the same external interface void visitRestaurant(Money wallet, Order whatToEat) perhaps with being repleted=TRUE as a postcondition. When selecting a restaurant for an evening, however, that interface is most obvious – the actual

choice is on the internal structure of the service. The quality of services that is of interest to the user therefore is not solely the external interface, it can also be the internal behavior.

The current description mechanism are not sufficient to allow this for clients. There are two main reasons for this:

- Several languages do not specify the internal behavior with a service description at all (eg. WSCL, WSMF).
- The means to specify the internal service-flow are low-level and allow only the expression of the control flow. This addresses only the syntactic structure of the process and leaves no room to have multiple semantically equal processes fulfill the specification. The only exception in this respect is WSFL which can express dataflows.

We could say that precision and recall are low when describing processes by their functional interfaces only. Returning to the restaurant example, "visiting a restaurant" is an *abstract* process that is implemented by various concrete ones, the table above shows four of them.

If we want a restaurant service where the food is cooked freshly after we order it, only Full service and Church supper can satisfy our request, while Buffet and Fast food cannot. The "cooked fresh" service is both an implementation of the abstract restaurant visits, but at the same time an *abstract* process which is implemented by the two named processes. The specification, that we want cooking to occur after ordering discriminates it from the other two processes that we do not want.

In the following, we propose to introduce notions of *abstraction* and *specialization* as relations amongst composite Web Services and the notion of *dependencies* to abstractly specify the internal behavior of composed services.

3 Specifying and Relating Processes

We now look closer at the notions of dependency and specializations.

3.1 Dependencies

The semantical construct we use above for the "cooked fresh" service is that what is cooked *depends* on what is ordered. Also, the start of cooking depends on the end of ordering. The dependencies imply that the control flow in the implementation of such a service will reach order before cook. This makes, however, no statement on whether we pay at the beginning of our visit or at the end. If necessary, we could specify this by stating a dependency of collect on serve. All four composite services described above can be specific by stating that we want to eat something cooked, serve depends on cook.

[MC94] observes that dependencies are the basis on which processes are coordinated and defines coordination as the *management of dependencies*. This is currently one of the most accepted notions of coordination.

How the dependencies are managed depends on the coordination mechanism applied. If multiple entities depend on the availability of some resource, a central coordination

mechanism could be applied that selects one entity to get a lock on the resource. Another mechanism would be a market-like coordination where entities would have to bid for the resource. If no dependency amongst activities exist, the scheduling can be arbitrary – if we only want a place where food is cooked fresh after we ordered, we do not care when we have to pay.

In our example, we use a temporal dependency between ordering and cooking. In the beginning, we have hidden another dependency for simplicity – the food has to be served to be eaten. That dependency expresses that the food has to be transferred to the customer and that the dependency is resolved when is it placed on his desk.

[Cro91,Del96] have studied kinds of dependencies and associated mechanisms. Table 3 shows the main kinds of dependencies distilled by these studies. If a resource is shared, entities can depend on exclusive access to the resource. Some resource allocation mechanism manages that dependency, for example by scheduling, locking etc. If two entities exchange information the work of the consumer depends on the ability to use that information, for example by understanding the format and semantics of the data. Standardization is a mechanism that tries to resolve that dependency by globally understandable data formats. See the references for more elaborate analysis.

Table 3. Coordination mechanisms and managed dependencies [Cro91,Del96]

Coordination mechanism	Dependency managed	
Resource allocation		Shared resources
Notification Transportation Standardization	Prerequisite Transfer Usability	Producer/ Consumer
Synchronization		Simultaneity
Goal selection Decomposition		Task/Subtask

[RG00,RG01] introduce dependencies as a modeling concept to describe scenarios for testing software components. The dependencies modeled and the mechanism to manage them – specific styles of execution order – are shown in table 4.

From these results – there are certainly more studies on dependencies – we see that the usual focus on a control flow focuses on a secondary aspect. A concrete control flow describes the result of a mechanism that manages dependencies. There are many ways to find such a result, eg. solving a set of constraints by a central scheduler or a decentralized mechanism like bidding for a resource like a CPU. Also, there are many concrete schedules to manage some dependency. If, for example, a and b depend on some resource r, the control flows a.b and b.a (the dot means sequential composition here), are equally valid results of managing this dependency by establishing exclusive access.

Table 4. Modeling concepts for dependency charts [RG00,RG01]

Dependency class	Dependency
Temporal	Strict sequence
	Real-time
Causal	Loose sequence
	Data dependency
	Resource dependency
Abstraction	Generalization/Refinement
	Aggregation

Coordination Mechanism	Execution order
Sequence	Sequence
Alternative	Choice
	Conditioned choice
Iteration	Loop
	Conditioned loop
Concurrency	Accidental
	Enforced
	Prohibited

3.2 Abstraction and Specialization

The above example talks about processes in three levels of abstractions: 1) the concrete processes in restaurants, 2) the abstract specification of "freshly cooked" and 3) the most abstract notion "restaurant visit".

For computational processes, sound notions for the formal specification of behavior exist. For business processes, however, "qualities" like "freshly cooked" become important that are hard to capture. For the "computation" "visit a restaurant", all behaviors shown above are computationally equivalent. With the intention that one has when expressing abstractions on business processes, other notations are necessary.

[MCL$^+$99] mentions two dimensions when analyzing processes. The most common view is to talk about the *parts* of a process, that is the sub-activities that have to be taken. In our example, this refers to the respective services that compose the restaurant visit. Another, equally important dimension is the *type* of process. In our example, this refers to the grouping of Full service and Church supper into the type "freshly cooked". The levels of abstraction mentioned then lead to a hierarchy of specializations and generalizations of processes.

[WL95] approach specialization concepts for processes. They develop specialization and refinement transformations and the respective generalization and abstraction transformations. With these, hierarchies of processes can be derived. The kinds of restaurant visits can be generalized into a generic restaurant visit which includes the union of all possible orderings of services. With specialization then, new processes can be derived.

Still, for business processes, not all possible specializations of the most abstract service "do something" are useful ones. It remains to the modeler to put the focus on interesting, useful and possible processes.

3.3 Typing Web Services

The typing concepts for Web services are currently not very elaborate. The notions of port- and service-types are quite similar to the notions of interfaces in object-based standards such as OMG/CORBA. There, interface-types are related by specialization and generalization and a formal notion of a contravariant subtyping of interfaces.

We expect that such mechanisms are equally usable for Web Services and will enable some sort of Web Services trading. At such a trader, one would request a service of some interface/service type and get a reference to a service with a compatible interface/service type.

This well known mechanism of service-discovery addresses only the syntactic aspects of what a client expects from the service. The contract between the client and the server concerns only the kind and format of data exchanged but says nothing about what the service does.

We propose that dependencies are used as an additional information about the internal workings of a service. It could be provided together with the service type description and can be taken into account during service discovery. The description of internal service characteristics by dependencies is complementary to the description of external characteristics by interfaces.

Abstraction and specialization serve several purposes:

- During discovery, clients can express abstract expectations on the dependencies ruling the workings of the service. The service found during service discovery will observe these dependencies, perhaps some additional ones (see section 5.2).
- The level of detail in the description can be more abstract or more concrete depending on how much information the service provide is willing to disclose. All descriptions along the abstraction/specialization hierarchy are, however, valid descriptions of the service.
- Abstraction and specialization express a relative semantic of what the services do. This semantic is explicitly provided by the service description, but there are also option for deriving it automatically (see section 5.3).

4 A Dependency Markup Language

Starting with some example processes that are similar we claimed that the notions of dependencies and generalization/specialization are better suited to describe composite services than just functional interfaces or just a specific control flow.

For Web Services, we propose a *Dependency Markup Language* (DML) which provides the necessary language to express both. The resulting specification is more abstract than a concrete control flow and a more specific service description than a functional interface.

Figure 1 shows the structure of DML in terms of defined elements. A DML description consists of a set of dependency type declarations and a set of process descriptions. Each process contains a set of dependency descriptions.

Figure 2 shows an excerpt of the respective XML Schema to document attributes. Dependency types declare a name for them and also can be related by a specializes-declaration.

In the current version, each dependency in a process connects two services (from and to). It can refer to the events of starting or ending the services. The type-Attribute defines the kind of dependency. Processes them self can be related by specialization.

With that, we can put our knowledge on restaurants into a DML file as shown in figure 3.

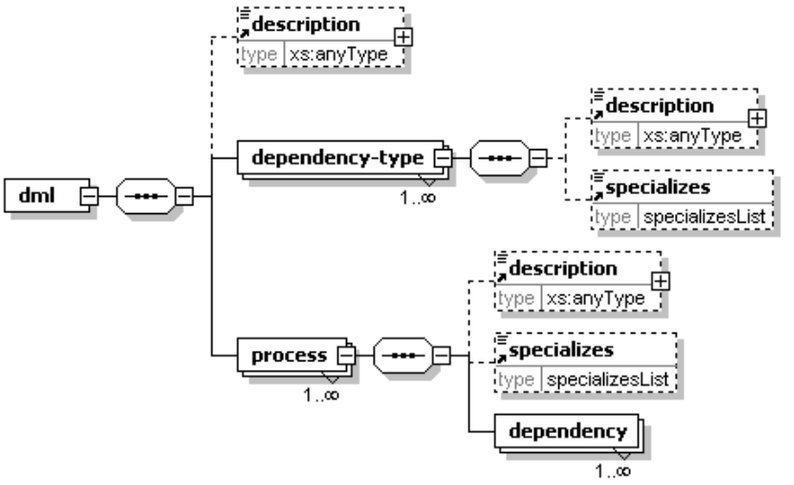


Fig. 1. The structure of DML

5 Processing DML

Given a description of a composite service with DML, there are several ways to process it. We foresee a *coordination environment* for DML that contains several services that perform functionalities as described in the next subsections.

5.1 Coordinating Web Services

The dependency-type specifications in DML carry no information about how the dependencies are managed. As seen in 3.1, there are several ways to manage dependencies resulting in different execution orders of services.

The coordination environment for DML contains *coordination services* that *bind* themselves to dependency types they are able to manage. Based on the information given by the DML specification, a coordination service generates a specific service service flows relative to the specification languages for composite services mentioned in the beginning.

To ensure openness and interoperability of these coordination services, a dependency ontology will be necessary. Currently, it is built by the specialization hierarchy within DML. The next step would be the design of an open dependency ontology on the basis of DAML+OIL which would be closely related to approaches like DAML-S. Figure 4 shows the DML notation for the dependency types mentioned in section 3.1.

5.2 Discovering Web Services with DML

In the coordination environment *dependency matchers* are able to determine whether a control flow is a specialization of a process. A concrete control flow is nothing than a

```
[...]
<xs:attributeGroup name="identification">
 <xs:attribute name="id" type="xs:ID" use="required"/>
 <xs:attribute name="name" type="xs:string"/>
</xs:attributeGroup>
<xs:attributeGroup name="specialization">
 <xs:attribute name="specializes" type="xs:anyURI" use="optional"/>
</xs:attributeGroup>
<xs:simpleType name="event">
 <xs:restriction base="xs:string">
  <xs:enumeration value="end"/>
  <xs:enumeration value="start"/>
 </xs:restriction>
</xs:simpleType>
<xs:attribute name="servicereference" type="xs:anyURI"/>
<xs:simpleType name="specializesList">
 <xs:list itemType="xs:anyURI"/>
</xs:simpleType>
<xs:element name="specializes" type="specializesList"/>
[...]
<xs:element name="dependency-type" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="description" minOccurs="0"/>
   <xs:element ref="specializes" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="identification"/>
  <xs:attributeGroup ref="specialization"/>
 </xs:complexType>
</xs:element>
[...]
<xs:element name="process" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="description" minOccurs="0"/>
   <xs:element ref="specializes" minOccurs="0"/>
   <xs:element name="dependency" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attribute name="name" type="xs:string"/>
     <xs:attribute name="type" type="xs:anyURI" use="required"/>
     <xs:attribute name="from" type="xs:anyURI"/>
     <xs:attribute name="from-event" type="event" default="end"/>
     <xs:attribute name="to" type="xs:anyURI"/>
     <xs:attribute name="to-event" type="event" default="start"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="identification"/>
  <xs:attributeGroup ref="specialization"/>
 </xs:complexType>
</xs:element>
[...]
```

Fig. 2. An excerpt from the DML schema

```
[...]
 <process id="restaurantVisit" name="Visit to a restaurant">
  <description>
   An abstract description of a restaurant visit where only
   cooked food is eaten.
  </description>
  <dependency type="looseSequence" from="cook" to="serve"/>
 </process>
 <process id="freshlyCooked" specializes="restaurantVisit">
  <description>
   An abstract description where things are cooked after an order.
    </description>
  <dependency type="looseSequence" from="takeOrder" to="cook"/>
 </process>
 <process id="fullService" specializes="freshlyCooked">
  <dependency type="strictSequence" from="takeOrder" to="cook"/>
  <dependency type="strictSequence" from="cook" to="serve"/>
  <dependency type="strictSequence" from="serve" to="collect"/>
 </process>
 <process id="fastFood" specializes="restaurantVisit">
  <dependency type="strictSequence" from="cook" to="takeOrder"/>
  <dependency type="strictSequence" from="takeOrder" to="collect"/>
  <dependency type="strictSequence" from="collect" to="serve"/>
 </process>
 <process id="buffet" specializes="restaurantVisit">
  <dependency type="strictSequence" from="cook" to="takeOrder"/>
  <dependency type="strictSequence" from="takeOrder" to="serve"/>
  <dependency type="strictSequence" from="serve" to="collect"/>
 </process>
 <process id="churchSupper" specializes="freshlyCooked">
  <dependency type="strictSequence" from="collect" to="takeOrder"/>
  <dependency type="strictSequence" from="takeOrder" to="cook"/>
  <dependency type="strictSequence" from="cook" to="serve"/>
 </process>
[...]
```

Fig. 3. The restaurant example in DML

dependency specification, only at a less abstract level. Since coordination mechanisms are able to generate such a specialization by applying coordination mechanism, they can also determine whether they can generate a given process (they might not be able to falsify this) as a specialization of a certain DML specification. [WL95] contains a basic set of the necessary relations to do that.

With that, one can implement an enhanced discovery of Web Services that take into account dependencies in addition to any functional interfaces. Such a service would resolve the problems mentioned in the introduction.

5.3 Classifying Processes

In DML the specialization hierarchy has to be specified explicitly. There seem to be more options for an automatic detection of such relations and an automatic classification of

```
[...]
<dependency-type id="any"/>
<dependency-type id="mit-dependency" specializes="any"/>
<dependency-type id="sharedResources" specializes="mit-dependency"/>
<dependency-type id="producerConsumer" specializes="mit-dependency"/>
<dependency-type id="prerequisite" specializes="producerConsumer"/>
<dependency-type id="transfer" specializes="producerConsumer"/>
<dependency-type id="usability" specializes="producerConsumer"/>
<dependency-type id="simultaneity" specializes="mit-dependency"/>
<dependency-type id="taskSubtask" specializes="mit-dependency"/>
<dependency-type id="unizh-dependency" specializes="any"/>
<dependency-type id="temporal" specializes="unizh-dependency"/>
<dependency-type id="strictSequence">
 <specializes>temporal looseSequence</specializes>
</dependency-type>
<dependency-type id="realTime" specializes="temporal"/>
<dependency-type id="causal" specializes="unizh-dependency"/>
<dependency-type id="looseSequence" specializes="causal"/>
<dependency-type id="dataDependency" specializes="causal"/>
<dependency-type id="resourceDependency" specializes="causal"/>
<dependency-type id="abstraction" specializes="unizh-dependency"/>
<dependency-type id="generalization" specializes="abstraction"/>
<dependency-type id="refinement" specializes="abstraction"/>
<dependency-type id="aggregation" specializes="abstraction"/>
[...]
```

Fig. 4. Dependencies from tables 3 and 4 in DML

composite services on that basis. Works like [Nie95,MHK98,ZG00] have studied such kind of typing of processes.

For our example, it can be detected that fullService contains the dependency that is specified for freshlyCooked since the dependency from takeOrder to cook exists. The type of the dependency in fullService is strictSequence which is a specialization of looseSequence according to our hierarchy of dependencies. From that, one could infer that fullService specializes freshlyCooked.

However, there are limits to such a classification. The dependencies still capture structural properties of processes. The processes under consideration implement some functionality. Even if two processes share no structural characteristics, they can still implement the same functionality and thus both specialize the same abstraction which might even be empty of dependencies.

6 Outlook

The DML specification is currently being finalized. On this basis, a coordination environment is to be build and tested. This includes the implementation of a set of coordination mechanisms, their binding to dependencies and the generation of control flows from DML. This also includes the implementation of dependency matchers as described.

The implementation of the mentioned services of the coordination environment seems to be straightforward. The coordination services transform DML specifications into formats for service composition and might even be implemented in XSL. Service discovery requires a repository of DML descriptions and an appropriate and simple lookup algorithm. A classification service needs the implementation of some more complex algorithms as mentioned in the above description.

The main obstacle to set up such a coordination environment is to make it rich in expressibility of dependencies. The dependency ontology has to be enlarged by further studies [Tol00]. It has to be checked how notions of dependencies can be related by specialization to lead to a unified hierarchy. It has to be tested how the mentioned mechanisms for automatic classification of processes can be applied for DML specifications and whether they lead to useful results. Multiparty dependencies are currently not considered and have to be explored. Furthermore, the dynamic change of dependencies is a topic for future research.

References

[ABH+02] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci andTerry Payne, and Katia Sycara. DAML-S: Web Service Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *The Semantic Web – ISWC 2002*, volume 2342 of *LNCS*, pages 348–363. Springer-Verlag, 2002.

[Agr01] Ashish Agrawal, editor. *Business Process Modeling Language (BPML) Specification*. Business Process Management Initiative, 2001.

[AMS02] Selim Aissi, Pallavi Malu, and Krishnamurthy Srinivasan. E-Business Process Modeling: The Next Big Step. *IEEE Computer*, 35(5):55–62, May 2002.

[BBB+02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web Services Conversation Language (WSCL) 1.0. W3c note, World Wide Web Consortium, 2002. http://www.w3.org/TR/wscl10/.

[Bus01] Business Process Team. ebXML Business Process Specification Schema. Technical report, UN/CEFACT and OASIS, 2001. http://www.ebxml.org/specs/ebBPSS.pdf.

[Cro91] Kevin Ghen Crowston. *Towards a Coordination Cookbook: Recipes for Multi-Agent Action*. PhD thesis, Sloan School of Management, MIT, 1991. CCS TR# 128.

[Del96] Chrysantos Nicholas Dellarocas. *A Coordination Perspective on Software Architecture: Towards a Design Handbook for Integrating Software Components*. PhD thesis, Massachusetts Institute of Technology, 1996.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. Technical report, Vrije Universiteit Amsterdam, 2002.

[Ley01] Frank Leymann. Web Services Flow Language Web Services Flow Language Web Services Flow Language Web Services Flow Language (WSFL 1.0). Technical report, IBM Software Group, 5 2001.

[MC94] Thomas W. Malone and Kevin Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[MCL+99] Thomas W. Malone, Kevin Crowston, Jintae Lee, Brian Pentland, Chrysanthos Dellarocas, George Wyner, John Quimby, Charles S. Osborn, Abraham Bernstein, George Herman, Mark Klein, and Elissa O Donnell. Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Management Science*, 45(3):425–443, 3 1999.

[MHK98] Max Mühlhäuser, Ralf Hauber, and Theodorich Kopetzky. Typing Concepts for the Web as a Basis for Re-use. In Anne-Marie Vercoustre, Maria Milosavljevic, and Ross Wilkinson, editors, *Proceedings of the Workshop on the Reuse of Webbased Information*, Report Number CMIS 98–111, pages 79–89. CSIRO Mathematical and Information Sciences, 1998.

[Nie95] Oscar Nierstrasz. Regular Types for Active Objects. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, chapter 4, pages 99–121. Prentice Hall, 1995.

[RG00] J. Ryser and M. Glinz. Using Dependency Charts to ImproveScenario-Based Testing. In *Proceedings of the 17th International Conference on Testing Computer Software (TCS2000)*, Washington D.C., 6 2000.

[RG01] J. Ryser and M. Glinz. Dependency Charts as a Means to Model Inter-Scenario Dependencies. In G. Engels, A. Oberweis, and A. Zündorf, editors, *Modellierung 2001*, volume P-1 of *GI-Edition – Lecture Notes in Informatics*, pages 71–80, 2001.

[Tha01] Satish Thatte. XLANG. Web Services for Business Process Design. Technical report, Microsoft Corporation, 2001.

[Tol00] Robert Tolksdorf. Models of Coordination. In Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agent World First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000*, number 1972 in LNAI, pages 78–92. Springer Verlag, 2000.

[WL95] George M. Wyner and Jintae Lee. Applying Specialization to Process Models. In *Conference on Organizational Computing Systems*, Tools, pages 290–301, 1995.

[ZC02] Mladen A. Vouk Zhengang Chang, Munindar P. Singh. Composition Constraints for Semantic Web Services. In *Proceedings of the International Workshop Real World RDF and Semantic Web Applications 2002*, 2002. http://www.cs.rutgers.edu/~shklar/www11/.

[ZG00] Michael Zapf and Kurt Geihs. What Type Is It? A Type System for Mobile Agents. In Robert Trappl, editor, *Proceedings of the 15th European Meeting on Cybernetics and Systems Research*, pages 585–590. Austrian Society for Cybernetic Studies, April 2000.

Web Based Service for Embedded Devices

Ulrich Topp[1], Peter Müller[1], Jens Konnertz[2], and Andreas Pick[2]

[1] ABB Corporate Research, Wallstadter Straße 59, 68526 Ladenburg, Germany,
{ulrich.topp,peter.o.mueller}@de.abb.com
[2] Institut für Automatisierungs- und Softwaretechnik, University of Stuttgart, Germany,
konnertz@ias.uni-stuttgart.de

Abstract. Field devices are becoming more and more 'intelligent'. This report describes how one can use this in conjunction with approved and upcoming internet technologies and standards to overcome many problems arising due to the fact that every manufacturer has at least one way to communicate to his devices. The presented case study utilizes two approaches: One is an embedded web server with CGI capability, the other is an embedded application of the SOAP protocol.

1 Introduction

In industrial automation systems many small devices are located in the field of operation, like sensors for various physical measurements or actors to perform some action on the process . This is why we call them "field devices". In a normal plant there are several standards for such devices to communicate with some controlling computer and/or the operator station. Among these standards are Profibus, Foundation Fieldbus, CanBus, and others. These are suitable for control networks with distances below 500m. In other situations field devices are used in a locally wide spread application, like in power networks or along a pipeline. Thus the user needs a way to operate and maintain his equipment remotely. In addition, often several devices are grouped together, and these devices are not necessarily from the same manufacturer. Here the urgent need is, to have a common way of communication in order to avoid multiple access infrastructure. This paper describes the use of standard internet technologies to achieve remote service and operation capability. Two proposed solutions enable the user to survey his device remotely and to discover needed maintenance tasks prior to an urgent need, often signalled by total failure of the device. Using standard technologies, like dynamical generated html and SOAP, based on TCP/IP as transport protocol we describe the needed supplier independence. In future systems, there will be one local IP-network and one remote connection to the operator and the maintenance staff.

As ABB is introducing its new Industrial-IT platform as basis for all automation software, this paper also includes the description how this work fits into ABB'S software strategy.

Accordingly, this paper starts with an discussion of today needs and trends (sec. 2), describes the applied technologies and prototypes (sec. 3) and concludes with some cost estimation and an outlook (sec. 4 and 5).

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 141–153, 2003.

2 Motivation and Scope

2.1 The Situation Today

The classical service for field devices is done by a service engineer going into the field and visiting and maintaining every part of the equipment manually. A step forward is the usage of remote service capabilities offered by connecting to the device using a modem line. This gives the engineer the possibility to examine remotely the status of the device and to be better prepared if a visit is necessary. Often it will be enough to change some parameters or just to reset the device to ensure a proper function until the next maintenance visit. The benefit of this approach is limited in situations where in one place several different devices are located, often from several manufacturers. The limitation occurs due to usage of proprietary protocols to communicate and to the fact that the devices often could not be connected in a coherent network. Here the usage of standards concerning the networking (e.g. TCP/IP) and the higher level of communication (Internet protocols like http) could bring a major advantage in order to facilitate or even enable remote communication. This will allow a much better service and life cycle management.

2.2 Trends from Outside

A recent NEXUS report [1] on technology roadmaps in industrial instrumentation highlights two technology trends that will dominate the future of process instrumentation a) Microsystems technology or Micro Electromechanical Systems (MST/MEMS) and b) Internet technology. Especially the combination of both will result in "Smart Systems" comprising additional functions for monitoring, diagnosis, asset management, (self-) configuration and offering new opportunities and possibilities such as remote sensing, remote access and advanced diagnostics.

Despite the importance of accuracy, performance and cost of devices, user needs are becoming motivated by a new set of drivers that may overshadow the traditional attributes that previously drove the demand for field devices and sensors. These drivers, lead by field networks, include asset management software and the growing need for Web enabled communication, may reverse the trend of slow growth of field devices and continue the prosperity of discrete sensors [2]. In addition more and more requirements arise to have a better integration that spans not only process and alarm data but integrating all aspects of a device starting from nameplate data, maintenance records, documentation, configuration, alarm handling and of course operation and monitoring. Users believe that information assets from smart devices are key to trouble-shooting instrumentation problems and reducing maintenance costs. By identifying devices that need attention in lieu of the typical preventive maintenance program based on time in service, they save costs of unnecessary maintenance. Invensys and Endress+Hauser are actively researching device failure modes/analysis to develop sophisticated embedded high-level sensor diagnostics. Invensys in an alliance with the University of Oxford, formed the Invensys University Technology Center (UTC) to research areas such as advanced sensors, condition monitoring and self-validation, known today as SEVA. The goal of Invensys is to incorporate algorithms into their process field devices to validate sensors, improve control loop tuning and overall industrial processes. UTC has patented

methods to provide data that will continuously update the health of the instrument and tune instruments back into calibration due to minor faults [3].

Placing remote I/O and small PLCs with IP/Ethernet connections on the shop floor is currently economical, and is becoming quite popular with systems integrators (FF → FF-HSE, ControlNet → EtherNet/IP, Profibus → PROFINet, Interbus → IDA ...). ARC predicts that over the next five years, IP/Ethernet will dominate all field connections except for the final connection to the lowest level sensors and actuators [4]. As devices begin to communicate over the Internet using TCP/IP or UDP/IP protocols allowing users to view and manipulate data using a standard browser, a new device category, Internet Appliances, is emerging. From our point of view the need of contribution to these trends seems obvious. ABB has to be an early adopter of advanced technologies especially in the case that the advantage over conventional ones is that evident.

The following table (table 1) summarizes the pro's and con's of the different approaches.

Table 1. This is an overview over important categories, which could be used to compare the usefulness of different software approaches. Proprietary protocols, as used in the past, clearly have a advantages in resources because they are tailored for one special application. But concerning the reuse of software, interoperability and AIP integration only solutions based on standards will be cost effective and successful.

	Conventional, proprietary protocol	webserver with dynamic html	webserver with SOAP services
Interoperability	–	+	+
User Interface	– (client side generated by hand)	+ (web browser)	O (client side generated with tool support)
Resources	+	O	–
Reuse of components	–	O	+

2.3 IndustrialIT

In 2001 ABB announced a new software platform for all its products in order to enable a global integration of all information, be it static or dynamic. The following gives a (very) short overview over some base principles of IndustrialIT.

The Idea of IndustrialIT is to model the real world objects in a manner similar to how we as human beings look at them: We see the object and depending on the context and on our role respective to this object we recognise and use different characteristics of it. This approach leads to a software platform, the Aspect Integrator Platform (AIP), where all real objects are modelled through abstract objects which initially has only some generic and basic functionality (name, type information and so on). Special characteristics are added through adding 'Aspects'. These are made up of pieces of software implementing some functions or implementing an interface to an external data source or software package. For example, structural information such as the physical location of the (real)

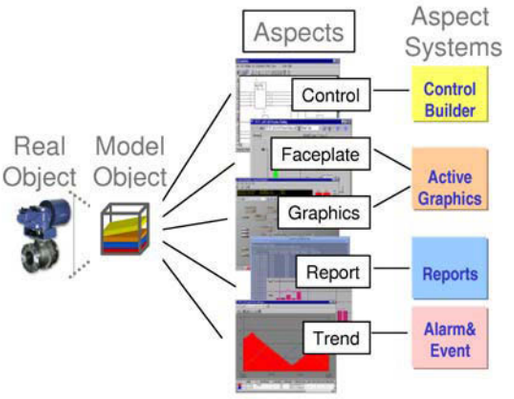


Fig. 1. The principles of Industrial-IT: The real world objects are modelled through aspect objects, to which functionality is added by means of Aspects. These Aspects are interfaces to Aspect Systems, which in turn implement the appropriate function

object in the plant or the functional placement in the process or the logical arrangement in a production order is added through 'Structure Aspects', forming a multidimensional grid, where the system and the user could freely navigate. Figure 1 shows several other possibilities to add functionality using Aspects. In addition, this platform is designed in a way, that it is fully network transparent, allowing to access the information of the objects independently of the users location. Goal of this approach is to integrate all information sources and make them available in realtime to every user whatever role in the plant he plays, wherever he is located, and whenever he needs the information.

2.4 Conclusion and Proposal

Now we are in the situation that we see our demands and we could proceed to the solution, which was studied in the past few months. Two main paths have been under study. The first (see fig. 2.4) consists of an embedded web server, which offers a HTML based user interface and allows the user to do device control, diagnosis and monitoring tasks using a standard internet browser or an Aspect System, integrating the devices functionality into the AIP.

The second development path (see fig. 2.4) uses the SOAP protocol to implement several ways of communication and remote procedure call, thus opening the way towards an nearly unlimited communication with and controllability of the device.

3 Proposed Solution: Embedded Web Service Plus SOAP

Before we go into the details of the technologies used, we clarify the scenarios where remote service is needed. After that the test case is discussed, a device for gas chromatography, and the last two sections cover the two main development paths, we worked out.

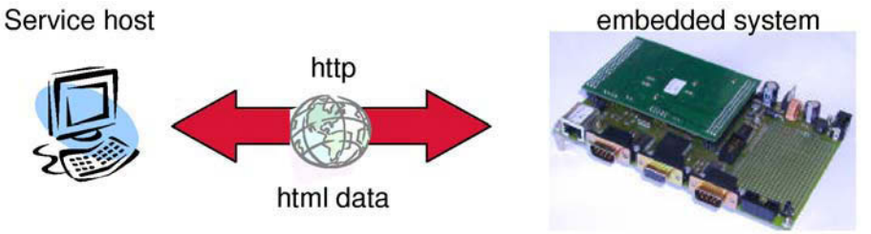


Fig. 2. System topology with the operators PC, the service host, and the embedded system, where a web server provides device information and operation capabilities through both static and dynamic generated html pages. On the service host a standard web browser could be used as smple user interface as well as a AIP operator station, integrating the device into a larger system.

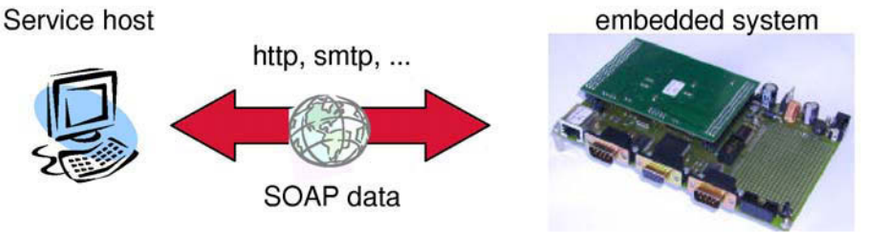


Fig. 3. System topology where the embedded system offers its services using SOAP, thus providing a much more powerful and generaic way to access measurement and status data from the device. This approach s not limited to a http connection but could also be used with the email protocol SMTP. Obviously, the next logical step would be to combine both paths bringing the enormous flexibility of SOAP communication into the IndustrialIT platform.

3.1 Typical Remote Service Scenarios

This subsection presents typical deployment scenarios applicable to all field devices. In detail these are:

- **Operation / Monitoring:** The user is able to perform a particular measurement (in case of a sensor) or control task (in case of an actor) and to capture the resulting values.
- **Diagnosis / Monitoring:** This module takes some measurements concerning the status of the device. For example environmental measures like temperature, humidity, voltage of power supply and so on. The user is able to capture the resulting time series.
- **Configuration:** The user is able to manipulate remotely a wide range of configuration parameters concerning the operation of the device. The parameters are permanently stored locally on non-volatile memory.
- **Alarms:** Inform maintenance staff about errors or send out maintenance triggers (e.g. battery voltage reaches a critical low level). The device is configured to generate

alarms when certain conditions occur. The alarm is transmitted to the user by means of e.g. e-mail.

For all four kinds the feasibility has been tested. Figure 4 shows the principal data exchange and messages sent between the service host and the embedded device.

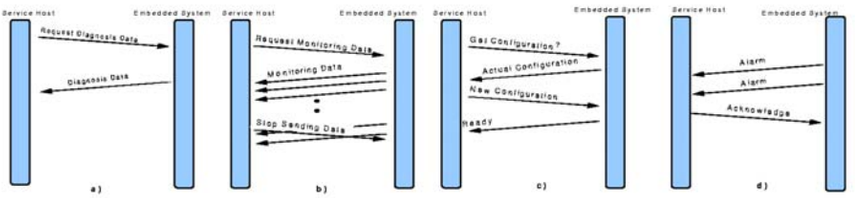


Fig. 4. The different scenarios for remote service: a) Remote Diagnosis b) Remote online monitoring c) Remote Configuration and d) Remote alarm handling.

In general there are three kinds of communication services necessary:

- **Request / Response:** This kind of communication can be used to request diagnosis data or the device configuration
- **Subscription:** With this kind of communication callbacks can be realised. This is typically used to retrieve process data without forcing a client to poll. Note that the role of the client and server changes after the subscription was initiated.
- **Spontaneous:** The role of the server and the client has changed to the request / response scenario. Here the client starts communication without request from the service host. This is typically used for sending alarms.

3.2 Test Case: The Gas Chromatograph on a Pipeline

Gas Chromatography (GC) is an analytical technology employed within ABB for general process control as well as for dedicated metering and custody transfer applications in the natural gas market. Whereas in the first market, ABB serves customers with the specification and setup of customized and very specialized gas chromatographic instruments, the second is a market, where the application is ever identical, namely BTU-Measurement of natural gas, and strategic high focus is placed on cost, sevicability and diagnostics for remote installations as well as general robustness of the system. Such devices are located along pipelines for natural gas in order to monitor the quality of the gas. This serves as the described scenario of locally very spread devices which need to be operated and serviced.

In general the GC consists of 4 functional blocks:

- Temperature stabilization and control: The measurement is valid only if defined climatic environment is guaranteed. Thus PID controllers are employed to stabilize as well as ramp temperature in certain functional regions of the GC.

- Flow stabilization: The measurement is valid only if defined flow of the gas under test is guaranteed. Thus a PID controller is used to stabilize this gas flow. A defined sample volume is pushed through the system with a carrier gas. Two such carrier gas streams are to be controlled in flow.
- Valve control: The measurement is controlled by switching a 9-open/close-valve-system through four different states.
- Main measurement: The uGC splits the gas components in a way that they pass one after the other over the measurement cell, a thermal conductivity detector. Here the deviation of thermal conductivity (measured as voltage) is employed to measure the quantitative admixture of a component to the carrier gas. Different components are identified through their arrival time in the detector.

Every block has its own set of parameters like the two constants defining the linear relationship between the measured voltage and the physical property of a PID control loop.

The microprocessor in use has a twofold task. First, it has to control the operation of the device as described above. Second, it has to provide the 'virtual faceplate' offered as a web service. (see fig. 5) This means all necessary control options and parameters has to be handled through a web interface, be it a web server with HTML and CGI, see sec. 3.3, or a SOAP server, see sec. 3.4.

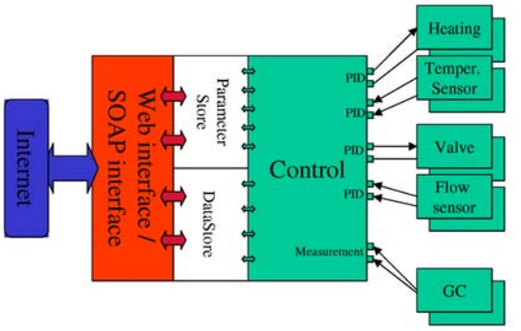


Fig. 5. General architecture of the embedded system. The control part covers the sampling of sensor values and updating the actuators according to the set parameters as well as the complete measurement cycle. The web/SOAP interface part handles the connection requests from the service host/operator station. Both parts are connected through two types of 'stores. The ParameterStore holds all parameter infromatiion, while the DataStore serves as a small database of historical data.

3.3 Development Path 1: Embedded Web Server with CGI Interface

A web server responds to a client request always in the same manner: It sends a block of text over the internet, mostly consisting of HTML-code describing the content of the answer and its formatting. The HTML-code could be located statically at the server or

could be generated dynamically by the server. The first way is chosen for information which is likely not to change and the second, obviously, for presenting some data from the server, which may change in time. In particular variable data like configuration parameters and, of course, process values fall in this category.

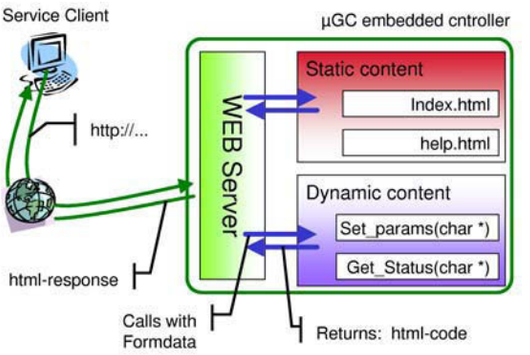


Fig. 6. Architecture of the embedded Web-server. The http-request from the service client (top left side) is processed by the web server. Depending on the requested document reference, html-code from the static content pool is returned, or the server calls a registered object method, evtl. with additional form data. This method in turn generates the html-respinse on the fly incorporating some real time values of the system.

This branch of the study implements a web server on the embedded controller, which uses both ways: The portal to the service and some information and help texts are statically located in the embedded systems non volatile memory.

From the portal page of the device the user is linked to the various services, shown in the following pictures (fig. 7:)

- Operational Status (with buttons to initiate some actions)
- Configuration (Overview over different sections and as an example the configuration of a PID controller)
- Diagnosis (As graphical representation as well as textual, in order to export it to mathematical applications)
- Measurement Data is up to now provided as textual information.
- Documentation

This implementation make it very easy to integrate the device in an IndustrialIT environment. The AIP provides standard aspect categories to connect to web servers and to show up the HTML information. Figure 8 shows an example scenario for the integration. The aspect object for the GC is equipped with some aspects linked to the appropriate web pages for configuration, diagnosis and so on (see the context menu in the figure). Some basic information needed by the AIP system, like the IP address of the embedded controller, are stored in an standard aspect of type 'General Properties'.

Fig. 7. Views of the different services: Top left shown is the configuration page of the measurement; top right the operational status of the device. On the bottom the output of a java applet is shown with the data of the temperature and gas flow controllers.

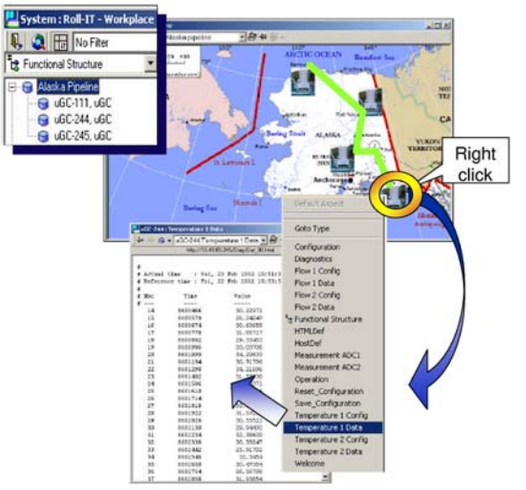


Fig. 8. The GC embedded controller in a AIP system. The functional structure in the top left corner shows a pipeline object and below three GC devices. The pipeline object has a graphical map aspect associated. This map shows the pipeline in Alaska together with four locations of gas chromatographs. Right-clicking evokes the according context menu, providing the user all available aspects of the device, allowing to navigate e.g. directly to the temperature data of one of the devices.

The next step of integration would be not only to display the process values as text but to fetch them into the system as binary value. The way to achieve that, is through a

simple program written in Visual Basic, converting the ASCII representation of values sent by the embedded system into numeric variables. Through the mechanisms of the platform these variables could be published as OPC values, thus opening the way to use the build-in analysis features of the platform, like trend displays, historians, and so on. This step is not taken because of the much more powerful possibilities the usage of a SOAP based service offers, which we will outline in the next section.

3.4 Development Path 2: Advanced Solution with Embedded SOAP

Short walk through SOAP and WSDL technology. If an application aims to provide a broad range of services and interfaces, the implementation must avoid proprietary methods. It assures the maintainability as well as the interoperability, if one uses accepted standards. In the field of services offered via internet connections, the SOAP [5] standard is the way to go. SOAP is an acronym for 'Simple Object Access Protocol' and is a XML based specification for message based communication and for remote procedure call (RPC) communication. It allows to evoke specific function calls on the server from internet connected clients (see fig. 9. What SOAP makes it that powerful is the way function parameters are serialized, that means how they are transformed from the client system specific binary format to a completely system independent format. Furthermore SOAP allows the server to respond to the client not only a short status message but also every kind of data, which could be serialized following the same rules as above.

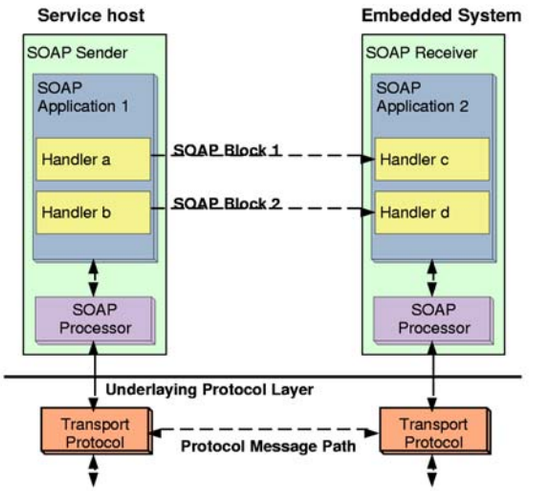


Fig. 9. The basic principles SOAP handles remote procedure calls. The service host is able to access functions, the embedded system has opened for remote use. The packing and unpacking of parameters is handled by the SOAP engine and is fully transparent to the application

Once you have implemented a SOAP server, the only issue is to make its services, that are the available function calls, public to all possible clients. For this purpose the Web Service Description Language (WSDL) was introduced by the W3 Consortium [6]. This language has exactly the above needed purpose: The standardized description of services, a site offers to clients via the internet. It is not bound only to SOAP services but most SOAP toolkits offer helper programs to generate automatically the WSDL file derived from the source code (see fig. 10. Or the other way around: The developer starts with a WSDL file, specifying the service, and the tool generates the skeleton code for a SOAP server/client in the desired programming language (e.g. C++ or Java).

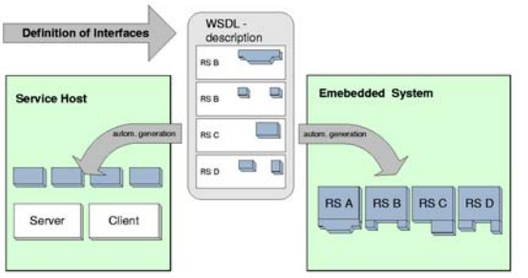


Fig. 10. The Web Service Description Language WSDL is used for an abstract definition of remote services (RS). It opens the way for automatic generation of code skeletons for the SOAP client as well as for the SOAP server

Being able to make functions calls on a low level, it will be possible to achieve a deep integration between the software running on the embedded device and the AIP server system. This includes that the corresponding Aspect System makes use of most of the ABB-Aspect and -Object related operations like navigation, change notification, and so on. All this will be highly maintainable and reusable through the use of the open SOAP standard.

Sample implementation. The given timeframe allowed us to implement two of the above service scenarios (see Sec. 3.1): The remote configuration of a PID controller and an alarm service (see fig. 11). The first is the classical case of client-server communication, but the second reverses the roles: The embedded system is now the client, which searches a service host to post an alarm message. Since this is not to be modelled in a normal web-server architecture here the use of SOAP is essential.

4 Technical and Cost Comparison

The two approaches show different advantages and, as shown in the Table 1 (page 143), are applicable in several scenarios. This picture becomes clearer, if we consider the costs in terms of memory consumption. (CPU performance becomes more and more a less

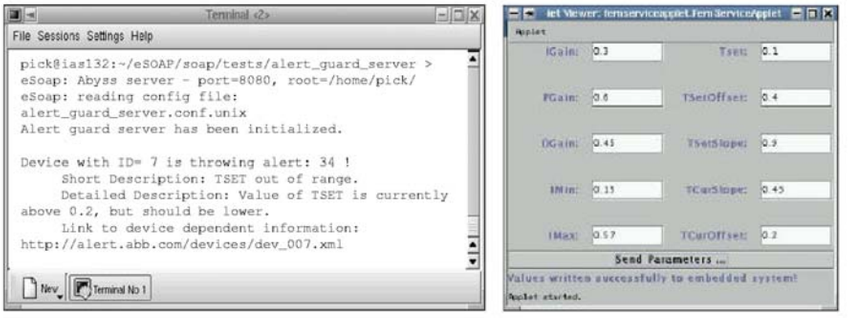


Fig. 11. Two simple SOAP applications on the service host: Left hand is the alert guard server, which shows up an error, the embedded system reported. And on the right side a Java applet shows the configuration form for a PID controller and in the bottom line the response from the embedded system indicating the successful transmission.

critical boundary condition.) The implementation of the two approaches, described in sections 3.3 and 3.4, have a very different resource allocation.

- The embedded web server with CGI interface needs around 24kByte code size compared to ca. 43kByte for the control part of the software.
- The implementation with SOAP is much larger, together with some helper libraries the used eSOAP toolkit uses around 400kByte.

In both cases there are in addition some runtime memory needs depending on the used TCP stack. As an important remark, we state that we had not the time to apply any optimisation strategy. This means, we used the different toolkits as they are offered in the public domain. This leaves room for enhancement especially, if we search for commercial solutions for the TCP stack, or the SOAP toolkit, which might be memory and resource optimised in order to fit also in smaller systems. Reducing the memory needs by factor of 2 to 5 seems possible.

Anyway, the above numbers classify the two approaches for different target applications: In cases, where memory is expensive, as in small devices like simple sensors, one would prefer the less powerful solution. This will enhance the product with key features regarding operation and maintenance without making it too expensive. But there are many applications, where the device consists of more than a sensor, and in order to handle all parts a larger controller or even a small PC is employed. Then memory is not an issue because in such systems RAM is counted in Megabytes rather than in kilobytes.

5 Conclusion and Outlook

The use of state of the art internet technologies will allow a much deeper integration of field devices with the AIP than today possible. In addition, the usage of world approved standards assures interoperability with other third party products and software. The test

case of the gas chromatograph clearly shows how the service capabilities could benefit from the power, which is today offered by modern microprocessors, which are already installed in the device in order to operate it. The studied scenario is obviously not limited to this GC application. The results are applicable to all devices, which are equipped with microprocessors and a network connection. Since future strategy (e.g. .net framework) is strongly coupled with the use of SOAP, this study could be used as starting point for further projects, targeting the implementation of a generic service description for embedded measurement and control devices, in order to improve significantly remote operation and maintenance capabilities.

Next steps of research include 1) Identification other potential devices and systems for the remote service capabilities 2) Thorough implementation of the different scenarios together with AIP integration 3) Further employment of the SOAP strategy and of course with increasing importance 4) Incorporate security issues like authentification or data encryption

References

1. http://www.emsto.com/NEXUS/
2. ARC Advisory Group, Field Device and Sensor Strategies for the E-World, February 2001
3. Invensys UTC for Advanced Instrumentation, http://seva.eng.ox.ac.uk/
4. ARC Advisory Group, Software Trends for Automation, December 2000
5. http://www.w3.org/TR/soap12-part1/
6. http://www.w3.org/TR/wsdl

Using Ontology to Bind Web Services to the Data Model of Automation Systems

Zaijun Hu

ABB Corporate Research Wallstadter Straße 59, Ladenburg, Germany,
zaijun.hu@de.abb.com

Abstract. Supplying integrated data model with multiple structures for addressing diverse requirements from different production lifecycles and control levels is the natural development of automation systems. Data processing, presentation and manipulation require well-designed computation model that consists of interrelated groups of function components or applications that can be realized in the form of web services. Binding the available web services to the designed data model becomes a challenging task if a great amount of web services is confronted. In this paper we will use ontology technology to address the binding problem. We will present the required ontology model including the formal expression of ontology, object model, mapping to XML representation and the corresponding system architecture for binding web services.

Keywords: Ontology, web service, automation system

1 Introduction

Automation systems are becoming more and more complex, integrated and multiple-functional to deal with various needs from the production lifecycles including purchase, design, engineering, operation, etc and the different control levels ranging from the field device layer to ERP layer. To simply the development of such a system the design of data model is usually decoupled from the building of application logics. The separation of data modeling from the creation of the computational model that provides diverse applications or components for using, processing and viewing the data described in the data model facilitates, on the one hand, the development of automation systems in flexibility, extensibility and distributed development paradigm. But on the other hand it also, at the same time, creates problems such as heterogeneity in the implementation platform. Some applications use Java for the implementation, and some others use COM from Microsoft as implementation platform. Because of the platform- and language-neutrality web services are gradually showing the undoubted advantage in addressing the heterogeneity problem. With the development of more and more web services the binding or the association of the developed web services processing, viewing and using data to the corresponding data described in the data model is becoming one of new challenges. In this paper we are going to present an ontology method to address the binding problem.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 154–168, 2003.

2 Industrial IT – ABB Aspect Integrator Platform

ABB Aspect Integrator Platform (AIP) is created for the integration of different information hierarchies – device layer, automation layer, manufacture execution and asset management layer, enterprise management layer and collaboration layer, and it also creates an integration platform for integrating diverse functional applications and components. The information model defined in AIP provides an unified modeling mechanism conform to IEC 61346 to organize or structure the information entities in an integrated automation system that contain plants, equipments, components, devices, instruments, sensors, actuators, controller etc. The basic elements in the model are Aspect Object and Aspect. An **Aspect Object** in the model is a container that holds different parts of an object in an automation system. Examples for an object can be a reactor, a pump or a node (computer). An **Aspect** is one 'piece' of data and operations that is associated with an object. Typical aspects of an (process) aspect object are its control program, operator faceplate, trend configuration, function specification etc. Figure 1 shows an example for presenting the AIP architecture concept:

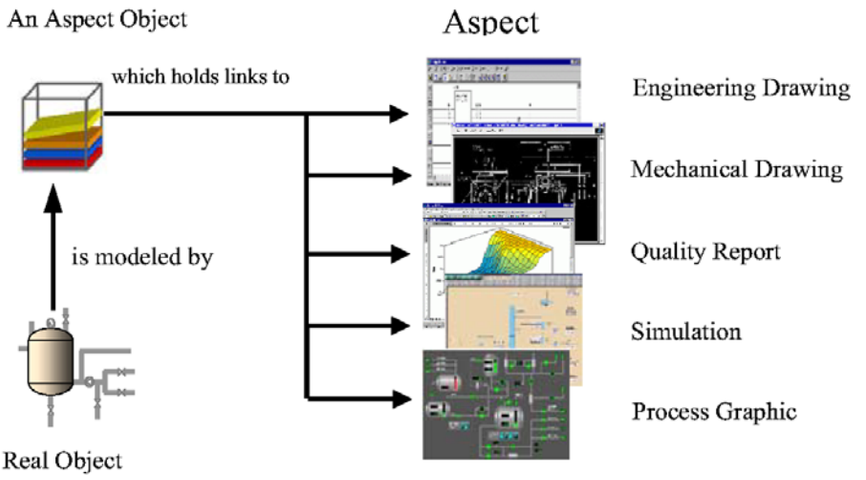


Fig. 1. An example for AIP architecture concept

A structure defined in the AIP architecture is conform to the one in IEC 61346 and defines semantic relationships among the organized objects and also rules for creation of the semantic relationship. In IEC 61346 a structure is separated from the objects and represented through an additional aspect. In this way an object can be organized in different structures at the same time. IEC 61346 presents three examples of information structures that are important for the design, engineering, operation and maintenance – function-oriented struc-

ture, location-oriented structure and product-oriented structure. Each structure is represented through a defined hierarchy.

The function-oriented structure organizes objects based on their purposes or functions that are played in a system. The structural hierarchy is created through the subdivision of the functions or the purposes of objects. The function-oriented structure is the result of design and is interesting for engineering, operation and maintenance. The location-oriented structure defines the topological relationship among objects. The structural hierarchy of a location-oriented structure results from definition of spatial constitution relationship, e.g. ground area, building, floor, room etc. The location-oriented structure is useful for the engineering, operation, maintenance and so on. The product-oriented structure shows the breakdown of product-related information for a given product type. A product can consist of many components and each component can be made up of further subcomponents. This constituent relationship determines the product structure hierarchy. All structures are hierarchical, i.e., the parent-child relationships among objects determine the hierarchy. Figure 2 shows two structures – function- and location-oriented structure.

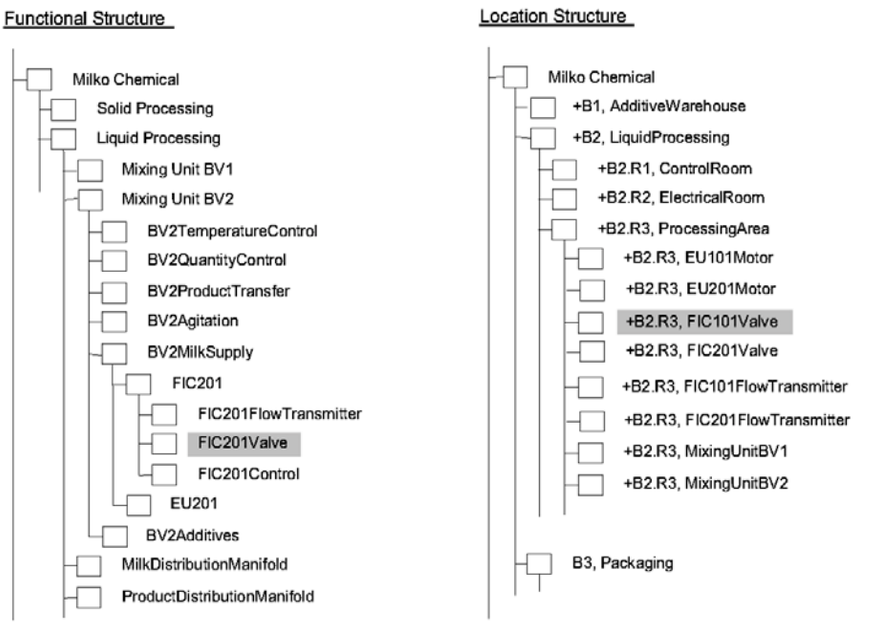


Fig. 2. Example for functional and location structure

An aspect can be realized with web services or traditional software applications and components. The AIP platform encourages the development paradigm

of separating modeling of aspect objects from the developing aspects. That means the developed aspects that determine the computation model of an automation system should be associated to the corresponding aspect objects. The AIP just provides an example for the background of this paper.

3 Some Definitions

In this paper we will use some definitions that build the discussion basis of this paper.

3.1 Ontology

"An ontology defines the terms used to describe and represent an area of knowledge"[4]. The purpose of ontologies is to share information in a certain context. Ontologies include computer-usable definitions of basic concepts in the context and the relationships among them. They encode knowledge in a context and also knowledge that spans the context. In this way, they make that knowledge reusable. In this paper ontologies are used to establish association between the data model of a domain on the one side and the operations in the form of web services on the other side. In other words they create a matching and binding mechanism between data and its operations.

3.2 Domain Ontology and Operation Ontology

A domain ontology is an ontology for a certain domain. A domain is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc. In our context a domain refers directly to an industrial process such as energy generation and the related components such as power plant, gas turbine, boiler, steam turbine, generator etc. that take part in an process execution. An operation ontology is an ontology for data operations. It defines terms to describe and represent the knowledge in the data operation area, e.g. simulation, mechanical drawing, visualization, optimization. The differentiation between the domain ontology and the operation ontology is necessary to better deal with the association of data operation with data.

3.3 Data Model

A data model in the context of this paper is used to describe and represent an industrial process with the related components and the control of the process with all necessary units. Figure 3 shows an example of the data model:

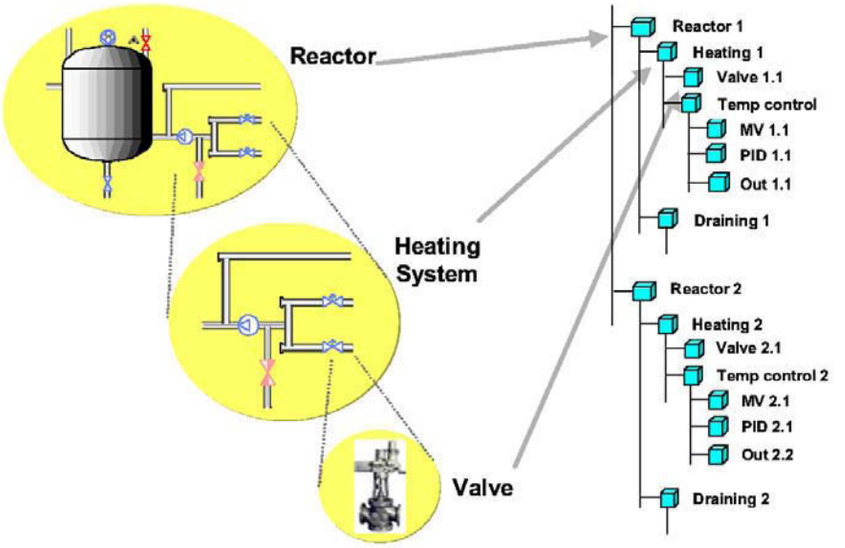


Fig. 3. An example of data model

3.4 Computation Model

A computation model defines structure, categorization and classification of data operations that deal with the operations of data such as drawing, simulation, cost calculation, report, visualization etc. It contains a group of related functions for operating data. Data operations are built in the form of software applications, components, and in this paper in the form of web services. A computation model can use the operation ontology to describe the structure, categorization and classification of operations.

3.5 Binding

After defining the data model and the computation model we can begin with the binding definition. A binding is a process that associates a computation model containing a group of functions to the corresponding data model. The binding is to find the most suitable functions that can be used to the data of interest or to provide a guideline to users for the association. What we are interested in this paper is the binding of web services that implement functions based on the web service protocol to the related data. Through the ontology technology we can convert the operation binding to the ontology binding, namely binding of the operation ontology to the domain ontology.

4 Ontology Model

4.1 Expression

To give a mathematic expression of an ontology we use the atomic concept term and composite concept term defined in [6]. Let V =$\{v_0, v_1, \ldots, v_n\}$ be the universal set of atomic concept terms that can be used and R = $\{r_0, r_1, \ldots, r_m\}$ be the set of all relationships between the terms, V' be a subset of V (V' $\subseteq$ V, V' = $\{v'_0, v'_1, \ldots, v'_k\}$) and R' be defined as subsets of R (R' = $\{R_i' \mid R_i' \subseteq R \wedge i=1,\ldots, k)$, I = $\{I_i | i=1, \ldots, k\}$ be a piece of information, the ontology Ψ can be expressed by $\Psi=\{ \Psi_i \mid i=1, \ldots, k \}$ where Ψ_i is an ontology element and can be expressed by a tuple $<v_i, R'_i, I_i, S>$ with $v_i \in V \wedge R'_i \in R' \wedge I_i \in I$. S in the tuple stands for the structure discussed in the Section 2 that is interesting in the context of this paper. We associate a piece of information to each atomic concept term to give some explanatory information on it. For example we can take an example of DC motor, such information can be explanation of the principle, the deployment area, main features and so on. This would be a great help for users that are not familiar with the meanings of the term.

4.2 Relation

Is-A Relation
Is-A relation is generalization relation. It is a binary relation and thus needs two terms taking part in the relation. We use symbol ϕ_{Is-A} (v_i, v_j) where i, j =1,..., k to signify the generalization relationship between $v_i$and v_j, or $v_i$is a generalization of v_j. An example of generalization relation is DC motor and electric motor. Electric Motor is a generalization of DC motor, expressed by ϕ_{Is-A} (Electric Motor, DC motor). Is-A relation has properties of reflexivity and transitivity, namely

- ϕ_{Is-A} (v_i, v_i) is true
- ϕ_{Is-A} $(v_i, v_j) \wedge \phi_{Is-A}$ $(v_j, v_l) \rightarrow \phi_{Is-A}$ (v_i, v_l)

Is-A relation can be used to organize the concept terms in a hierarchical structure. Imposing hierarchical structure on the concept terms can obtain precision increase substantially [13]. Another benefit of that is the information reuse, because the information associated to the generalized concept term can be used by its child. The Is-A relation also enhances possibility of the correct binding.
For the Is-A relation it is necessary to take into account the height of hierarchy levels. Figure 4 illustrate the concept:

For the Fig. 4 the relations ϕ_{Is-A} (A, B_1), ϕ_{Is-A} (A, B_2), ϕ_{Is-A} (B_1, C_1), ϕ_{Is-A} (B_1, C_2), ϕ_{Is-A} (A, C_1), ϕ_{Is-A} (A, C_2) hold. The difference between ϕ_{Is-A} (A, B_1) and ϕ_{Is-A} (A, C_1) is that B_1 has a direct Is-A relation to A while C_1 has an indirect Is-A relation to A via B_1. We call ϕ_{Is-A} (A, B_1) the Is-A relation of the first order and ϕ_{Is-A} (A, C_1) the Is-A relation of the second order and express them with ϕ_{Is-A} $(A, B_1)^1$ and ϕ_{Is-A} $(A, C_1)^2$. Is-A relation

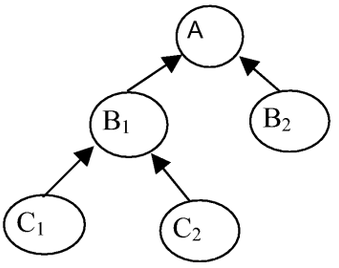


Fig. 4. Term hierarchy

is very important in inferring some possible operations that can be bound to certain data.

Sibling Relation
The sibling relation is used to describe the siblings in the Is-A relation hierarchy.

It can be expressed by

$\phi_{sibling}\ (\mathrm{v}_i,\ \mathrm{v}_j)|\exists \mathrm{k}\phi_{Is-A}\ (\mathrm{v}_k,\ \mathrm{v}_i)^1 \quad \wedge \phi_{Is-A}\ (\mathrm{v}_k,\ \mathrm{v}_j)^1$

The sibling relation is reflexive, transitive and symmetric.

Specific Relation
As an inverse part of the Is-A relation we use the specific relation for expressing

the specialization of a concept term. For example DC motor is a specialization of electric motor. The specific relation is also a binary relation requiring two terms. We use symbol $\phi_{spec}\ (\mathrm{v}_i,\ \mathrm{v}_j)$ where i, j =1,. . . , k to denote the specific relationship between $\mathrm{v}_i$and v_j, or $\mathrm{v}_i$is a specialization of v_j. Between the specific and Is-A relation the following implication holds

$\phi_{Is-A}\ (\mathrm{v}_i,\ \mathrm{v}_j) \leftrightarrow \phi_{spec}\ (\mathrm{v}_j,\ \mathrm{v}_i)$

Just as the Is-A relation the specific relation is also reflexive and transitive. It is to note that $\phi_{Is-A}\ (\mathrm{v}_i,\ \mathrm{v}_j) \quad \neq \neg\phi_{spec}\ (\mathrm{v}_i,\ \mathrm{v}_j)$. The specific relation defined here is different from the definition in [6] that also includes the synonym relation. In our definition we don't consider the synonym relation because including synonym could result in complexity in the structure of the concept terms and redundancy. We can also use the specific relation to structure the concept terms just as the Is-A relation. The question is if it is necessary to include the specific relation in the ontology model.
Just like the Is-A relation we also introduce the height of hierarchy levels and express it with

$\phi_{spec}\ (\mathrm{v}_i,\ \mathrm{v}_j)^n$.

The expression can be read as the specific relation of the nth order.
In the binding process the specific relation plays the same role in the identification of possible operations as the Is-A relation.

Part-Whole Relation
The part-whole relation describes the composition or containment relation between two concept terms. For example the relation between room and walls is such one. The part-whole relation is also a binary one needing two terms. We introduce symbol ϕ_{part} (v_i, v_j) to indicate that $v_i$is a part of v_j. The part-whole relation is reflexive and transitive. Just as the Is-A relation the Part-Whole relation also has order and can be expressed by ϕ_{part} (v_i, v_j)^n. Part-Whole relation can be used to infer some possible operations that can be associated to the corresponding data. The relation can be used to determine if it is possible to bind an operation with a data object or node when the data object has the part-whole relation to another one.

Buddy Relation
The buddy relation is defined by ϕ_{buddy} (v_i, v_j)$|\exists k$ ϕ_{part} (v_i, v_k)$^1 \wedge \phi_{part}$ (v_j, v_k)^1. The buddy relation is used to describe the relation between terms that share the same container or are parts of same terms. For example floor and wall have the buddy relation because they both are parts of room. The buddy relation is reflexive, transitive and symmetric. Like the part-whole relation the relation can also be used for the binding.

Synonym Relation
A term can have synonyms that have the same meanings as the term itself. In the binding process a data object in the data model represented by a term in the domain ontology can be associated with operations represented by a term in the operation model if both terms have the synonym relation. The synonym relation can be expressed by
$\phi_{Synonym}(v_i, v_j)$

The synonym relation is reflexive, transitive and symmetric.

Theorem 1. *A data object or data node (D) represented by a term in the domain ontology (TD_1) can be bound with an operation (Op) represented by a term in the operation ontology (TO_2) if (TO_2) is bound to (TD_2) and $\phi_{Synonym}$ (TD_1, TD_2) holds.*

Disjoint Relation
A term has a disjoint relation with another term if both terms don't contain shareable semantic. The disjoint relation can be expressed by $\phi_{Disjoint}(v_i, v_j)$.

The Disjoint relation is symmetric.

Theorem 2. *A data object or data node (D) represented by a term in the domain ontology (TD_1) cannot be bound with an operation (Op) represented by a term in the operation ontology (TO_2) if (TO_2) is bound to (TD_2) and $\phi_{Disjoint}$ (TD_1, TD_2) holds.*

Binding Relation
A binding relation describes that two terms are bound together in a certain way. It can be expressed by $\phi_{Binding}$ (v_i, v_j). The binding relation is reflexive, transitive and symmetric.

4.3 Object Model of an Ontology

The object model of an ontology describes a model that provides an way of mapping the ontology to memory for programmatic access, navigation and manipulation. For the ontology modeling we use MOF (Meta Object Facility) [7] proposed by OMG because MOF is widely accepted in the industry and uses the layered concept (data, model, metamodel, metametamodel) that is very suitable in our addressed context. For the simplicity reason we just adopt three layers from the MOF: data, model and metamodel. The metamodel of an ontology consists of two parts: the term model and the relation model. The term model describes the meanings of terms. It is just like a dictionary that contains explanation of vocabularies. The term model defines four meta-classes: TermCollection, TermElement, Term, TermExplanation. TermCollection is a collection of TermElement that contains two elements: Term (Vocabulary, Id) and TermExplanation. The meta-class TermCollection also has an attribute called Structure that is introduced to deal with the issue of multiple structures in the data model discussed in Sect. 2 and 3.3. Instantiated from the metamodel the model layer is created for the modeling of the domain ontology and the operation ontology. The created classes for the model layer and the metamodel for the domain ontology are displayed in Fig. 5. The same instantiation can also be applied for the operation ontology. For the simplicity reason the instantiated relation model for the operation ontology is omitted here.

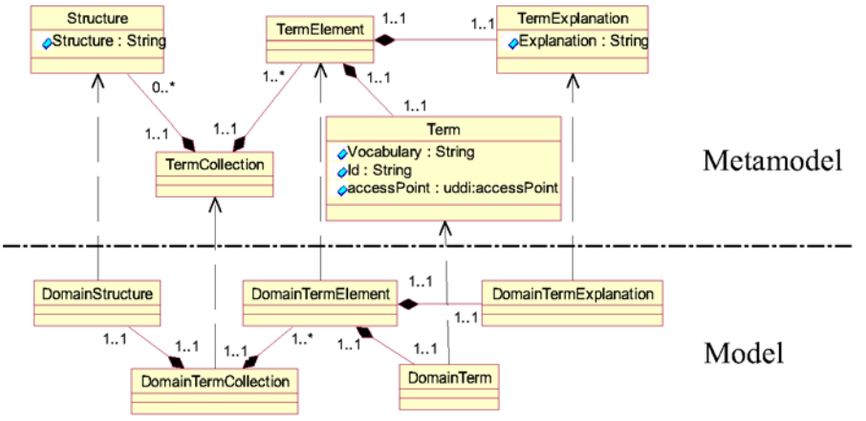


Fig. 5. The term model with the metamodel and the model layers

The relation model describes the relations among terms that are defined in Sect. 4.2. For the relation model two meta-classes are defined for the metamodel layer: RelationCollection and RelationElement. The RelationCollection is a collection of RelationElement that associates two terms. The RelationCollection also defines some operations that can be applied to the class. The operation can be classified into three categories: retrieval, check, and change. The retrieval operations are ones that can be used to retrieval terms that have specified relations to the specified term. The check operations check if both specified terms have the specified relation. The change operations are exploited to change the collection such as adding, removing, or changing relations. The metamodel for the relation model can be instantiated to the model layer of the relation model. In this paper the model layers for the domain ontology, the operation ontology and the binding relation between the domain ontology and the operation ontology are instantiated. Figure 6 shows the diagram illustrating the metamodel layer and the model layer of the relation model. The diagram only displays an instantiation example for the relations in the domain ontology.

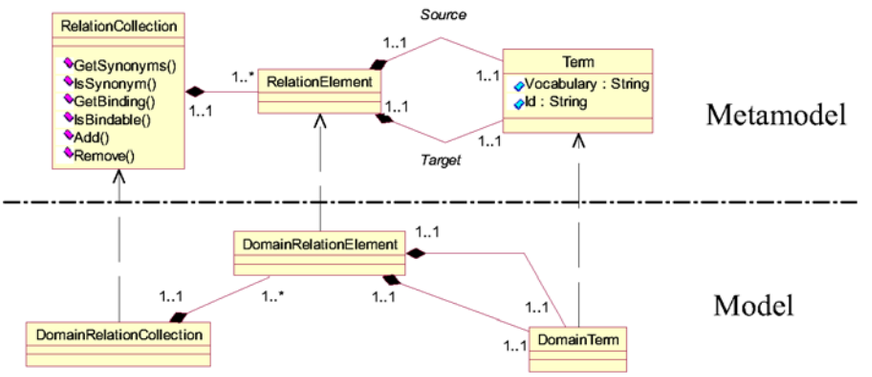


Fig. 6. The relation model with the metamodel and the model layers

4.4 XML Representation

The object model discussed in 4.3 just creates a precondition for the memory mapping of an ontology. It describes the basic elements of the model, their relationships and semantics. But it doesn't define the way of how the elements are structured in the memory, how data is validated against the model layer, how the model layer is checked against the metamodel layer, how to access and to navigate the model. XML is already a widely accepted markup language that is neutral to different platform and diverse programming language. What is more important of using XML is the semi-structural feature, i.e. XML provides not only a way of putting rule, structure and relationship to data but also flexibility

in extension of data structure. Because of the convincing advantages of XML we will also use it for representing the ontology object model. Figure 7 shows an XML schema file that maps the term model to the XML representation. The figure illustrates the concept on how we use XML to describe the metamodel layer and the model layer of the ontology object model. In XML schema there is no instantiation mechanism for the metamodel instantiation. Therefore we resort to another mechanism, i.e. substitution introduced in XML schema. First of all we defined XML elements: TermCollection, TermElement and Term, which are abstract. That means the elements must be replaced in the instance document with the defined substitution elements that are defined through "substitutionGroup". In this way we emulated the instantiation mechanism.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="TermCollection" type="TermCollectionType" abstract="true"/>
 <xsd:complexType name="TermCollectionType">
   <xsd:element name="TermElement" type="TermElementType" minOccurs="1" maxOccurs="unbounded" abstract="true"/>
   <xsd:element name="Structure" type="StructureType" minOccurs="0" maxOccurs="unbounded">
   <xsd:element name="accessPoint" type="string" minOccurs="0" maxOccurs="unbounded">
 </xsd:complexType>
 <xsd:complexType name="TermElementType">
  <xsd:element name="Term" type="TermType" minOccurs="1" maxOccurs="1" abstract="true"/>
  <xsd:element name="TermExplanation" type="TermExplanationType" minOccurs="0" maxOccurs="1"/>
 </xsd:complexType>
 <xsd:complexType name="TermType">
   <xsd:attribute name="Vocabulary" type="xsd:string" use="required"/>
   <xsd:attribute name="Id" type="xsd:string" use="required"/>
 </xsd:complexType>
 <xsd:complexType name="TermExplanationType">
   <xsd:attribute name="Explanation" type="xsd:string" use="required"/>
 </xsd:complexType>
 <xsd:element name="DomainTermCollection" substitutionGroup="TermCollection"/>
 <xsd:element name="DomainTermElement" substitutionGroup="TermElement"/>
 <xsd:element name="DomainTerm" substitutionGroup="Term"/>
 <xsd:element name="OperationTermCollection" substitutionGroup="TermCollection"/>
 <xsd:element name="OperationTermElement" substitutionGroup="TermElement"/>
 <xsd:element name="OperationTerm" substitutionGroup="Term"/>
</xsd:schema>
```

Fig. 7. XML schema for the term model

5 Web Service and System Architecture

In [9] Myerson presented different web service models with the so called architecture stack model from different organizations such as WebServices.Org, IBM, Oracle, Microsoft, Sun, W3C and so on. The architecture stack model describes the layers such as the transport layer, the message layer, the description layer, the discovery layer, the web service flow layer, and the negotiation layer. What are relevant for the context of this paper are the description layer described by WSDL (Web Service Description Language) and the discovery layer that can be specified through the UDDI specification, a common initiative from IBM, Microsoft and so on. UDDI provides a way to publish and discover information about web services. But UDDI does not define mechanism to deal with ontology and how to bind web services representing the computation model to the target data. Therefore an addition concept or extension and the corresponding component are required. Figure 8 shows the concept for the extension. There are

a domain ontology component that describes the term structure of the domain side and is attached to the data model of interest and an operation ontology component that determines the term structure of the operation side and is connected to the web service component providing web services. Each web service is identified through an URL address. In UDDI the URL address is contained in the accessPoint XML element while in the WSDL it is embedded in the Port XML element. The accessPoint and Port element uses the same URL address to reference the same web service. Therefore they can be used to connect the UDDI and WSDL. The relation between the data model and the web service is established by the ontology service that is the central component in the architecture. The component checks the potential web services based on the binding relation between the domain ontology and the operation ontology. Once it finds a matched web service it will bind the web service to the target data.

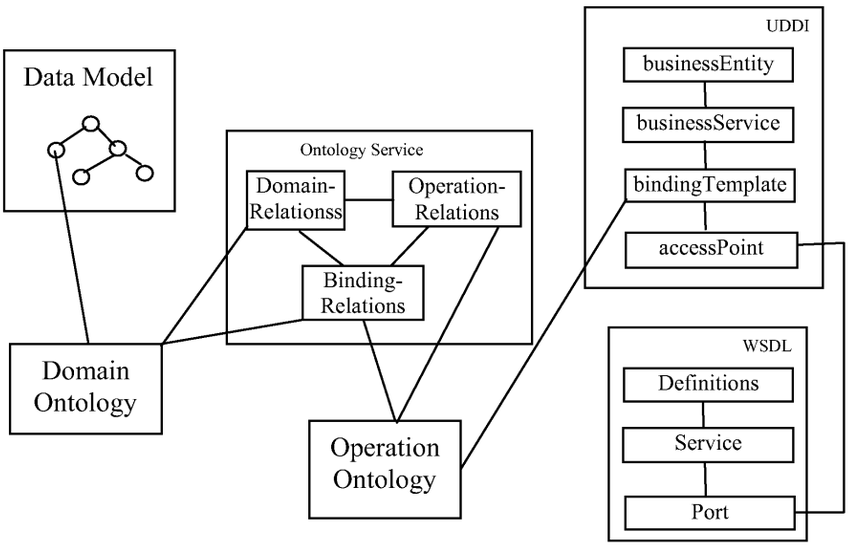


Fig. 8. System architecture for web service binding based on ontology

6 Example

For illustration purpose we take power plant as an example for the domain side and simulation for the operation side or the web service side.
The domain ontology and the operation ontology can be expressed in XML (Fig. 10). For the space reason we don't list all elements in the example. The two XML document will be stored in the memory as two separate document instance.

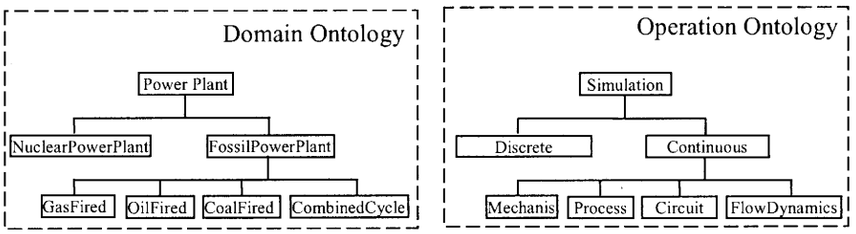


Fig. 9. Example for domain and operation ontology

```
<?xml version="1.0" encoding="utf-8" ?>
<DomainTermCollection>
  <DomainTermElement>
    <DomainTerm Vocabulary="PowerPlant"
Id="1"/>
  </DomainTermElement>
  <DomainTermElement>
    <DomainTerm Vocabulary="NuclearPowerPlant"
Id="2">
  </DomainTermElement>
  <DomainTermElement>
    <DomainTerm Vocabulary="FossilPowerPlant"
Id="3">
  </DomainTermElement>
  <DomainTermElement>
    <DomainTerm Vocabulary="CombinedCycle"
Id="4" accessPoint="Plant1">
  </DomainTermElement>
.....
</DomainTermCollection>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<OperationTermCollection>
  <OperationTermElement>
    <DomainTerm Vocabulary="Simulation"
Id="1"/>
  </OperationTermElement>
  <OperationTermElement>
    <OperationTerm Vocabulary="Discrete"
Id="100">
  </OperationTermElement>
  <OperationTermElement>
    <OperationTerm Vocabulary="Continuous"
Id="101">
  </OperationTermElement>
  <OperationTermElement>
    <OperationTerm Vocabulary="Process"
Id="102"
accessPoint="http://128.122.343.23/ps.asmx">
  </OperationTermElement>
.....
</OperationTermCollection>
```

Fig. 10. Domain and operation ontology in XML

These two documents can also be seen as two thesauruses that can be used for the binding engineering or the binding process. The relations of the domain ontology and the operation ontology are contained in two separate XML documents: the domain relation document and the operation relation document (Fig. 11). These two documents can be used to get all related terms for the specified terms. For example in the XML document for the operation relation model you can find which element is child element of the "Continuous". For the binding purpose the binding relation is used that can be expressed in the Fig. 12. In the XML document for the binding relation the term FossilbePowerPlant is associated with Simulation while the term "CombinedCycle" is attached to "Process". This binding relation is created in the ontology engineering, i.e. through ontology expert. Users who want to use simulation program don't need to know which term of the domain ontology is attached to the corresponding operation ontology. He only needs to tell the ontology service that he needs a simulation service. The ontology service will find the most suitable simulation service, in our example the process simulation service.

```
<?xml version="1.0" encoding="utf-8" ?>
<DomainRelationCollection>
  <DomainRelation Type="IsA">
    <DomainTerm Vocabulary="FossilPowerPlant"/>
    <DomainTerm Vocabulary="CombinedCycle" />
  </DomainRelation>
</DomainRelationCollection>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<OperationRelationCollection>
  <OperationRelation Type="IsA">
    <OperationTerm Vocabulary="Continuous"/>
    <OperationTerm Vocabulary="Process" />
  </OperationRelation>
</OperationRelationCollection>
```

Fig. 11. Domain and operation relations in XML

```
<?xml version="1.0" encoding="utf-8" ?>
<BindRelationCollection>
  <BindingRelation Type="Binding">
    <DomainTerm Vocabulary="CombinedCycle"/>
    <OperationTerm Vocabulary="Process" />
  </BindingRelation>
</BindingRelationCollection>
```

Fig. 12. Binding relations in XML

7 Conclusion

In this paper we presented an ontology method that can be used to deal with the binding problem, especially for the case of a great amount of web services. The main contribution in this paper is creation of the ontology model for the binding problem. Two kinds of ontology are identified to convert the binding problem to the connection of the domain ontology and the operation ontology. MOF model is used for definition of the object model. We have expressed the relations in formal form and then created the object model and the XML presentation. We also presented the system architecture and the web service model. Further work is required for the ontology engineering.

References

[1] Ragaller, K.: An Inside Look at Industrial ^{IT} Commitment, ABB Technology Day, 14 November 2001.

[2] Krantz L.: Industrial IT – The Next Way of Thinking, ABB Review 1/2000

[3] Lars G. Bratthall, Robert van der Geest, Holger Hofmann, Edgar Jellum Zbigniew Korendo, Robert Martinez, Michal Orkisz, Christian Zeidler, Johan S. Andersson: Integrating Hundred's of Products through one Architecture – The Industrial IT architecture. ICSE 2002

[4] International Electrical Commision (IEC). IEC 1346-1. Industrial Systems, installations and equipment and industrial products – Structuring principles and reference designations, First Edition, 1996

[5] Requirements for a Web Ontology Language, http://www.w3c.org

[6] Ling Feng, Manfred A. Jeusfeld, Jeroen Hoppenbrouwers. Beyond Information Searching and Browsing: Acquiring Knowledge from Digital Libraries. INFOLAB Technical Report ITRS-008, Tilburg University, The Netherlands, February 2001. http://citeseer.nj.nec.com/421460.html

[7] OMG. OMG Unified Modeling Language Specification Version 1.3. Technical report, Object Management Group Inc., March 2000
[8] W3C: XML Schema Part 0: Primer, 2001. http://www.w3.org
[9] Myerson, J. M.: Web Service Architecture, Published by Tect, 29 South LaSalle St. Suite 520, Chicago, Illinois, 60603, USA, http://www.webservicesarchitect.com.
[10] Ehnebuske D., Rogers D., Claus von Riegen: UDDI Version 2.0 Data Structure Reference UDDI Open Draft Specification 8 June 2001, http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf.
[11] Gruber., T. R.: A Translation Approach to Protable Ontology Specifications. Knowledge Acquisition, 5(2):199-220, 1993.
[12] Guarino, N., Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N.J.I. Mars (ed.), Towards Very Large Knowledge Bases, IOS Press 1995
[13] Guarino N., Masolo C., and Vetere G., OntoSeek: Content-Based Access to the Web, IEEE Intelligent Systems 14(3), May/June 1999, pp. 70–80
[14] Guarino. N.: The Ontological Level. Invited paper presented at IV Wittenstein Symposium, Kirchberg, Austria, 1993. In Casati, R., Smith, B. and White G. (eds.), Philosophy and the Cognitive Science, Vienna, Hölder-Pichler-Tempsky, 1994.

eXist: An Open Source Native XML Database

Wolfgang Meier

Darmstadt University of Technology
meier@ifs.tu-darmstadt.de

Abstract. With the advent of native and XML enabled database systems, techniques for efficiently storing, indexing and querying large collections of XML documents have become an important research topic. This paper presents the storage, indexing and query processing architecture of eXist, an Open Source native XML database system. eXist is tightly integrated with existing tools and covers most of the native XML database features. An enhanced indexing scheme at the architecture's core supports quick identification of structural node relationships. Based on this scheme, we extend the application of path join algorithms to implement most parts of the XPath query language specification and add support for keyword search on element and attribute contents.

1 Overview

eXist (http://exist-db.org) is an Open Source effort to develop a native XML database system, which can be easily integrated into applications dealing with XML in a variety of possible scenarios, ranging from web-based applications to documentation systems running from CDROM. The database is completely written in Java and may be deployed in a number of ways, either running as a stand-alone server process, inside a servlet-engine or directly embedded into an application.

eXist provides schema-less storage of XML documents in hierarchical collections. Using an extended XPath syntax [2, 6], users may query a distinct part of the collection hierarchy or even all the documents contained in the database. Despite being lightweight, eXist's query engine implements efficient, index-based query processing. An enhanced indexing scheme supports quick identification of structural relationships between nodes, such as parent-child, ancestor-descendant or previous-/next-sibling. Based on path join algorithms, a wide range of path expression queries is processed only using index information. Access to the actual nodes, which are stored in the central XML document store, is not required for these types of expressions.

The database is currently best suited for applications dealing with small to large collections of XML documents which are occasionally updated. eXist provides a number of extensions to standard XPath to efficiently process fulltext queries, including keyword searches, queries on proximity of search terms or regular expressions. For developers, access through HTTP, XML-RPC, SOAP and WebDAV is provided. Java applications may use the XML:DB API [18], a common interface for access to native or XML-enabled databases, which has been proposed by the vendor independent XML:DB initiative.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 169–183, 2003.

A growing number of developers is actively using the software in a variety of application scenarios. Applications show that eXist – despite its relatively short project history – is alread able to address true industrial system cases. For example, an italian automobile manufacturer currently distributes eXist as part of a multi-lingual technical documentation publishing system, which covers technical maintenance documentation for several car models.

Main contributions of this article are:

1. We provide a detailed description of the data structures, the indexing architecture and the query processing aspects implemented in eXist.
2. We show how an enhanced numbering scheme for XML nodes at the architecture's core could be used to implement efficient processing of complex path expression queries on large, unconstrained document collections. Contrary to other proposals, which focus on the efficient processing of ancestor-descendant relationships, our indexing scheme supports all axes of navigation as required by the XPath specification.
3. While previous contributions have indicated the superiority of path join algorithms over conventional tree traversals for a limited set of expressions [10, 15, 16], we extend the application of path joins to implement large parts of the XPath query language and add support for keyword search on element and attribute content.

The paper is organized as follows: The next section presents some details on the indexing and storage of XML documents as implemented in eXist. We will introduce the numbering scheme used at the core of the database engine and describe the organization of index and data files. Section 3 will then explain how the numbering scheme and the created index structures are used in query processing. In Section 4 we finally present some experimental results to estimate the efficiency and scalability of eXist's indexing architecture and query engine.

2 XML Indexing and Storage

This section takes a closer look at the indexing and storage architecture implemented in eXist. We will first provide some background information and then introduce the numbering scheme used at the core of the database.

2.1 Background

XML query languages like XPath [2, 6] or XQuery [4] use path expressions to navigate through the logical, hierarchical structure of an XML document, which is modelled as an ordered tree. A path expression locates nodes within a tree. For example, the expression

```
book//section/title
```

will select all "title" elements being children of "section" elements which have an ancestor element named "book". The double slash in subexpression "book//section" specifies that there must be a path leading from a "book" element to a "section" element. This corresponds to an ancestor-descendant relationship, i.e. only "section"

elements being descendants of “book” elements will be selected. The single slash in “section/title” denotes a parent-child relationship. It will select only those titles whose parent is a “section” element.

XPath defines additional node relationships to be specified in a path step by an axis specifier. Among the supported axes are ancestor, descendant, parent, child, preceding-sibling or following-sibling. The / and // are abbreviations for the child and descendant-or-self axes. For example, the expression “//section” is short for “/descendant-or-self::node()/child::section”.

According to XPath version 2.0 [2], which is contained as a subset in XQuery, the result of a path expression is a sequence of distinct nodes in document order. The selected node sequence may be further filtered by predicate expressions. A predicate expression is enclosed in square brackets. The predicate is evaluated for each node in the node sequence, returning a truth value. Those nodes in the sequence for which the predicate returns false are discarded. For example, to find all sections whose title contains the string “XQuery” in its text node children, one may use the expression:

```
book//section[contains(title, 'XQuery')]
```

The predicate subexpression specifies a value-based selection, while the subexpression “book//section” denotes a structural selection. Value-based selections can be specified on element names, attribute names/values or the text strings contained in an element. Structural selections are based on the structural relationsships between nodes, such as ancestor-descendant or parent-child.

Quite a number of different XML query language implementations are currently available to XML developers. However, most implementations available as Open Source software rely on conventional top-down or bottom-up tree traversals to evaluate path expressions.

Despite the clean design supported by these tree-traversal based approaches, they become very inefficient for large document collections as has been shown previously [10, 15, 16]. For example, consider an XPath expression selecting the titles of all figures in a collection of books:

```
/book//figure/title
```

In a conventional, top-down tree-traversal approach, the query processor has to follow every path beginning at “book” elements to check for potential “figure” descendants, because there is no way to determine the possible location of “figure” descendants in advance. This implies that a great number of nodes not being “figure” elements have to be accessed to test (i) if the node is an element and (ii) if its qualified name matches “figure”.

Thus index structures are needed to efficiently perform queries on large, unconstrained document collections. The indexing scheme should provide means to process value-based as well as structural selections. While value-based selections are generally well supported by extending traditional indexing schemes, such as B+-trees, structural selections are much harder to deal with. To speed up the processing of path expressions based on structural relationships, an indexing scheme should support the quick identification of such relationships between nodes, for example, ancestor-descendant or parent-child relationships. The necessity to traverse a document subtree should be limited to special cases, where the information contained in indexes is not sufficient to process the expression.

2.2 Numbering Schemes

A considerable amount of research has been carried out recently to design index structures which meet these requirements. Several numbering schemes for XML documents have been proposed [5, 8, 10, 14, 15, 16]. A numbering scheme assigns a unique identifier to each node in the logical document tree, e.g. by traversing the document tree in level-order or pre-order. The generated identifiers are then used in indexes as a reference to the actual node. A numbering scheme should provide mechanisms to quickly determine the structural relationship between a pair of nodes and to identify all occurrences of such a relationship in a single document or a collection of documents.

In this section we will briefly introduce three alternative numbering schemes, which have been recently proposed. We will then discuss the indexing scheme used at the core of eXist, which represents an extension to the level-order numbering scheme presented below.

An indexing scheme which uses document id, node position and nesting depth to identify nodes has been proposed in [16] (also discussed in [15]). According to this proposal, an element is identified by the 3-tuple (document id, start position:end position, nesting level). Start and end position might be defined by counting word numbers from the beginning of the document. Using the 3-tuples, ancestor-descendant relationships can be determined between a pair of nodes by the proposition: A node x with 3-tuple (D1, S1:E1, L1) is a descendant of a node y with 3-tuple (D2, S2: E2, L2) if and only if D1 = D2; S1 < S2 and E2 < E1.

The XISS system ([10], also discussed in [5]) proposes an extended preorder numbering scheme. This scheme assigns a pair of numbers <order, size> to each node, such that: (i) for a tree node y and its parent x, order(x) < order(y) and order(y) + size(y) ≤ order(x) + size(x) and (ii) for two sibling nodes x and y, if x is the predecessor of y in preorder traversal, order(x) + size(x) < order(y). While order is assigned according to a pre-order traversal of the node tree, size can be an arbitrary integer larger than the total number of descendants of the current node. The ancestor-descendant relationship between two nodes can be determined by the proposition that for two given nodes x and y, x is an ancestor of y if and only if order(x) < order(y) ≤ order(x) + size(x).

The major benefit of XISS is that ancestor-descendant relationships can be determined in constant time using the proposition given above. Additionally, the proposed scheme supports document updates via node insertions or removals by introducing sparse identifiers between existing nodes. No reordering of the document tree is needed unless the range of sparse identifiers is exhausted. This feature is used in [5] to assign durable identifiers keeping track of document changes.

Lee et al. [8] proposed a numbering scheme which models the document tree as a complete k-ary tree, where k is equal to the maximum number of child nodes of an element in the document. A unique node identifier is assigned to each node by traversing the tree in level-order. Figure 1 shows the identifiers assigned to the nodes of a very simple XML document, which is modelled as a complete 2-ary tree. Because the tree is assumed to be complete, spare ids have to be inserted at several positions.

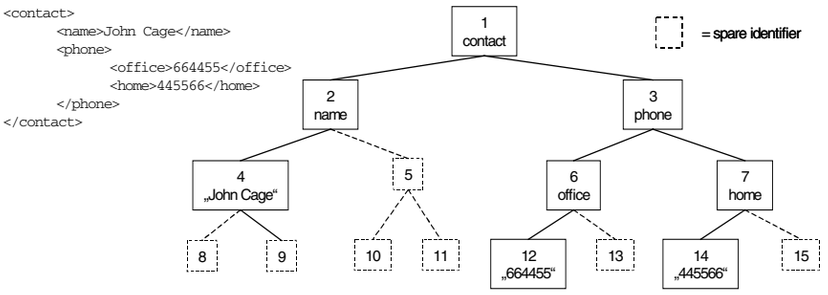


Fig. 1. Unique identifiers assigned by the level-order numbering scheme

The unique identifiers generated by this numbering scheme have some important properties: from a given identifier one may easily determine the id of it's parent, sibling or possible child nodes. For example, for a k-ary document tree we may obtain the identifier of the parent node of a given node whose identifier is i by the following function:

$$parent_i = \left\lfloor \frac{(i-2)}{k} + 1 \right\rfloor \tag{1}$$

However, the completeness constraint imposes a major restriction on the maximum document size to be indexed by this numbering scheme. For example, a typical article will have a limited number of top-level elements like chapters and sections while the majority of nodes consists of paragraphs and text nodes located below the top-level elements. In a worst case scenario, where a single node at some deeply structured level of the document node hierarchy has a the largest number of child nodes, a large number of spare identifiers has to be inserted at all tree levels to satisfy the completeness constraint, so the assigned identifiers grow very fast even for small documents.

The numbering scheme implemented in eXist thus provides an extension to this scheme. To overcome the document size limitations we decided to partially drop the completeness constraint in favour of an alternating scheme. The document is no longer viewed as a complete k-ary tree. Instead the number of children a node may have is recomputed for every level of the tree, such that: for two nodes *x* and *y* of a tree, *size(x) = size(y)* if *level(x) = level(y)*, where *size(n)* is the number of children of a node *n* and *level(m)* is the length of the path from the root node of the tree to *m*. The additional information on the number of children a node may have at each level of the tree is stored with the document in a simple array. Figure 2 shows the unique identifiers generated by eXist for the same document as above.

Our approach accounts for the fact that typical documents will have a larger number of nodes at some lower level of the document tree while there are fewer elements at the top levels of the hierarchy. The document size limit is raised considerably to enable indexing of much larger documents. Compared to the original numbering scheme, less spare identifiers have to be inserted.

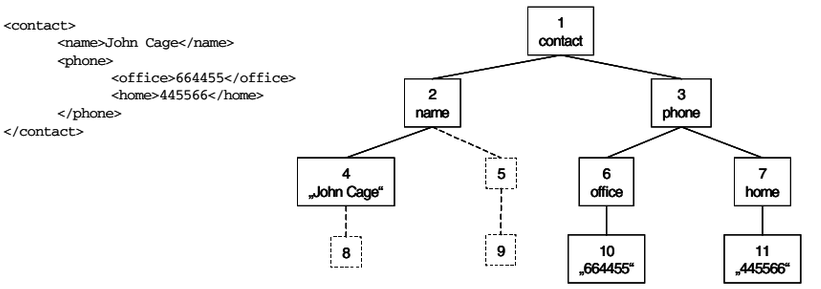


Fig. 2. Unique node identifiers assigned by the alternating level-order numbering scheme

Also inserting a node at a deeper level of the node tree has no effect on the unique identifiers assigned to nodes at higher levels. It is also possible to leave sparse identifiers between existing nodes to avoid a frequent reordering of node identifiers on later document updates. This technique has been described in [5] and [10]. However, eXist does currently not provide an advanced update mechanism as defined, for example, by the XUpdate standard [17]. Documents may be updated as a whole, but it is not possible to manipulate single nodes with current versions of eXist. Support for dynamic document updates is planned for future versions, but currently eXist is best suited for more or less static documents which are rarely updated. We have already started to simplify the generated index structures (see below) as a prerequisite for a future XUpdate implementation.

Using an alternating numbering scheme does not affect the general properties of the assigned level-order identifiers. From a given unique identifier we are still able to compute parent, sibling and child node identifiers using the additional information on the number of children each node may have at every level of the tree.

There are some arguments in favour of the numbering scheme currently implemented. Contrary to our approach, the alternative indexing schemes discussed above concentrate on a limited subset of path expression queries and put their focus on efficient support for the child, attribute and descendant axes of navigation. Since eXist has been designed to provide a complete XPath query language implementation, support for all XPath axes has been of major importance during development. For example, consider an expression which selects the parent elements of all paragraph elements containing the string “XML”:

```
//para[contains(., 'XML')]/..
```

The “..” is short for “parent::node()”. It will select the parent element of each node in the current context node set. Using our numbering scheme, we may easily compute the parent node identifier for every given node to evaluate the above expression. We are also able to compute the identifiers of sibling or child nodes. Thus all axes of navigation can be implemented on top of the numbering scheme.

This significantly reduces the storage size of a single node in the XML store: saving soft or hard links to parent, sibling, child and attribute nodes with the stored node object is not required. To access the parent of a node, we simply calculate its unique

identifier and look it up in the index. Since storing links between nodes is not required, an element node will occupy no more than 4 to 8 bytes in eXist's XML store.

Additionally, with our indexing scheme any node in an XML document may serve as a starting point for an XPath expression. For example, the nodes selected by a first XPath expression can be further processed by a second expression. This is an important feature with respect to XQuery, which allows multiple path expression queries to be embedded into an XQuery expression.

2.3 Index and Data Organization

In this section we provide some implementation details concerning index and data organization. We will then explain how the numbering scheme and the created index structures are used in query processing.

Currently, eXist uses four index files at the core of the native XML storage backend:

- `collections.dbx` manages the collection hierarchy
- `dom.dbx` collects nodes in a paged file and associates unique node identifiers to the actual nodes
- `elements.dbx` indexes elements and attributes
- `words.dbx` keeps track of word occurrences and is used by the fulltext search extensions.

All indexes are based on B+-trees. An important point to note is that the indexes for elements, attributes and keywords are organized by collection and not by document. For example, all occurrences of a "section"-element in a collection will be stored as a single index entry in the element's index. This helps to keep the number of inner B+-tree pages small and yields a better performance for queries on entire collections. We have learned from previous versions that creating an index entry for every single document in a collection leads to decreasing performance for collections containing a larger number (>1000) of rather small (<50KB) documents.

Users will usually query entire collections or even several collections at once. In this case, just a single index lookup is required to retrieve relevant index entries for the entire collection. This results in a considerable performance gain for queries spanning multiple collections. We provide some details on each index file in the following paragraphs:

The index file `collections.dbx` manages the collection hierarchy and maps collection names to collection objects. Due to performance considerations, document descriptions are always stored with the collection object they belong to. A unique id is assigned to each collection and document during indexing.

The XML data store (`dom.dbx`) represents the central component of eXist's native storage architecture. It consists of a single paged file in which all document nodes are stored according to the W3C's document object model (DOM) [9]. The data store is backed by a multi-root B+-Tree in the same file to associate the unique node identifiers of top-level elements in a given document to the node's storage address in the data section (see figure 3).

Only top-level elements are indexed by the B+-tree. Attributes, text nodes and elements at lower levels of the document's node hierarchy are just written to the data

pages without adding a key in the B+-tree. Access to these types of nodes is provided by traversing the nearest available ancestor found in the tree. However, the cases where direct access to these nodes is required are very rare. The query engine will process most types of XPath expressions without accessing dom.dbx.

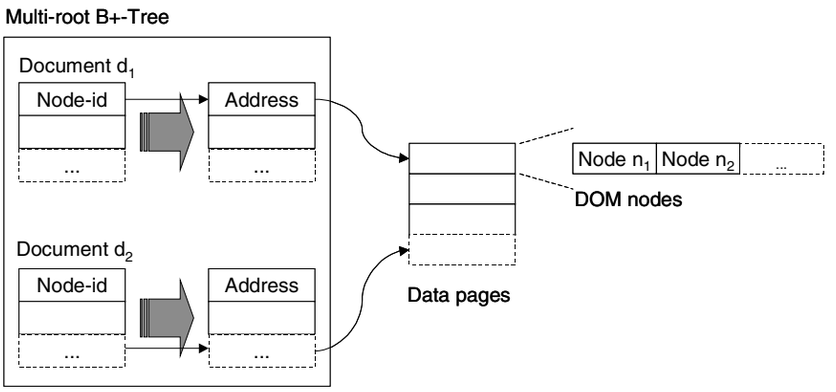


Fig. 3. XML Data Store Organization

Please note again that it is not necessary to keep track of links between nodes, e.g. by using pointers to the next sibling, first child or parent. The DOM implementation completely relies on the numbering scheme to determine node relationships. For example, to get the parent of a node, the parent's unique identifier is calculated from the node's identifier and the corresponding node is retrieved via an index lookup. As a result, the storage size of a document in dom.dbx will likely be smaller than the original data source size for larger documents.

Since nodes are stored in document order, only a single initial index lookup is required to serialize a document or fragment. eXist's serializer will generate a stream of SAX [12] events by sequentially walking nodes in document order, beginning at the fragment's root node.

Element and attribute names are mapped to unique node identifiers in file `elements.dbx`. Each entry in the index consists of a key – being a pair of <collection-id, name-id> - and an array value containing an ordered list of document ids and node ids, which correspond to elements and attributes matching the qualified name in the key. To find, for example, all chapters in a collection of books, the query engine requires a single index lookup to retrieve the complete set of node identifiers pointing to chapter elements.

Since the sequence of document and node ids consists entirely of integer values, it is stored in a combination of delta and variable-byte coding to save storage space.

Finally, the file `words.dbx` corresponds to an inverted index as found in many traditional information retrieval systems. The inverted index represents a common data structure and is typically used to associate a word or phrase with the set of documents in which it has been found and the exact position where it occurred [13]. eXist's inverted index differs from traditional IR systems in that instead of storing the

word position, we use unique node identifiers to keep track of word occurrences. By default, eXist indexes all text nodes and attribute values by tokenizing text into keywords. In words.dbx, the extracted keywords are mapped to an ordered list of document and unique node identifiers. The file follows the same structure as elements.dbx, using <collection-id, keyword> pairs for keys. Each entry in the value list points to a text or attribute node where the keyword occurred. It is possible to exclude distinct parts of a given document type from fulltext-indexing or switch it off completely.

3 Query Language Implementation

Given the index structures presented above, we are able to access distinct nodes by their unique node identifier, retrieve a list of node identifiers matching a given qualified node name or a specified keyword. In this section, we will explain how the available index structures are used by the query engine to efficiently process path expression queries.

eXist currently contains an experimental XPath query language processor. XPath represents a core standard for XML query processing, since it is embedded into a number of other XML query language specifications like XSLT and XQuery. eXist's XPath processor implements major parts of the XPath 1.0 standard requirements, though – at the time of writing – it is not yet complete. However, the existing functionality covers most of the commonly needed XPath expressions. Additionally, several extensions to standard XPath are available, which will be described below.

3.1 Path Join Algorithm

Based on the features provided by the indexing scheme, eXist's query engine is able to use path join algorithms to efficiently process path expressions. Several path join algorithms have been proposed in recent research: Zhang et al. [16] explored the efficiency of traditional merge join algorithms as used in relational database systems for XML query processing. They proposed a new algorithm, multi-predicate merge join, which could outperform standard RDBMS joins.

Two families of structural join algorithms have also been proposed in [15]: Tree-merge and stack-tree. While the tree-merge algorithm extends traditional merge joins and the new multi-predicate merge join, the stack-tree algorithm has been especially optimized for path joins as used in XML query processing.

General path join algorithms based on the extended pre-order numbering scheme of XISS have been proposed and experimentally tested in [10]. Three algorithms are assigned to distinct types of subexpressions: Element-Attribute Join, Element-Element Join and Kleene-Closure Algorithm.

eXist's query processor will first decompose a given path expression into a chain of basic steps. Consider an XPath expression like

```
/PLAY//SPEECH[SPEAKER='HAMLET']
```

We use the publicly available collection of Shakespeare plays for examples [3]. Each play is divided into ACT, SCENE and SPEECH sections. A SPEECH element

includes SPEAKER and LINE elements. The above expression is logically split into subexpressions as show in figure 4.

Fig. 4. Decomposition of Path Expression

The exact position of PLAY, SPEECH and SPEAKER elements is provided in the index file elements.dbx. To process the first subexpression, the query engine will load the root elements (PLAY) for all documents in the input document set. Second, the set of SPEECH elements is retrieved for the input documents via an index lookup from file elements.dbx. Now we have two node sets containing potential ancestor and descendant nodes for each of the documents in question. Each node set consists of <document-id, node-id> pairs, ordered by document identifier and unique node identifier. Node sets are implemented using Java arrays.

To find all nodes from the SPEECH node set being descendants of nodes in the PLAY node set, an ancestor-descendant path join algorithm is applied to the two sets. eXist's path join algorithms are quite similar to those presented in [10]. However, there are some differences due to the used numbering scheme.

We concentrate on the ancestor-descendant-join as shown in figure 5. The function expects two ordered node sets as input: the first contains potential ancestor nodes, the second potential descendants. Every node in the two input sets is described by a pair of <document-id, node-id>. The function recursively replaces all node identifiers in the descendant set with the id of their parent using function get_parent_set in the outer loop. The inner loop then compares the two sets to find equal pairs of nodes by incrementing either ax or dx depending on the comparison. If a matching pair of nodes is found, ancestor and descendant node are copied to output. The algorithm terminates if get_parent_set returns false, which indicates that the descendant list contains no more valid node identifiers.

The outer loop is repeated until all ancestor nodes of descendants in dl are checked against the ancestor set. This way we ensure that extreme cases of element-element joins are properly processed, where a single node is a descendant of multiple ancestor nodes.

The generated node set will become the context node set for the next subexpression in the chain. Thus the resulting node set for expression PLAY//SPEECH becomes the ancestor node set for expression SPEECH[SPEAKER], while the results generated by evaluating the predicate expression SPEAKER="HAMLET" become the descendant node set.

To evaluate the subexpressions PLAY//SPEECH and SPEECH[SPEAKER], eXist does not need access to the actual DOM nodes in the XML store. Both expressions are entirely processed on basis of the unique node identifiers provided in the index file. Additionally, the algorithm determines ancestor-descendant relationships for all candidate nodes in all documents in one single step.

```
Algorithm ancestor_descendant_join(al, dl)
  dl_orig = copy of dl;
  // get_parent_set replaces each node-id
  // in dl with the node-id of its parent.
  while(get_parent_set(dl)) {
    ax = 0;
    dx = 0;
    while(dx < dl.length) {
      if(dl[dx] == null)
        dx++;
      else if(dl[dx] > al[ax]) {
        if(ax < al.length - 1)
          ax++;
        else
          break;
      } else if(dl[dx] < al[ax])
        dx++;
      else {
        output(al[ax], dl_orig[dx]);
        dx++;
      }
    }
  }
```

Fig. 5. Ancestor-Descendant Join

Yet to process the equality operator in the predicate subexpression, the query engine will have to retrieve the actual DOM nodes to determine their value and compare it to the literal string argument. Since a node's value may be distributed over many descendant nodes, the engine has to do a conventional tree traversal, beginning at the subexpression's context node (SPEAKER).

This could be avoided by adding another index structure for node values. However, for many documents addressing human users, exact match query expressions could be replaced by corresponding expressions using the fulltext operators and functions described in the next section. We have thus decided to drop the value index supported by previous versions of eXist to reduce disk space usage.

3.2 Query Language Extensions

The XPath specification only defines a few limited functions to search for a given string inside the character content of a node. This is a weak point if one wants to search through documents containing larger sections of text. For many types of documents, the provided standard functions will not yield satisfying results.

eXist offers two additional operators and several extension functions to provide access to the fulltext content of nodes. For example, to select the scene in the cavern from Shakespeare's Macbeth:

```
//SCENE[SPEECH[SPEAKER &= 'witch' and near(LINE, 'fenny
snake']]
```

&= is a special text search operator. It will select context nodes containing all of the space-separated terms in the right hand argument. To find nodes containing any of the terms the |= operator is used. The order of terms is not important. Both operators support wildcards in the search terms. To impose an order on search terms the near(node set, string, [distance]) function selects nodes for which each of the terms from the second argument occur near to each other in the node's value and in correct order. To match more complex string patterns, regular expression syntax is supported through additional functions.

All fulltext-search extensions use the inverted index file `words.dbx`, which maps extracted keywords to an ordered list of document and unique node identifiers. Thus, while the equality operator as well as standard XPath functions like contains require eXist to perform a complete scan over the contents of every node in the context node set, the fulltext search extensions rely entirely on information stored in the index.

4 Performance and Scalability

To estimate the efficiency of eXist's indexing and query processing some experimental results are provided in this section. We compare overall query execution times for eXist, Apache's Xindice [1] and an external XPath query engine [11] which is based on a conventional tree-traversal based approach. In a second experiment we process the same set of queries with increasing data volumes to test the scalability of eXist.

We have chosen a user-contributed data set with 39.15 MB of XML markup data containing 5000 documents taken from a movie database. Each document describes one movie, including title, genre, ratings, complete casts and credits, a summary of the plot and comments contributed by reviewers. Document size varies from 500 bytes to 50 KB depending on the number of credits and comments. Experiments were run on a PC with AMD Athlon 4 processor with 1400 MHZ and 256 MB memory running Mandrake Linux 8.2 and Sun's Java Development Kit 1.4.

We formulated queries for randomly selected documents which might typically be of interest to potential users. For example, we asked for the titles of all western movies or films with certain actors or characters.

The Jaxen XPath engine [11] has been selected to represent a conventional, top-down tree-traversal based query engine. For our experiment, Jaxen runs on top of eXist's persistent DOM implementation. Additionally, we processed the same set of queries with an alternative native XML database, Apache's Xindice. Since Xindice requires manual index creation, we defined an index on every element referenced by our queries. Our test client used the XML:DB API to access Xindice as well as eXist.

Each query in the set has been repeated 10 times for each test run to allow B+-Tree page buffers to come into effect. This corresponds to normal database operation where the database server would run for a longer period of time with many users doing similar queries with respect to input document sets and element or attribute selections. Xindice and eXist use the same B+-Tree code base. Running on top of eXist's persistent DOM, Jaxen equally benefits from page buffering mechanisms.

As described above, eXist does not create an index on element and attribute values. For a second test run, we thus replaced all exact match expressions by equivalent fulltext search expressions. For example, the expression //movie[.//credit='Gable, Clark'] has been reformulated as follows: //movie[near(.//credit, 'Gable, Clark')]. Both sets of queries are equivalent with respect to the generated number of hits for our data set.

Average query execution times for selected queries are shown in table 1. Execution times for retrieving result sets have not been included. They are the same for the eXist-based approaches. Retrieving results merely depends on the performance of eXist's serializer, which has no connection to the query engine.

Table 1. Avg. query execution times for selected queries (in seconds)

XPath Query	eXist	eXist + extensions	Xindice	Jaxen
/movie[.//genre='Drama']//credit[@role='directors']	3.44	1.14	10.62	21.86
/movie[genres/genre='Western']/title	0.79	0.23	1.39	7.58
/movie[languages/language='English']/title	1.45	0.97	34.18	8.50
/movie[.//credit/@charactername='Receptionist']	3.12	0.21	27.04	51.48
/movie[contains(.//comment, 'predictable')]	2.79	0.20	25.75	31.49
/movie[.//credit='Gable, Clark']	4.47	0.35	0.38	33.72
/movie[.//languages/language='English']/title[starts-with(.,'42nd Street')]	1.63	0.32	17.47	32.64
/movie[languages/language='English' and credits/credit='Sinatra, Frank']	5.16	0.58	0.11	13.26

Our results show that eXist's query engine outperforms the tree-traversal based approach implemented by Jaxen by an order of magnitude. This supports previous research results indicating the superiority of path join algorithms [10, 15, 16]. It is also no surprise that search expressions using the fulltext index perform much better with eXist than corresponding queries based on standard XPath functions and operators. Results for Xindice show that selections on the descendant axis (using the // symbol) are not very well supported by their XPath implementation. Contrary to Xindice, eXist handles these types of expressions efficiently.

In a second experiment, the complete set of 5000 documents was split into 10 subcollections. To test scalability we added one more subcollection to the database for each test sequence and computed performance metrics for eXist with the standard XPath and extended XPath query sets. Thus the raw XML data size processed by each test cycle increased from 5 MB for the first collection up to 39.15 MB for 10 collections. As before, each query has been repeated 10 times. Average query execution times for the complete set of queries are shown in figure 6.

We observe for both sets of queries that query execution times increase at least linearily with increasing source data size. Thus our experiment shows linear scalability of eXist's indexing, storage and querying architecture.

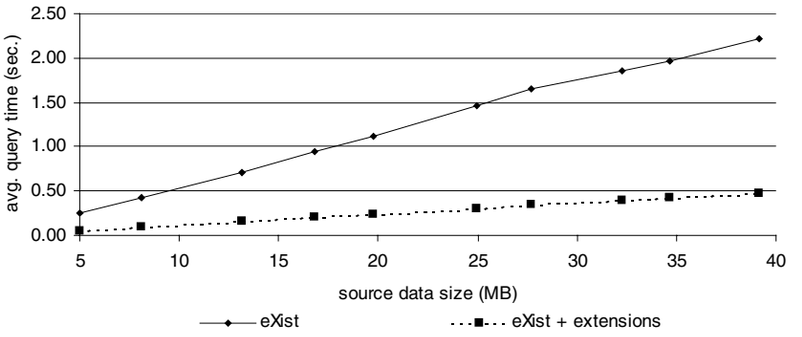


Fig. 6. Avg. query execution times by source data size

5 Outlook

Despite the many projects already using eXist, there is still much work to be done to implement outstanding features and increase usability and interoperability. Some of eXist's weak points – namely indexing speed and storage requirements - have already been subject to a considerable redesign. We are currently concentrating on complete XPath support, possibly using existing implementations developed by other projects.

Another important topic is XUpdate – a standard proposed by the XML:DB initiative for updates of distinct parts of a document [17]. eXist does currently not provide an advanced update mechanism. Documents may only be updated as a whole. While this is a minor problem for applications dealing with relatively static document collections, it represents a major limitation for applications which need to frequently update portions of rather large documents.

As explained above, the numbering scheme could be extended to avoid a frequent reordering of node identifiers on document updates by introducing sparse identifiers between nodes [5, 10]. The necessary changes to handle sparse identifiers have already been implemented. We have also started to simplify the created index structures, making them easier to maintain on node insertions or removals. However, some work remains to be done on these issues.

Additionally, support for multiversion documents using durable node numbers has been proposed in [5]. The scheme described there could also be implemented for eXist.

Being an open source project, eXist strongly depends on user feedback and participation. Interested developers are encouraged to join the mailing list and share their views and suggestions.

References

[1] The Apache Group. "Xindice Native XML Database". http://xml.apache.org/xindice.

[2] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. "XML Path Language (XPath) 2.0. W3C Working Draft 30 April 2002. http://www.w3.org/TR/xpath20. Working Draft, 2002.

[3] John Bosak. XML markup of Shakespeare's plays, January 1998. http://ibiblio.org/pub/sun-info/standards/xml/eg/.

[4] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon and M. Stefanescu. "XQuery 1.0: An XML Query Language". http://www.w3.org/TR/xquery. W3C Working Draft, W3C Consortium, December 2001.

[5] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, and Donghui Zhang. "Efficient Complex Query Support for Multiversion XML Documents". In *Proceedings of the EDBT Conference*, 2002.

[6] James Clark, and Steve DeRose. "XML Path Language (XPath) Version 1.0". W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath. W3C Recommendation, W3C Consortium, 1999.

[7] Darmstadt University of Technology, IT Transfer Office. "PRIMA – Privacy Management Architecture". http://www.ito.tu-darmstadt.de/PRIMA/.

[8] Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon, and P. Bruce Berra. "Index Structures for Structured Documents". In *Proceedings of the 1st ACM International Conference on Digital Libraries,* March 20-23, 1996, Bethseda, Maryland, USA. ACM Press, 1996.

[9] A. Le Hors, P. Le Hegaret, G. Nicol, J. Robie, M. Champion and S. Byrne. "Document Object Model (DOM) Level 2 Core Specification Version 1.0". http://www.w3.org/TR/DOM-Level-2-Core/. W3C Recommendation, Nov. 2000.

[10] Quanzhong Li and Bongki Moon. "Indexing and Querying XML Data for Regular Path Expressions". In *VLDB 2001, Proceedings of 27th International Conference on Very Large Databases*, September 11-14, 2001, Roma, Italy.

[11] Bob McWhirter and James Strachnan. "Jaxen: Universal XPath Engine". http://www.jaxen.org.

[12] David Megginson. "SAX: Simple API for XML". http://sax.sourceforge.net/.

[13] G. Salton and M. J. McGill. "Introduction to Modern Information Retrieval". McGraw-Hill, New York, 1983.

[14] Dongwook Shin, Hyuncheol Jang, and Honglan Jin. "BUS: An Effective Indexing and Retrieval Scheme in Structured Documents". In *Proceedings of the 3rd ACM International Conference on Digital Libraries*, June 23–26, 1998, Pittsburgh, PA, USA. ACM Press, 1998.

[15] Divesh Srivastava, Shurug Al-Khalifa, H.V. Jagadish, Nick Koudas, Jignesh M. Patel, and Yuqing Wu. "Structural Joins: A Primitive for Efficient XML Query Pattern Matching". In *Proceedings of the ICDE Conference*, 2002.

[16] Chun Zhang, Jeffrey F. Naughton, David J. DeWitt, Qiong Luo, and Guy M. Lohmann. "On Supporting Containment Queries in Relational Database Management Systems". In *Proceedings of the SIGMOD Conference*, 2001, Santa Barbara, California, USA.

[17] The XML:DB Project. "XUpdate Working Draft". http://www.xmldb.org/xupdate/. Technical report, 2000.

[18] The XML:DB Project. "XML:DB Database API Working Draft". http://www.xmldb.org/xapi/. Technical report, 2001.

WrapIt: Automated Integration of Web Databases with Extensional Overlaps

Mattis Neiling*, Markus Schaal*, and Martin Schumann

Free University of Berlin, Department of Economics,
Institute for Information Systems, Garystraße 21, D-14195 Berlin, Germany
mneiling@wiwiss.fu-berlin.de, schaal@cs.tu-berlin.de,
info@schumannsoftware.de

Abstract. The world wide web does not longer consist of static web pages. Instead, more and more web pages are created dynamically from user request and database content. Conventional search engines do not consider these dynamic pages, as user input cannot be simulated, thus providing often insufficient results.
A new approach for online integration of web databases will be presented in this paper. Providing only one sample HTML result page for a source, result pages for new requests will be found by structural recognition. Once structural recognition is established for one source, other web databases of the same universe (e.g. movie databases) can be integrated on the fly by content-based recognition. Thus, the user receives results from various sources.
Global schemata will not be produced at all. Instead, the heterogeneity of the single sources will be preserved. The only requirement is given by the existence of an extensional overlap of the databases.

1 Introduction

The world wide web is increasing at an enormous rate. Already now it can be viewed as the largest source of information. According to Najork and Viennese [MN01], the following estimation applied in October 2000: 2,5 billion pages were directly accessible and 550 billion pages were dynamically generated. Especially, more and more databases are made available via the web. In order to access information from these web databases, queries are entered into certain input forms (text boxes, check boxes, etc.). The query result is returned by dynamically generated *result pages*. Users need to access each source individually in order to formulate appropriate queries and to identify the results.

Automated searching across several web databases requires knowledge about the structure of each individual site. Nowadays, individual wrappers are developed for each source separately. This solution is not very satisfying, as new sources appear every day and legacy sources may change their layout and programming style. Instead of creating one wrapper per source, a spider is used for

* Part of this work was supported by the Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316)

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 184–198, 2003.

querying sources. Following this wrapper-free approach, a method for recognition of result pages is required. Related work (e.g. Crescenci et. al. [CMM01]) has proposed the use of several (at least two) sample *result* pages per source in order to recognize other result pages. Our approach extends this idea to multiple sources, requiring only *one* sample page for *one* source.

Our idea is as follows: The structure of result pages of the first source can be derived by comparing the sample page with new query results (so-called *structural recognition*). Once the structure of the first source is known, domain-specific content can be extracted from this source. Then, the same query is sent to another source and result pages are recognized there by content-based matching (so-called *content-based recognition*).

Thus, wrapper-free search can be extended towards automated integration of previously unknown or changing web databases.

2 State of the Art

Semi-automated wrapper generation has been recognized as challenge in the scientific community. At least two approaches exist, namely Lixto (cf. Baumgartner et. al. [BFG01]) and W4F (World Wide Web Wrapper Factory, cf. Sahuguet et. al. [SA99]).

Lixto generates Wrappers by user interaction. A graphical user interface (Interactive Pattern builder) enables the user to select text parts within HTML pages. Each text part is an example for a kind of data which can be extracted. Lixto represents the HTML page internally as a tree where text parts correspond to partial trees. Additional knowledge has to be provided by the user in terms of so-called Lixto or Elog rules for the extraction of the actual data.

As it is the case for Lixto, W4F also uses a query language (HEL, HTML Extraction Language) for producing extraction rules. From these rules JAVA source code is produced. This source code is the actual wrapper and contains not only the extraction methods, but also methods for communication with the web server.

Besides semi-automated wrapper generation, there is also an approach for automated wrapper generation, namely Roadrunner (cf. Crescenci et. al. [CMM01]). Very similar to the approach described here, Roadrunner starts out with two sample result pages and analyzes the structure of such pages by parsing the HTML pages in parallel. Identical elements are ignored. Different text strings are used in order to localize content, while different HTML tags are used in order to analyze optional or repeating structures. The Roadrunner approach leads to a grammar for result pages.

While our approach builds upon the concepts realized in Roadrunner, we have gone one step further by extending the search to other sources without having previous knowledge except for their Unified Resource Locator (URL).

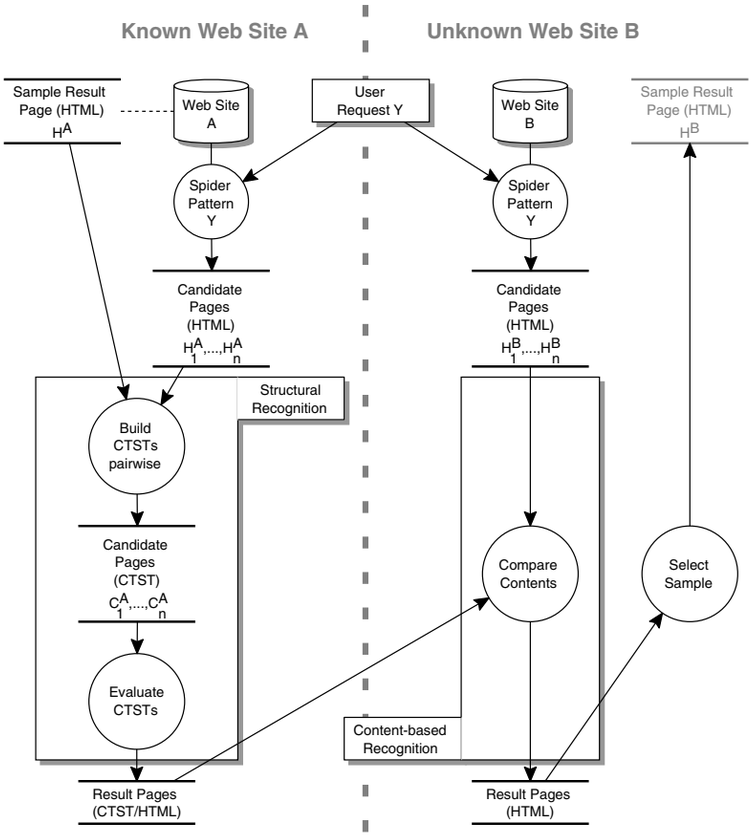


Fig. 1. Abstract Data Flow Diagram of *WrapIt*; the concept of the CTST's, the Content-Tagged Structure Trees, is introduced in section 3 and further discussed in section 4.

3 Our Approach

Content in the world wide web is provided by different web sites in a variety of different kinds. The results and the communication steps for achieving these results vary strongly. Some web sites may answer a query directly with the result, others show a summary first. The automated distinction between the *result pages* (containing the details) and other pages (such as summaries, etc.) is our core issue. The recognition of *result pages* is a central issue for automated searching of previously unknown web databases.

The recognition of result pages requires knowledge about the structure and/or content of such a page. We have used two different methods for two different situations:

Structural Recognition: In cases, where the source is already known and structure is given by sample result pages, we follow the Roadrunner-approach [CMM01] and perform a search for pages of similar structure. For a given user request, the result pages are those, which have a similar structure. However, we do not compute a grammar for result pages, but generate so-called Content-Tagged Structure Trees (CTSTs)

Content-based Recognition: In cases, where the source is not previously known, a user request will also be given to the previously known sources. Similar results (relating to the same real-world object) may then be recognized by comparing the contents of the result pages of the known source with all candidates from the previously unknown source. Once a result page has been recognized by this method, it can be further used as a sample page for structural recognition.

Content-based recognition requires an extensional overlap between a known source and an unknown source. Moreover, the user request must produce results from the extensional overlap. Therefore, we distinguish between administrative and user level use of our prototype *WrapIt*. Content-based recognition is reserved for the administrative level only where experts are required to enter requests that are likely to refer to the extensional overlap. For user level use, knowledge about extensional overlaps is not required and pages are structurally recognized by comparison with sample pages.

Figure 1 depicts the complete approach as an abstract data flow diagram. Web site A (left side) is assumed to be previously known, while web site B is the unknown site. In both cases, a simple spider is used for the generation of HTML candidate pages. The spider tries to enter the user request pattern Y into appropriate form fields and follows links.

For structural recognition, the candidate pages are matched pairwise with the sample result page, leading to so-called Content-Tagged Structure Trees (CTSTs). This matching is performed upon the DOM-tree[1] and can be rather sophisticated as repetitive and optional structures have to be recognized. Details will be discussed in Section 4.

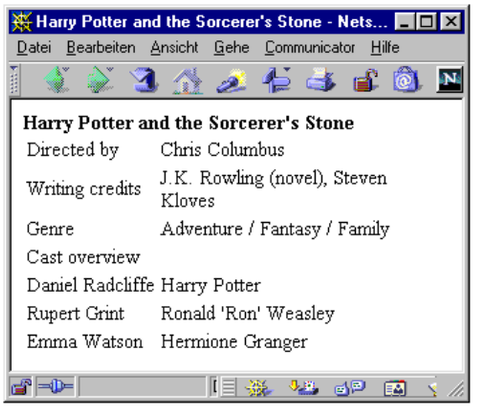
The CTSTs are subsequently evaluated and ranked heuristically with respect to a so-called S-measure, which gives high ranking for candidate pages who have a similar structure but textual distinctions. CTSTs with sufficiently high ranking are considered being result pages and the textual distinctions are stored as term list for content-based recognition. Details will be discussed in Section 5.

For content-based recognition, the HTML candidate pages (of web site B) are directly searched for the previously stored content-tags with usual information retrieval techniques. The closest results are considered being result pages for web site B. Details will be discussed in Section 6.

[1] Document Object Model for HTML according to `http://www.w3c.org/dom`

4 Structural Matching

The structural analysis is used for the recognition of result pages when a sample page is already known. For any pair of HTML sample page H^A and HTML candidate page H_i^A the respective DOM-trees D^A and D_i^A are built first. The DOM-tree represents the hierarchical tree-structure of HTML source code explicitly. Metatags are ignored and the root is always the `body`-Tag. Textual information is always located in the leaves of the tree. The path to the leaves represents the formatting.

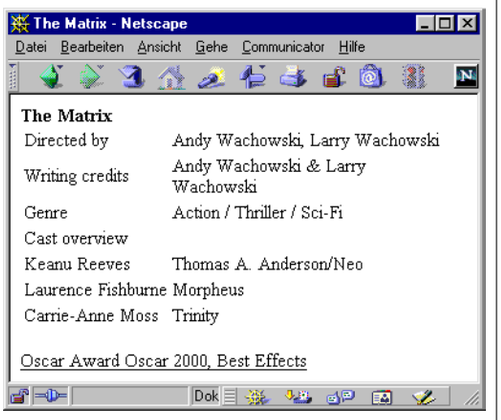



```
<body>
 <b>Harry Potter and the Sorcerer's Stone</b>
 <table border=0 width=400>
  <tr>
    <td>Directed by</td>
    <td>Chris Columbus</td></tr>
  <tr>
    <td>Writing credits</td>
    <td>J.K. Rowling (novel),
        Steven Kloves</td></tr>
  <tr>
    <td>Genre</td>
    <td>Adventure / Fantasy /
        Family</td></tr>
  <tr>
    <td>Cast overview</td></tr>
  <tr>
    <td>Daniel Radcliffe</td>
    <td>Harry Potter</td></tr>
  <tr>
    <td>Rupert Grint</td>
    <td>Ronald 'Ron' Weasley</td></tr>
  <tr>
    <td>Emma Watson</td>
    <td>Hermione Granger</td></tr>
 </table>
</body>
```

Fig. 2. Screen dump of a sample page H^A of the movie *Harry Potter and the Sorcerer's Stone* (left) and the corresponding HTML-code (right)

The Content-Tagged Structure Tree (CTST) for such a DOM-tree D_i^A with respect to a referential DOM-tree D^A will be generated as follows. By sequential pairwise comparison of these paths the leaves of D_i^A are tagged with respect to the following cases:

1. The path is present in both trees and the leave content is identical. Such leaves are marked with a DELETION tag. Most likely, they represent irrelevant information that appears on any instance of a result page at this web site (like ads, headings, non-specific attributes).
2. The path is present in both trees, but the leave content is different. Such leaves are marked with a CONTENT tag. Most likely, they represent actual content being retrieved from the database upon user request.
3. The path is present only in D_i^A. Such paths are marked with an OPTIONAL tag.
4. The path represents a repeated structure. Such paths are marked with an ITERATION tag.




```
<body>
 <b>The Matrix</b>
 <table border=0 width=500>
  <tr>
    <td>Directed by</td>
    <td>Andy Wachowski,
        Larry Wachowski</td></tr>
  <tr>
    <td>Writing credits</td>
    <td>Andy Wachowski
        & Larry Wachowski</td></tr>
  <tr>
    <td>Genre</td>
    <td>Action / Thriller /
        Sci-Fi</td></tr>
  <tr>
    <td>Cast overview</td></tr>
  <tr>
    <td>Keanu Reeves</td>
    <td>Thomas A. Anderson/Neo</td></tr>
  <tr>
    <td>Laurence Fishburne</td>
    <td>Morpheus</td></tr>
  <tr>
    <td>Carrie-Anne Moss</td>
    <td>Trinity</td></tr>
 </table>
 <br>
 <U>Oscar Award Oscar 2000, Best Effects</u>
</body>
```

Fig. 3. Screen dump of a candidate page H_1^A of the movie *Matrix* (left) and the corresponding HTML-code belonging (right)

A small example will illustrate the process of tree matching as described here.

Example 1. Starting with two simple HTML pages H^A and H_1^B about movies as depicted in Figures 2 and 3, the HTML code is transformed into standard DOM-trees D^A and D_1^B.

The structure of these DOM-trees is derived from the structure of the HTML code of as follows:

A node is created for each pair of HTML-tags like `<b>...<\b>`, `<tr>...<\tr>`, or `<td>...<\td>`.

Further nodes lying between these tags are child nodes with respect to the enclosing node, i.e. all that is between these tags is placed at succeeding nodes respectively. Textual information (the content) is always placed at terminal nodes which will be called leave nodes also.

Structure recognition (and CTST creation) is now performed on DOM-trees. We compare the paths to the leaves between H^A and H_i^A as follows:

If a path is present in both trees with identical content, then there is no distinguishing information given by that. Lets consider path

`<body>/<table>/<tr>/<td>`

with leave content

`Directed by`

This node is to be tagged with a DELETION tag. But if only the path matches properly and the content is different as it is the case for leave nodes

`<body>/<b>Harry Potter ...`

`<body>/<b>Matrix`

respectively, an identical structural element with different content has been found. Such a leave node is then to be marked with a CONTENT tag.

If paths to leave nodes occur only in one of both DOM-trees, the node is tagged with an OPTIONAL tag, as in the case of

```
<body>/<U>Oscar Award Oscar 2000, Best Effects
```

in H_i^A. Last but not least, repeated structures are recognized as well. Paths occurring at least twice within the DOM-tree are marked with the REPETITION tag. Note, that such paths may not necessarily lead to leave nodes, as it is the case for the path

```
<body>/<table>/<tr>
```

in both DOM-trees.

Both result pages are displayed as screen dumps and HTML code in the Figures 2 and 3. The resulting CTST-tree is shown in Figure 4 as screen dump.

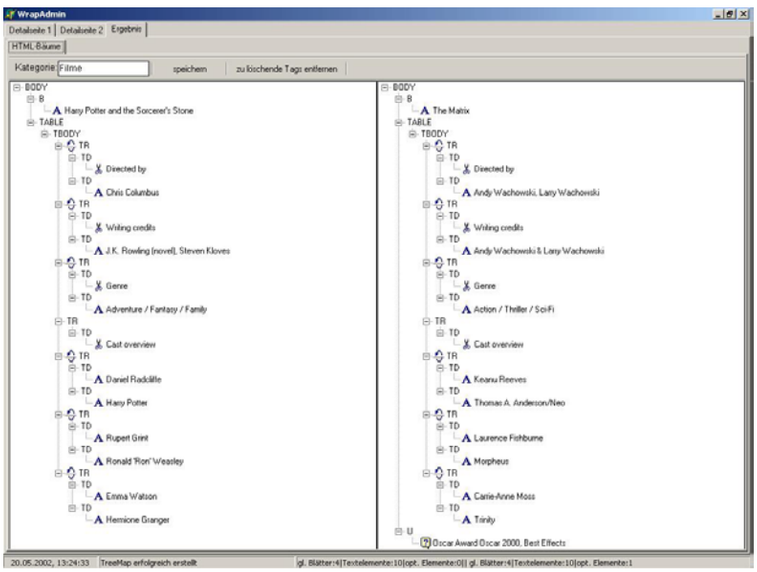


Fig. 4. Screen dump of the resulting CTST's of the HTML-pages of fig. 2 (left hand side) and fig. 3 (right hand side)

Intuitive pictograms have been employed for the visualization of the different tags. The A-symbol represents the CONTENT tag, the *scissors*-symbol represents the deletion tag, etc.

Intuitive pictograms have been employed for the visualization of the different tags of the CTST (compare fig 4):

- the **A**-symbol represents the CONTENT tag,
- the *scissors*-symbol represents the DELETION tag,
- the *balloon with a question mark* represents the OPTIONAL tag, and
- the *rotation*-symbol represents the REPETITION tag.

5 CTST Evaluation

Each CTST C_i^A will be evaluated with the heuristic measure S :C→ $\mathbb{R}$, where C is the space of all CTSTs with

$$S(C_i^A) = \#ContentTag - \#OptionalTag \tag{1}$$

Numerous other measures can be thought of. This measure has been experimentally shown to effectively distinguish between result pages and other pages by choosing 0 as treshold value, i.e.

$S(C_i^A) > 0$**:** Most likely, the candidate page is a result page.
$S(C_i^A) = 0$**:** Most likely, both pages are identical.
$S(C_i^A) < 0$**:** Most likely, the candidate page is no result page.

Actually, we have not found values near to zero (in most cases[2]). Further experiments will show whether this measure together with threshold value 0 is indeed a good classifier for result pages.

6 Content Matching

In order to recognize result pages from unknown web sites, the content has to be analyzed. For a specific query Y the result pages for known web sites are computed first by structural analysis. Similar results from known and unknown sites are likely to refer to the same real world object.

In a pre-processing step, term lists are to be retrieved from both pages whose similarity is to be measured. The measurement of similarity is then performed upon these term lists instead of the original page. The method for term list extraction is described next:

For known sources: Content for the term lists is taken from those leaves in the CTST that are labelled with the CONTENT or OPTIONAL tag respectively. The term list contains all words contained in this leaves and some phrases

For unknown sources: In this case, the actual content is not localized. The term list will be generated from all leave nodes of the DOM-tree.

[2] Regrettably there is also a counter example which has been documented in the result section (cf. Example 3 in Section 8).

After the pre-processing step, a term list L_k^A is retrieved from some result page H_k^A of known web site A (using the CONTENT and OPTIONAL tags in the CTST) and a second term list L_j^B is retrieved from some candidate page H_j^B of unknown web site B (using the DOM-tree). The frequency of each term within the respective page is stored in addition to the term itself.

For measuring similarity, we adopt the well-known *vector space model* for *documents* [Sal89]. In the *vector space model* documents consist of terms, e.g. words or phrases. The similaririty of two term lists can be computed with the TF–IDF weighting scheme. TF–IDF abbreviates Term Frequency– Inverse Document Frequency, a widely used method for comparing text documents. W. Cohen applies this approach successfully to the mediation of web data sources cf. [Coh98].

In the vector space model, a *document vector* v consists of nonnegative real numbers, the i-th component of v corresponds to the i-th term t_i of a term enumeration. In our approach, the *documents* d are given by the result pages H^A, H_k^A for the web database A and the result pages H^B, H_k^B for the web database B, respectively. The value of $v(i)$ corresponding to a term t_i is defined for a result page d by

$$v(i) = v_d(i) = \begin{cases} 0 & : \quad t_i \notin d \\ (1 + \log TF(d,i)) \cdot \log IDF(i) & : \quad t_i \in d \end{cases} \tag{2}$$

where the $TF(d,i)$ denotes *term frequency* of the i-th term t_i in the result page d. $IDF(i)$ is the relative *inverse document frequency* of the i-th term t_i within a given collection DOC of result pages. $IDF(i)$ is defined as follows

$$IDF(i) = \frac{|DOC|}{|\{d \in DOC \mid t_i \in d\}|}\,. \tag{3}$$

The similarity of two documents is defined by the inner product of the vector space:

$$Sim(d_1, d_2) = v_{d_1}^T v_{d_2} = \sum_i v_{d_1}(i) \cdot v_{d_2}(i) \tag{4}$$

The similarity of two documents gets the higher the more terms coincide. Rare terms t being in both documents d_1, d_2 lead to high values for the corresponding component of $v(i)$. By this, rare terms are heavy indicators for similarity, while widely used terms receive only little weight.

For efficient computation of $IDF(t)$, we use a domain-specific dictionary DIC instead of DOC. DIC is generated from the known web sites by posing queries with requests from a predefined list. The dictionary DIC is then build from the contents of all result pages that have been found. An entry in DIC consists of a term t_i and its cumulated frequency $DIC(i)$ in the assessed result pages DOC. If the number of the assessed result pages $N = |DOC|$ is stored separately, the formula (3) becomes

$$IDF(i) = \frac{N}{DIC(i)}. \tag{5}$$

For the calculation of the similarity between two HTML-pages H_k^A, H_j^B only terms t_i occurring both in L_k^A and L_j^B are considered, since either $v_A(t)$ (referring to the term list L_k^A) or $v_B(t)$ (referring to the term list L_i^B) equals zero in the other cases (cf. Equation 2). Formula (4) can thus be rewritten for efficient computation of the similarity between HTML-pages H_k^A and H_j^B as follows (with (2) and (5)):

$$\begin{aligned} & Sim(H_k^A, H_j^B) = Sim(L_k^A, L_j^B) \\ & = \sum_{i:t_i \in L_k^A \cap L_j^B} v_{L_k^A}(i) \cdot v_{L_j^B}(i) \\ & = \sum_{i:t_i \in L_k^A \cap L_j^B} \underbrace{\left(1 + \log TF(L_k^A, i)\right)}_{\text{value for } t_i \text{ in } L_k^A} \cdot \underbrace{\left(1 + \log TF(L_j^B, i)\right)}_{\text{value for } t_i \text{ in } L_j^B} \cdot \underbrace{\left(\log \frac{N}{DIC(i)}\right)^2}_{IDF\text{–weight for } t_i} \end{aligned} \quad (6)$$

where $TF(L_j^B, i)$ denotes the frequency of the i-th term t_i in term list L_j^B. In this way we compute the similarity directly from the term lists, without explicit use of the vector representation as described above. This procedure has a complexity of $O(n)$, where n denotes the (maximal) size of the term lists (number of terms). However, typically n becomes not large for result pages (approximately 100 terms).

By employing the TF–IDF model, candidate pages from unknown sources can be ranked by their similarity to the result pages of the known web site. Lets assume, that candidate page H_j^B (unknown source) has the highest similarity value when compared to some H_k^A (known source). All other pairs of candidate and result pages have lower values. Then, both pages are assumed to refer to the same real-world object. Result page H_j^B (unknown source) is stored as new sample page H^B such that web database B is a known source from now on.

7 Remarks on the Spider

User requests are usually sent to the web server via forms. The request is formulated by text entered in one or several text boxes and by choices from predefined lists. Gratifyingly, most web databases offer quick search requiring input for only one text box. However, we have only studied web databases were the user query can be entered in the first text field of the form.

Except for following links, this is the only task for the spider to be performed. A more sophisticated approach to the analysis of forms is given by Raghavan and Garcia-Molina [RGM01].

8 Experimental Results

Our prototype *WrapIt* is implemented as standalone application with Borland Delphi 5 and uses classes shipped with the Microsoft Internet Explorer 6.0,

Fig. 5. Thumbnails of the result pages for the request *matrix* from 6 databases, as shown in table 1. It can be seen, that layout and structure differ.

namely the library `mshtml.dll`. We use these classes to display HTML-files and for the spider querying the web database and crawling through the results.

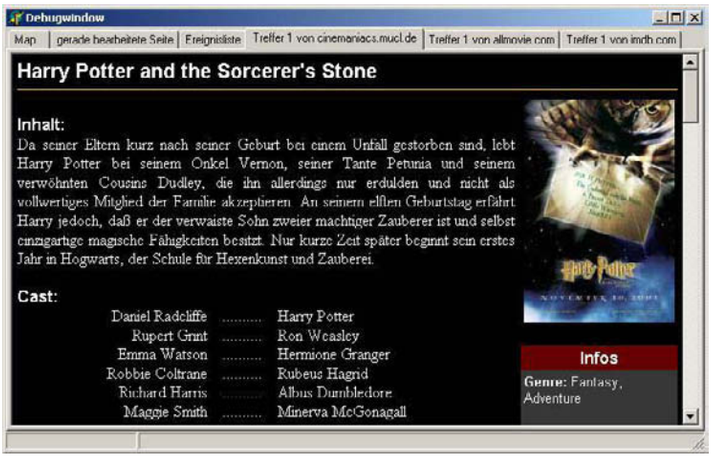


Fig. 6. Screen dump of the prototype *WrapIt*, displaying a result page for the request *potter* (Example 3)

All tests presented here have been carried out on a PC equipped with the processor AMD Athlon 1.7XP+ (1.433 GHz) with 512MB RAM on Windows XP Professional. Since massive web querying is performed, the connectivity to the Internet might be of interest, too. The PC was connected to the Internet by

use of ADSL with 768 Kbit/s downstream and 128 Kbit/s upstream bandwidth through a Ethernet-card.

We applied tests within two universes of discourse (or domains), namely movie databases and online book sellers. Since the prototype is implemented based upon Internet Explorer 6.0, the user can follow the querying, page loading and browsing by the program by watching the actual HTML page, thus he can trace the progress of the program. Alternatively, when performing an administrator run, a log window can be viewed, where the querying, page loading and matching results of the program are documented.

First of all we present the results for the administrative usage level, where previously unknown web databases are to be analyzed by the content-based recognition described in section 6.

Example 2 (Content-based recognition). For the administrative usage level the administrator feeds one sample HTML-page from a web database A to the program (by browsing to the page and saving it with the embedded IE 6.0), in our movie example it consists of the result page of the movie *Lord of the Rings* from `http://www.imdb.com`. Additionally, a list of URL's of movie databases is stored in an `.ini`-file, in our case for 9 databases (including the known source `http://www.imdb.com`). The administrative user request in our example is *Matrix*, since new and Oscar-awarded movies are likely to appear in most of the movie databases.

The results are presented in table 1. We found the movie in all but one database[3] (`http://rasp.nexenservices.com`). The overall runtime was about eight minutes, since the queries were send sequentially.

Example 3 (Structural recognition — movies). Once the sample HTML pages are found and stored as demonstrated in example 2, user requests can be handled by structural recognition. We present the results of two tests in Table 2 and 3. For the first test the runtime was between 70 seconds and two minutes. The S-measure of the CTSTs was a clear indicators for result pages, i.e. they were correctly identified (cf. table 2, querying of three databases). In a second test run, all seven databases were queried (cf. table 3). Here, the results were less satisfying with the S-measure below 0 even for semantically correct results in four cases. Further research should analyze the reasons for this misclassification and come up with a refined S-measure. The runtime was about three and seven minutes respectively for each request.

Example 4 (Structural recognition — book sellers). Content-based recognition has been successfully applied for two databases in the book sellers domain starting with a sample page about the book *Harry Potter* from `www.amazon.com`.

[3] The reason is more technical: since this web-site uses redirection, our spider was not able to load the redirected page— mostly because the I.E. 6.0 usually delivers the first and only sometimes the redirected page

Table 1. Results of the test for the administrative usage level, applying content-based recognition to get new sample HTML-pages from previously unknown movie databases (example 2)

web database (URL, without `http://`)	hits	time [s]	new sample page	Sim[4]
A: `www.imdb.com`[5]	20	79.3	The Matrix	–
B_1: `cinemaniacs.mucl.de/frames`	3	39.8	The Matrix	53.2
B_2: `rasp.nexenservices.com/index.php`	1	19.9	no page	–
B_3: `www.allmovie.com`	7	32.2	The Matrix	251.7
B_4: `www.cinema.de/suche`	5	39.4	The Matrix	29.0
B_5: `movies.yahoo.com/movies`	13	65.0	The Matrix	216.3
B_6: `movies.eonline.com`	17	87.9	Matrix News	33.1
B_7: `www.scoops.be/home.asp?lang=1`	4	48.1	The Matrix	98.3
B_8: `tvguide.com/movies`	3	52.7	The Matrix	63.2

Table 2. Results of the test for the user search, applying structural recognition on HTML-pages from 3 movie databases (example 3)

user request	web database (URL)	hits	time [s]	highest ranked result page	S-value
Amélie	www.imdb.com	12	56.7	Fabuleux destin d'Amélie Poulain, Le	49
Amélie	cinemaniacs.mucl.de	3	10.2	Le fabuleux destin d'Amélie Poulain	57
Amélie	www.allmovie.com	1	11.2	no result page	
Men in Black	www.imdb.com	12	58.9	Man In Black 2	34
Men in Black	cinemaniacs.mucl.de	3	10.2	no result page	
Men in Black	www.allmovie.com	12	11.2	Men In Black	123
Harry Potter	www.imdb.com	13	87.1	Harry Potter and the Chamber ...	61
Harry Potter	cinemaniacs.mucl.de	3	10.4	Harry Potter and the Chamber ...	78
Harry Potter	www.allmovie.com	4	19.5	Harry Potter and the Chamber ...	150

Details are omitted here. Structural recognition can be applied based upon sample pages from both domains. Again, the response time was quite large with many HTML-pages to be loaded and analyzed (cf. table 4). The S-measure was a good classifier here, except for the case of `www.alibris.com`, where values of -1 and -2 occurred.

The performance seems to be rather unsatisfactorily — the answering times range between a half and two minutes. However, we have to perform many web queries on numerous databases, all of these being executed in sequel by our present prototype. At this time we have not implemented parallel querying of databases, which could lead to an enormous performance gain. Another reason for the poor performance of our prototype is given by extremely complex HTML pages, typically with 500–1500 HTML-elements containing between 100–250 text

[4] The highest similarity-value (according to (6)) of the result pages of the database B_i compared with the sample result page from A is reported.

[5] Structural recognition took place for `www.imdb.com`, with the objective to find a new sample page usable for the content-based recognition at other databases; The CTST of the new sample page about the movie *Matrix* had a S-Value of 48 relative to the previous given sample page about the movie *Lord of the Rings*.

Table 3. Results of the second test for the user search on 7 movie databases, applying structural recognition (example 3)

user request	web database (URL)	hits	time [s]	highest ranked result page	S[6]
First Contact	www.imdb.com	5	22.9	First Contact	48
First Contact	cinemaniacs.mucl.de	1	9.6	no result page	
First Contact	www.allmovie.com	9	10.7	no result page	
First Contact	movies.yahoo.com	4	25.8	Star Trek: First Contact	1.8
First Contact	movies.eonline.com	2	28.1	Star Trek: First Contact	-220
First Contact	www.scoops.be	3	38.6	Star Trek: First Contact	4.9
First Contact	www.cinema.de	1	40.6	no result page	10
Back to the Future	www.imdb.com	11	56.0	Back to the Future Part III	5.0
Back to the Future	cinemaniacs.mucl.de	1	7.9	no result page	
Back to the Future	www.allmovie.com	37	104.3	Back to the Future Part III	255
Back to the Future	movies.yahoo.com	4	25.8	Back to the Future Part II	22
Back to the Future	movies.eonline.com	10	60.3	Back to the Future	-18
Back to the Future	www.scoops.be	10	52.0	Back to the Future Part III	25
Back to the Future	www.cinema.de	1	74.5	no result page	

Table 4. Results of searching book seller databases by structural recognition (example 4)

request	URL (without http://)	hits	time [s]	highest ranked result page	S-value
A Brief History of Time	www.amazon.com	20	159.3	A Brief History of Time	36
A Brief History of Time	www.barnesandnoble.com	19	98.0	The Illustrated A Brief ...	54
A Brief History of Time	www.alibris.com	23	266.4	A Brief History of Time	-1
A New Kind of Science	www.amazon.com	12	133.8	A New Kind of Science	36
A New Kind of Science	www.barnesandnoble.com	3	24.5	A New Kind of Science	54
A New Kind of Science	www.alibris.com	0	10.2	no result page	–
First Contact	www.amazon.com	27	199.2	First Contact	58
First Contact	www.barnesandnoble.com	3	23.7	From First Contact through Reconstruction	11
First Contact	www.alibris.com	40	169.1	Star Trek: First Contact	-2

parts. Thus the DOM-tree gets large and the *realtime* creation and comparison of CTSTs becomes very costly. Different strategies can be thought of in order to overcome this problem. Besides parallelization, cutting of computation during DOM- and CTST-creation at the earliest possible time might be a good method if incompatible paths are detected at an early stage.

Satisfactorily, at second, the tests on the two domains (books and movies) establish a proof of concept, as the approach works well. In fact, structural recognition was successful for most web databases, with only one sample HTML-page given. Additionally, the detection of new sample HTML-pages from previously unknown data sources was also successful in many cases using our newly invented content-based recognition. Once a sample page was found, the database could be queried along with all the others. Hence, the database was integrated.

[6] We report the highest value of S, the similarity measure of structural recognition according to (1).

9 Summary

We have presented a new wrapper-free approach for the automated integration of web databases. More specifically, the recognition of result pages is done by structural recognition for known web databases and by content-based recognition for new web databases.

Structural recognition deploys the structural similarity of pages generated by the same web database. The query results are compared with a given sample page. This is done by use of their HTML structure. Content-based recognition is based upon the extensional overlap between web databases. Result pages from different web databases but referring to the same real-world object are matched by term-frequency based comparison.

For the chosen examples (movies and books) our experiments produced satisfying results. However, the approach depends heavily upon the initial sample pages, the chosen user request and, last but not least, upon the kind of web database content. We are convinced that our approach is a good foundation for further development of wrapper-free automated integration of web databases.

Last but not least, *WrapIt* can easily adapt to changes in web databases.If there is no fundamental change in the extension of the data, the evolution of web databases, including changes in communication, data modelling, structure, layout etc. can be managed automatically by our approach.

References

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Declarative information extraction, Web crawling, and recursive wrapping with lixto. *Lecture Notes in Computer Science*, 2173, 2001.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 109–118, Orlando, September 2001. Morgan Kaufmann.

[Coh98] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD*, Seattle, Washington, 1998.

[MN01] Janet L. Wiener Marc Najork. Breadth-first search crawling yields high-quality pages. In *Proceedings of Tenth International World Wide Web Conference*, Hong Kong, May 2001.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, pages 129–138, Orlando, September 2001. Morgan Kaufmann.

[SA99] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, 1999.

[Sal89] Gerald Salton, editor. *Automatic Text Processing*. Addison-Wesley, Reading, Massachusetts, 1989.

Enhancing ECA Rules for Distributed Active Database Systems

Thomas Heimrich[1] and Günther Specht[2]

[1] TU-Ilmenau, FG Datenbanken und Informationssysteme, 98684 Ilmenau
[2] Universität Ulm, Abteilung Datenbanken und Informationssysteme, 89069 Ulm

Abstract. ECA (event/condition/action) rules have been developed for central active database systems. In distributed active database systems the problem of inaccessibility of partial systems raises and thus the undecidability of ECA conditions referring to remote systems. This work proposes an enhancement of ECA rules for distributed active database systems to react also in the case of inaccessibility and undecidability. Therefore, the ECA evaluation will be enhanced to a strict function with the inaccessibility state Ω and a new alternative action AA enriches the classical ECA rules. The advantages and the usage of this approach are shown by an example of maintaining data consistency in distributed active database systems.

1 Introduction

Today, distributed working becomes more and more important in service enterprises, in field services, and many other areas. In most of these areas work cannot be done completely autonomously: decisions are based on local and remote data, systems have to react on changes at remote hosts or propagate own changes to remote servers. Thus during work online connections are often necessary. However, remote systems may be inaccessible occasionally. In case of active database systems this often leads to undecidable ECA conditions and rules. Up to now, this results in a dissatisfying wait state, if because of timeout people can not go on working since important information is missing.

By enhancing ECA rules with additional actions for the case of undecidability of ECA conditions (e.g. if remote systems are not reachable), it becomes possible for active databases to react alternatively, which makes the entire ECA mechanism more robust. Since abort is also an action, the classical case can be subsumed easily.

The rest of the paper is organized as follows: Section 2 starts with a brief introduction to active databases systems. Then we enhance ECA rules to strict functions and enrich ECA rules to ECA-AA rules by alternative actions. In Section 3 we show the advantage and the use of ECA-AA rules for maintaining data consistency in distributed active database systems. We summarize our work in Section 4.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 199–205, 2003.

2 The ECA Mechanism and Its Enhancement for Distributed Active Database Systems

Up to now ECA rules have been mainly used in central active database systems [2,5,6]. Simple variants of ECA rules have been integrated into SQL:99 and are available in some object-relational database systems. In these simple rules events are restricted to insert, delete or update operations. In distributed active databases, both event evaluation and condition evaluation can have an indefinite result because of unavailable subsystems. Hitherto this leads to an abort after a timeout even if this is not desirable or necessary at all. The goal of this Section is to develop and present a solution on this problem.

2.1 Active Databases (Short Repetition)

Active database systems [2,5,8] can react to occurring events using ECA rules. This ability can be used, for example, to control relationships between data objects even beyond system boundaries.

Reactions on events are specified as rules. Rules are triples of the kind (Event, Condition, Action). These ECA rules are also known as triggers or alerters. An event is something that occurs at a specific point in time. Conditions are predicates related to a database. They determine under which constraint an event is important. Conditions are optional. An action specifies what is to be done, if a situation of interest occurs, i.e., event and condition evaluate to true.

Active databases distinguish between different categories of events. The two main categories are simple and composite events. Simple events can be split into database events, time events, and abstract events. A database event is any operation on the database including start, commit, and abort of transactions. Time events are activated at a specific point in time. Using abstract events, a reaction to external events occuring outside of the database become possible. However, the system must be explicitly informed about these events. In practice, this requires the explicit activation of the rule by an application program. Simple events can be combined to composite events using logical operators.

2.2 Requirements for ECA Mechanisms in Distributed Active Database Systems

2.2.1 Decentralized Event Detection

A general architecture for heterogeneous active database systems is introduced in [6]. This architecture enables event detection within distributed active databases. The main components are a central "shared knowledge repository" and a central ECA rule base. The shared knowledge repository contains transformation information and procedures. Based on them, different data models, data manipulation languages, and object representations of diverse database systems can be accessed by an "intelligent agent". System-wide rules are also being held in a central way. Local event detectors

signal occurring events to local components, and pass all events that are of global interest to the central ECA rule base.

The disadvantage of this architecture is its central approach. Sending events to the central rule base makes it necessary to establish a connection to the rule base. For mobile systems, for example, such a connection cannot be guaranteed over a longer period of time. As a consequence a buffering of occurred events is proposed in [6]. The central event detector can react to these events only with delay. But late reactions can lead to undesired effects due to the fact that the state of the database system may have changed in the meantime. Especially, it may happen that attribute values are set to an old or incorrect value due to late update.

Furthermore, a central event detection is not adequate for distributed database systems with high autonomy degree. For this kind of system a decentralized event detection and a decentralized ECA rule base is required.

Up to now, all ECA mechanisms and architectures for distributed heterogeneous active database systems assume a very limited autonomy of the individual subsystems.

2.2.2 Strictness of ECA Rules

In central databases, the evaluation of events and conditions of ECA rules is always possible. This cannot be ensured in distributed database systems with high degree of autonomy. For them the event or condition evaluation may be indefinite (Ω) due to unaccessability. Thus our second requirement for ECA mechanisms in distributed active database systems is the strictness of ECA rules in order to treat the special case of indefiniteness.

2.3 Enhancing the ECA Mechanism for Distributed Active Database Systems with High Autonomy Degree

The ultimate goal of the enhancement is a more flexible ECA mechanism, which allows us to continue work even if subsystems are not reachable. This is achieved by adding strictness to event and condition evaluation.

We consider the condition evaluation first. The evaluation of a condition c formally corresponds to function $f(c)$ which either evaluates to TRUE or FALSE (see equation (1)). Usually c is recursively composed by $c_1, c_2, \ldots, c_n$ subconditions, which are concatenated by boolean operators. Atomic conditions are all kind of equations, inequations and boolean values. Of course, c_i can also refer data in remote subsystems and subconditions may even be evaluated on remote hosts. Thus we get:

$$f : C \rightarrow \{true, false\} \tag{1}$$

$$f(c) = h(h(\ldots h(f_{@1}(c_1), f_{@2}(c_2)), \ldots), f_{@k}(c_n)) \tag{2}$$

c_i condition i $(0 \le i \le n)$,
evaluated at (remote) subsystem j, denoted by $f_{@j}$ if important
(omitted later on) $(0 = j = k = n)$.
$@j_1$ and $@j_2$ are not necessarily different and may be even the local host

h any boolean operator

In distributed active systems, subsystems may be unreachable. Thus, $f_{@i}(c_i)$ can be indefinite. We denote this by Ω and extend both, the domain and the codomain of *f(c)*, with the Ω element (indefinite). The introduction of Ω formally turns *f(c)* into a total function. We call *f* a strict function, if holds: *f* evaluates to Ω, if any input parameter of *f* is Ω [1]. Of course, *h* has to be total and strict as well.

$$f : C \rightarrow \{true, false, \Omega\} \tag{3}$$

Analogous to condition evaluation, evaluation of events (e) can be defined as a total and strict function *g(e)*. Like *f(c)*, also *g(e)* maps to the codomain {true, false, Ω}.

$$g : E \rightarrow \{true, false, \Omega\} \tag{4}$$

true: event $e \in E$ did occurre
false: event e has not occurred
Ω : it is indeterminable whether the event e occurred or not

The firing of an ECA rule is defined as follows:

$$if\{g(e) \wedge f(c)\}\ then\ \text{execute A}\ [else\ \text{don't execute A}]\ fi \tag{5}$$

If one of the parameters in the *if*-condition is Ω, the firing of the ECA rule leads to the processing of the *else*-case (nothing happens). Up to now, indefiniteness in one of the *if*-conditions is not considered explicitly. That is the reason of enhancing the ECA mechanism with a new, alternative action (AA), which is executed in the Ω-case[1]. Of course, the alternative action can activate further rules and thus further (alternative) actions.

Enhanced ECA rules:
An enhanced ECA rule, called ECA-AA is defined as a 4-tuple (*Event, Condition, Action, Alternative Action*). The *alternative action* is executed (instead of action) when the condition evaluation of C returns Ω. An ECA-AA rule will become a traditional ECA rule if no *alternative action* is defined.

Usually we are only interested in defining alternative actions if E did occur and C is indeterminable (Ω). There are only very limited use cases where also the evaluation of E to Ω is important, like in security systems. For instance in a security control system

[1] Deviating from the definition in [1] it is completely sufficient, if the if-then-else statement is only strict concerning the condition and the entered branch (and not globally strict).

the interruption of operation of an external video camera, acting as event initiator, should cause an additional alternative action, like closing a door and ringing an alarm bell. But usually only positive events cause an ECA rule evaluation, since otherwise the absence of any external event initiator would cause an infinite call of AA, which is in general not intended. We distinguish between both cases. Herewith, the ECA evaluation definition (5) becomes:

Usual mode:

if $\{g(e) \wedge f(c)\}$	*then*	execute action A
else if $\{g(e) \wedge f(c) = \Omega\}$	*then*	execute alternative action AA
	else	do not execute any action

fi

Security mode:

if $\{g(e) \wedge f(c)\}$	*then*	execute action A
else if $\{g(e) \wedge f(c) = \Omega\}$	*then*	execute alternative action AA_C
else if $\{g(e) = \Omega\}$	*then*	execute alternative action AA_E
		/* usually includes suspending this rule in order to avoid infinite calls */
	else	do not execute any action

fi

3 Using Enhanced ECA Rules for Maintaining Data Consistency

In the following we show how ECA-AA rules (in the usual mode) can be used in order to guarantee data consistency in a distributed active database system.

3.1 Specification of Consistence Constraints

Dependences between data objects can generally be described by the tuple <S,D,P,C,A>, also known as D^3 (data dependency descriptor) [7]. S stands for the source objects and D for the destination objects. Source objects and destination objects can be arbitrary database objects (e.g., tables, tuples, attribute values).

P is a predicate which describes the data dependencies between source and destination objects. According to the ECA rules, the point in time at which P evaluates to true can be considered as an event.

C specifies a condition that, if fulfilled, leads to the execution of action A. C also can specify a point in time at which P must be true. It is worth mentioning that C specifies no consistency conditions about the dependencies between source and destination objects (see example below). A is an action which can call further actions and which must be executed to achieve the consistency of the overall system. This action makes sure that P is fulfilled.

The use of the tuple <S,D,P,C,A> is illustrated by the following example. The source objects are the attributes s_1 to s_n, which are distributed over different databases

on different computers. These computers can be mobile computers, like laptops, which are not permanently reachable. The destination is supposed to be the attribute *d*. Destination objects and source objects are in a consistent state if $s_1 + ... + s_n = d$ is satisfied (e.g., planned amount of money for the adjustment of an insured loss must be greater or equal than the sum of all partial damages). This consistency condition is only valid if attribute *c* is greater than 100. Attribute *c* can also reside on a remote database.

The notation using the tuple <S,D,P,C,A> looks as follows:

S:	$s_1 , ... , s_n$	source objects
D:	*d*	destination object
P:	$s_1 + ... + s_n = d$	consistency relationship between source objects and destination object
C:	*c>100*	consistency condition
A:	$d := s_1 + ... + s_n$	action

Inaccessibility of systems can always occur in distributed databases. P or C can be indefinite in the above example. As a consequence it is also indefinite whether action A is to be executed or not.

We enhance the tuple <S,D,P,C,A> with an entry for alternative action (AA). Then it is possible to execute a defined action even in case of indefiniteness of P or C. In the above example an alternative action may set the attribute *d* to a maximal value. The notation of the example with the new tuple <S,D,P,C,A,AA> looks as follows:

S:	$s_1 , ... , s_n$	source objects
D:	*d*	destination object
P:	$s_1 + ... + s_n = d$	consistency relationship between source objects and destination object
C:	*c>100*	consistency condition
A:	$d := s_1 + ... + s_n$	action
AA:	*d := maximal value*	alternative action

3.2 Transformation into Enhanced ECA Rules

Enhanced ECA rules may directly evaluate Data Dependency Descriptors of the form <S,D,P,C,A,AA>. The following rule shows the general mapping of the tuple <S,D,P,C,A,AA> to an enhanced ECA rule and, in addition, an instantiation on the base of the above example.

Event:	not P	Point in time on which $d >= s_1 + ... + s_n$ is not true for the first time.
Condition:	C	*c>100*
Action:	A	$d := s_1 + ... + s_n$
Alternative Action:	AA	*d := maximal value*

3.3 Advantages of Enhanced ECA Rules

Using enhanced ECA rules, a system architecture without central event detection and central rule base can be built. Therefore every subsystem must consist of an active database system, and it must be able to detect events across systems.

Every subsystem can specify its consistency conditions in the form of enhanced ECA rules. Thus a decentralized rule base is build up. The event detection is decentralized, too, because every subsystem can also detect events in remote subsystems.

With the proposed ECA-AA rules every subsystem can react in case of indefinite event or condition evaluation. Thereby a high degree of autonomy of the subsystems is provided. Data consistency in distributed database systems can only be achieved if the indefinite event and condition evaluation is taken into account.

4 Conclusions

This paper proposes an enhancement of the well-known ECA rules. Traditional ECA rules are enhanced by an element for *alternative actions*. The alternative action is executed if the event or condition evaluation is indefinite. In contrast to traditional ECA rules, the new ECA-AA rules always provide a defined reaction. An example about maintenance of data consistency in distributed active database systems has shown the practical applicability of the approach.

References

[1] Bauer F.L., Wössner H.: *Algorithmische Sprachen und Programmentwicklung*, Springer-Verlag 1981

[2] Dittrich K. R., Gatziu S.: *Aktive Datenbanksysteme – Konzepte und Mechanismen,* dpunkt.verlag 2000

[3] Helal A. A., Heddaya A. A., Bhargave B. B.: *Replication Techniques in Distributed Systems*, Kluwer Academic Publishers 1996.

[4] Oezsu M. T., Valduriez P.: *Principles of distributed database systems,* 2nd Ed. Prentice-Hall 1999.

[5] Paton N. W.: *Active Rules in Database Systems,* Springer-Verlag 1998

[6] Pissinou N., Vanapipat K.: *Active Database Rules in Distributed Database Systems*. Intl. Journal of Computer Systems, 11(1), January 1996, pp. 35–44

[7] Rusinkiewicz M., Sheth A., and Karabatis G.: *Specifing interdatabase dependencies in a multidatabase environment*. IEEE Computer, 24(12), December 1991. Special Issue on Heterogeneous Distributed Databases, pp. 46–53.

[8] Zimmermann J.: *Konzeption und Realisierung eines aktiven Datenbanksystems: Architektur, Schnittstellen und Werkzeuge,* Logos-Verl., 2001

Improving XML Processing Using Adapted Data Structures

Mathias Neumüller and John N. Wilson

Department of Computer and Information Sciences
University of Strathclyde in Glasgow, Scotland, U.K.
{mathias,jnw}@cis.strath.ac.uk

Abstract. From its origins in document processing, XML has developed into a medium for communicating all kinds of data between applications. More recently, interest has focused on the concept of native XML databases. This paradigm requires that database queries can be resolved by direct searching of XML data structures. Relational databases can be compressed without the loss of direct addressability. A similar approach can be applied to XML data structures. Compression in the relational paradigm is associated with improved performance. We review this approach and show results from the implementation of a prototype compressed DOM. Our research indicates that it is possible to optimise queries over compact XML structures by choosing appropriate physical representations.

1 Introduction

Emerging standards such as XPath and XQuery are founded on the vision of XML as both a standard for document and data interchange between applications and also as a structure that may need to be queried directly in much the same way as a database system. Whilst the principles of querying hierarchical data structures were developed early in the history of computer science [24], further development of direct querying capability for XML data sources requires close attention to be paid to issues of acceptable performance. Whereas in hierarchical databases the designer had full control over the physical representation and database schema, XML documents are only defined in terms of a common data model and a textual representation. Physical storage varies widely and schema design is often an ad hoc process carried out by designers with skills in an application domain rather than specific database design skills. Emerging native XML databases (NXDs [22]) are designed to make XML applications independent from the physical storage in much the some way as relational databases. They offer tailor-made storage solutions for XML documents and allow access to the data using a standardised interface such as the Document Object Model (DOM). We are exploring the consequences of applying compression directly to appropriate XML data to maximise the use of main memory storage in query processing. The varied nature of XML documents suggests that a range of algorithms are

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 206–220, 2003.

necessary to optimise the performance benefits that can be achieved by efficient internal representation. The aim of our research is to develop a framework that enables the choice of optimally efficient data representation and processing techniques based on the analysis of XML data structures. Due to the inherent performance limitation of external XML representations such as database mappings and textual XML files we are focusing our research on native, memory internal representations.

We start our paper by giving a short review of related work in section 2. We then show how compressed representations in the relational field are used already and how this can be adapted for semistructured data (section 3). Section 4 describes the DDOM system, a prototype of a compressed DOM implementation based on dictionaries. We have measured memory consumption and have preliminary results for query performance (section 5). In section 6 we discuss some of the research issues that need to be explored before the compression technology can be applied in the context of a native XML database system.

2 Related Work

Significant research has already been carried out in the area of integrating XML data into relational and object-relational databases. There are many approaches to breaking up XML data into tuples, each with its own advantages and disadvantage. Some approaches interpret the XML data as a graph and store it in form of edges and nodes [6]. This approach is very flexible, but requires many joins upon querying or reconstruction. An improvement of this scheme is to store the graph as a set of complete paths and nodes [20]. However, both approaches fail to exploit any regularity that may occur in the XML structure or data. Other approaches, usually based on object-relational databases break the XML data into bigger, more complex blocks [10]. If similar structures occur in different parts of the document, these can be stored and queried together. However, most of these approaches require the presence of an XML Schema or a DTD in order to generate the necessary complex data types and relations and are thus less flexible. In the case of completely irregular data or loosely defined schemata they lead to the creation of a large number of sparsely populated tables, which is also disadvantageous. There are also hybrid mappings that try to combine advantage of both approaches [17].

In relation-based data structures, data compression is often useful for accelerating performance. Conventional databases usually represent data items as strings [7] although it is also possible to use fixed length pointers to compress domain values [16]. We have been able to exploit this compact representation of relational data [4]. New research in this field shows how to compress the data even further, using advanced compression techniques on the required dictionary structures [9]. A flexible framework used to apply optimised compression algorithms in the context of query results was described in [3].

Compression in the context of XML is mainly focused on efficient transmission. Research exploiting the semistructured nature of XML documents has

been carried out [13], resulting in a serial compression algorithm for XML documents. The performance of this approach exceeds the compression ratio achievable with general-purpose compression algorithms. However, in a database environment content addressability is necessary, therefore such methods are not applicable. Some of the already available native XML databases, like Tamino [19] and Xindice [21], also use compression techniques. However, the compression is used for secondary storage only and typically based on a conventional serial compression algorithm. The binary XML format [25] suggested by the WAP forum uses tokenized representation of the XML tags, but does not apply this approach to the document data.

3 Architecture

The relational database model is founded on the concept that data can be normalised into regular table structures. This is a useful simplification but many applications, especially Internet-based information systems, require the storage and processing of irregular data structures. A data model that supports the representation of semistructured data has the potential to overcome the limitations of relational structures. Despite its widespread use, data centric XML applications tend to suffer from poor performance of the underlying technology since this is based on assumptions of document-centricity rather than data-centricity.

3.1 Fundamentals and Assumptions

Compression is a central point of our approach to achieve better performance. Our work on relational database systems shows that compression can result in significant performance benefits by moving more of the workload from secondary into primary storage, i.e. from relatively slow disk storage into fast RAM. However, this approach can only be successful if individual elements remain accessible, thus serial, variable length compression algorithms such as LZW [26] are not appropriate. Our current method of compression enables us to compress the data off-line and resolve queries by decompressing only the output data. Dictionary encoding shows good compression ratios for relational data as well as for verbose XML documents, especially if they are machine generated using typically a rather small vocabulary.

3.2 Compressing Relational Data

The benefit of dictionary-based storage methods is that data can be represented using only minimal bit fields. At the same time direct addressability that is a fundamental requirement of efficient database models is preserved. First we will show how this can be applied in the simpler, relational case. Figure 1 shows some example data about sports clubs. Note that the data is already in third normal form and thus fairly compact. However, by using minimal bit strings to represent values, the data elements can be stored in an even more compact

MEMBERSHIPS	
MEMBER	**ACTIVITY**
Miller	Volleyball
Miller	Golf
Smith	Volleyball
Wood	Golf

ACTIVITIES	
ACTIVITY	**LOCATION**
Rugby	Activities room
Volleyball	Gym
Golf	Gym

Fig. 1. Structure of the uncompressed example relations

MEMBERSHIPS	
MEMBER	**ACTIVITY**
00	01
00	10
01	01
10	10

ACTIVITIES	
ACTIVITY	**LOCATION**
00	0
01	1
10	1

Fig. 2. Structure of the compressed example relations

form. There is, of course, still the overhead of dictionaries needed to convert the tokens, although these can also be compressed [9].

The relation data would typically be stored in tables with fixed length fields. Thus the table **ACTIVITIES** would have a size of (10+15) characters $\times$3 tuples $\times$8 bits/character = 600 bits.

Inspection of the tables suggests that it would be possible to represent the information content in a more compact form by using codes to represent the domain values rather than using the domain values themselves in the relation. The attribute **ACTIVITY** contains three different values therefore in its most compact form, it could be represented as a two bit integer. Since there are only two values in the **LOCATION** attribute, it could be represented as a one bit integer. The compressed integer representation of the relations is shown in Figure 2.

The effect of representing data in this encoded format is to reduce the space occupied by **ACTIVITIES** to 9 bits and **MEMBERSHIPS** to 16 bits. To this it is necessary to add the dictionary that allows the tokens to be converted to their string equivalents and for the reverse process to take place (Figure 3). Dictionaries can be represented as lists of domain values. In the case of all non-unique attributes the number of entries in the dictionary will be less than the number of tuples in the relation.

3.3 Compressing Semistructured Data

One possible representation of this data as XML is shown in part in Figure 4. It can be seen that the complete representation would contain almost the same redundancy that is exploited in the approach shown above for relations. The only reduction in redundancy is achieved by allowing set valued attributes, here

ACTIVITY
Rugby
Volleyball
Golf

MEMBER
Miller
Smith
Wood

LOCATION
Activities Room
Gym

Fig. 3. Dictionaries of the compressed example relations

two **activities** are stored below one **member** entry. But there also exists further redundancy caused by repetitions in the tag structure. Depending on the degree of flexibility of an associated DTD or XML Schema, the structure may be known in advance. Even if only well-formedness is assumed, the name of any closing tag is guaranteed by the XML syntax and thus redundant.

The structure of the compressed representation is shown in Figure 5. Our implementation of the compression strategy results in the structure being separated from the content. The tokenized structure representation is very closely related to the textual representation, allowing mixed content, comments and other XML specific data items to be included. As in the original XML document, the order of individual entries is important as it encodes sibbling order. In combination with special start and end tags it also encodes ancestor/descendant relationships. References from the structure point to different dictionaries that store the document data. Note that the metadata, e.g. element and attribute names, are stored in a global context, whereas the data content is stored in context dependant dictionaries. This allows related information to be kept together. Multiple entries of the same string within one dictionary domain are avoided, thus reducing the redundancy. The compression algorithm currently used for our data structure is applied equally to data and metadata. This approach results in significant reduction in the volume of the data stored.

3.4 Querying Compressed Data

There are two fundamentally different ways to query compressed data sources. The entire data can be decompressed and then queried in the uncompressed domain. Alternatively, one can compress the query and then resolve it on the compressed data, decompressing only the result set.

Using the first approach, it is possible to benefit from compression only if the retrieval of the compressed file followed by its decompression takes less time than the retrieval of the larger, uncompressed version. However, the smaller the selectivity of a query, the less efficient this approach will be. Many queries in data-centric applications will return only a small sub-set of the actual data. Thus it seems to be more desirable to translate the query itself into the compressed domain and only to return and decompress the actual result set.

In the case of dictionary compression this is very easy. The lexemes occurring in the query are sought in the document dictionaries. If no matching entry exists, the query will yield no result. If matching tokens exist, these in turn are sought in the document structure. Comparisons of these short binary tokens are typically

```
<club>
     ...
     <activities>
          <activity>
               Volleyball
          </activity>
          <location>
               Gym
          </location>
     </activities>
     ...
     <memberships>
          <member>
               Miller
          </member>
          <activity>
               Volleyball
          </activity>
          <activity>
               Golf
          </activity>
     </memberships>
     ...
</club>
```

Fig. 4. The club example data represented in XML

Type	#
Document	-
Element	1
...	
Element	2
Element	3
Text	1
/Element	3
Element	4
Text	1
/Element	4
/Element	2
...	
Element	5
Element	6
Text	1
/Element	6
Element	3
Text	2
/Element	3
Element	3
Text	3
/Element	3
/Element	5
...	
/Element	1
/Document	-

#	**Element**
1	club
2	activities
3	activity
4	location
5	memberships
6	member

#	**Text:activity**
1	Rugby
2	Volleyball
3	Golf

#	**Text:location**
1	Activities room
2	Gym

#	**Text:member**
1	Miller
2	Smith
3	Wood

Fig. 5. The structure (l.) of the compressed XML document together with the associated dictionaries (r.)

faster than string comparisons. However, using this architecture without any indices, a linear scan through the structure has to be performed. This may be slower, especially in the semistructured case when long path expressions need to be checked.

4 Implementation

The implementation of the compressed relational system has already been described in [4]. Since our Java implementation of the Dictionary compression based Document Object Model (DDOM) was developed independently of this system, implementation details of the compressed relational systems will be omitted from this paper. The DDOM is based on the architecture described in the previous section. For reasons of technical simplicity it currently supports read-only access on a document once it is parsed or generated. We believe that this

limitation is acceptable for the targeted application area, e.g. mining of large, static, scientific data sets. It is not however, a fundamental limitation of the design. Furthermore we do not support persistence at the moment, which would be required in a production system.

One of the major differences between dictionary compression in relational systems and semistructured data is the definition of the dictionary domains. In the relational case this is relatively simple. Every attribute of a given relation is associated with its own domain. All values of this attribute lie within this domain. The only difficulty that can arise is that more than one attribute may share one domain, which can be hard to detect automatically. In the case of XML data the concept of domains is less well defined. As a first criterion we used the node type to distinguish different domains. Element and Text nodes for example exist in different domains. Additionally, those nodes that represent leaves in the DOM tree, e.g. Text nodes, are stored in sub-domains formed by their immediate parent node. In the example document, the values of the Text nodes for "Miller", "Smith" and "Wood" are stored in the domain **Text:member_name** as shown in the right part of Figure 5. Note that this is a technical simplification due to the fact that no type information is available in schema-less XML. Even if schema information is available, domains remain arguable. In the XML version of Shakespeare's plays[1] for example, **TITLE** elements occur in many contexts, for example inside the **PERSONAE**, **ACT** and **PLAY** elements. These form semantically different domains (e.g. *play titles* opposed to *act titles*). However, syntactically there is just one definition of the **TITLE** *type*. In our current approach all titles would be stored in one common dictionary, following the syntactical definition. This also helps to avoid the creation of too many sparsely populated dictionaries. The problem of identifying which elements belong to which domain can not be resolved automatically in absence of a precise schema. The approach currently used is simple but works well for data-centric documents that usually do not contain highly nested data structures.

The difficulty of separating individual domains also influences the representation of the structure. The structure of a document is stored in the form of an array as shown in the left part of Figure 5 with both columns represented by binary number types. The type of an entry is stored in a 8-bit type. Because references to all possible domains can occur in almost every possible context, we so far have been unable to use minimal bit patterns to store the references. Currently 32-bit integers are used, which represent a significant waste, especially for domains with very low cardinality.

The structure array together with the associated dictionaries contain the entire data of an XML document. No tree of DOM nodes is stored to reduce memory consumption. Only the Document node that contains the structure and the dictionaries exists at any given time. However, the methods supplied by the DOM interface are required to return Node objects. These are generated dynamically and contain only a reference into the structure array. As soon as no further external reference to such a Node object exists, it can be garbage

[1] Available at `http://www.ibiblio.org/xml/examples/shakespeare`.

collected. Internally the DDOM uses methods that work directly on the structure array to avoid the overhead of frequent object generation and destruction. Externally however this mechanism is necessary to achieve DOM conformance.

The DOM interface does not support querying directly. Therefore we were required to use an external engine that works on top of this interface to perform XQL queries. However, the method *getElementsByTagName* of the DOM node types Document and Element allows the selection of ancestor nodes by name. Our implementation of this method follows the idea of doing as much work in the compressed domain as possible. Hence the tag name that is passed in as argument is sought in the dictionary containing the element names and its compressed representation is then sought in the corresponding part of the structure array. This is currently done by linear scanning, resulting in a $\mathcal{O}(n)$ runtime behaviour. At least for higher level elements, where n is of significant size this would need to be improved upon using indexes. The current approach does however avoid costly string comparisons and is able to resolve queries for non-existing tagnames early.

5 Performance

Test documents with various sizes were generated from a database used by a domain name server (DNS). The individual entries contain fields for the server name, the four parts of its numeric IP address and up to six parts of its symbolic domain name. In the case of XML (Figure 6) only those parts of the domain name that are present are stored, whereas in the original database null values are used to represent missing entries.

5.1 Memory Consumption

Figure 7 shows the memory consumption of several representations of the same XML data as a function of the database cardinality. Compressed and uncompressed textual XML documents are compared with different DOM representations. Our DDOM requires less memory than the popular Xerces and Crimson implementations[2], which both required the heap to be enlarged from 64 MB to 256 MB in order to process the largest document in this measurement. However, even the DDOM representation required still more memory than the textual representation. This is partly caused by the Java implementation as it always uses 16 bit representations for characters whereas the XML file is using 8 bit encoding. Both conventional DOM implementations show a linear growth with database cardinality. By contrast, the DDOM shows sub-linear growth. We use the gzip compressed file size as a practical measure of the document entropy. The graph clearly shows that there is still a large potential for further memory savings.

[2] Both available at `http://xml.apache.org`.

```
<?xml version="1.0" ?>
<server-LIST>
  <server>
    <HOSTNAME>web0</HOSTNAME>
    <LEVEL0>uk</LEVEL0>
    <LEVEL1>ac</LEVEL1>
    <LEVEL2>strath</LEVEL2>
    <LEVEL3>cis</LEVEL3>
    <LEVEL4>www</LEVEL4>
    <IP0>130</IP0>
    <IP1>159</IP1>
    <IP2>196</IP2>
    <IP3>115</IP3>
  </server>
</server-LIST>
```

Fig. 6. Example entry of the DNS database in XML format

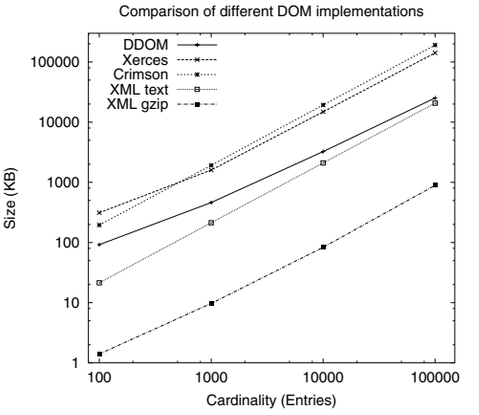


Fig. 7. Memory consumption of different representations of a DNS database (taken from [15])

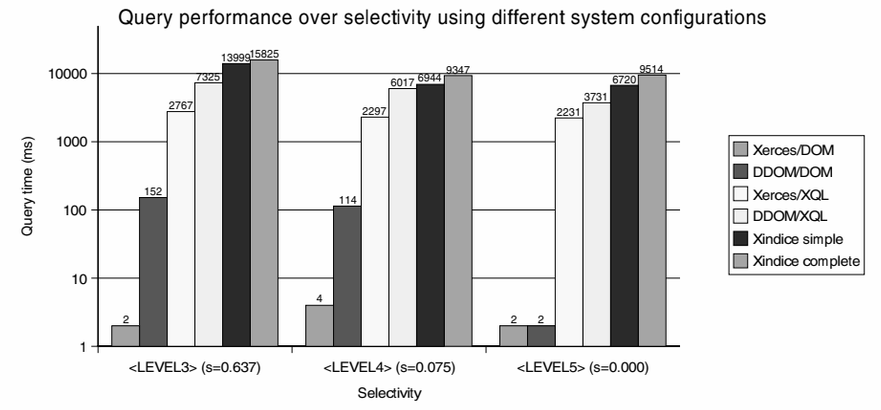


Fig. 8. Query performance of different query and storage systems for queries with different selectivity

5.2 Query Performance

Measuring the performance of a storage/query system for XML in itself is somewhat complicated. The requirements of different applications vary widely. This is reflected by a growing number of XML query languages [2]. Standard benchmarks for XML databases are just emerging [18]. Since XML is based on the idea of documents, document-centric storage systems are further developed than data-centric ones. Queries in the document-centric domain are typically limited

to locating entire documents that match certain requirements or to the execution of relatively simple transformations. Therefore storage is often based on information retrieval systems using full-text indices. In contrast, data-centric applications frequently use relational databases as a back end. Queries in such systems are typically aimed at retrieving only a small fraction of a set of documents.

Figure 8 shows some preliminary query results, indicating some of the problems and possibilities in terms of the performance of queries posed on XML data. Querying for the same results using different query and storage strategies and systems shows a wide range of performance variations. The measurements were performed using a single XML document containing 10,000 entries from the DNS database. The query selects elements with a certain name, in this case the elements containing the fourth, fifth and sixth component of the domain names. The selectivity of this query is decreasing as almost all domain names contained in the example document have four parts, but none has six parts. The chosen query may look overly simple, but provides the chance of comparing the DOM interface directly to any higher level query system. Although of limited practical use on its own, it is the most commonly used selection operator in XML, thus preceding almost every other query. We also think that this query can be used to show the variance in performance achievable.

Again the DDOM was compared against the Xerces implementation. The query can be directly resolved by using the *getElementsByTagName*("LEVELn") method. To allow a comparison with a more realistic system, that would be capable of handling more complicated queries, the query was repeated with an XQL query engine on top of the two DOM implementations ("//LEVELn"). The query engine used was taken from the GMD-IPSI XQL Engine[3] implementation and is not optimised for use with either of the implementations and thus restricted to the standard DOM interface. DOM representation, query engine and the proprietary query application run in a single Java virtual machine (JVM) with a default maximum memory space of 64 MB. Finally the measurements were repeated using a native, disk-based XML database, in this case Xindice [21]. No indexes were generated to allow a comparison with the DOM implementations that also do not use indexes. For Xindice, measurements were performed on a warm cache. Database engine and query application run on a single computer but in different JVMs. The server runs in a JVM with a maximum of 168 MB of memory, the client uses the JVM default setting of 64 MB. Xindice is accessed using its XML:DB interface and supports XPath for querying. The query can be stated in two different ways. The simple version ("LEVELn") just locates elements with the given name, whereas the more complex version ("//LEVELn") searches for these as descendants of the document root. Although these two queries are semantically identical, query times vary measurably.

Four orders of magnitude lie between the limited but direct access to main-memory based DOM representations and the flexible but slow disk-based NXD. However, using a separate query engine on top of the DOM implementations

[3] Available at `http://xml.darmstadt.gmd.de/xql`.

consumes most of this performance advantage, despite the fact that the entire querying process is performed in the computer's main memory. It also becomes apparent that the DDOM implementation suffers from the dynamic object creation as its performance diminishes with higher query selectivity. This limitation is exaggerated by the fact that the external XQL engine performs four complete traversals of the DOM tree using only methods supplied by the Node interface, rather than using the more specialised methods of the Element or Document nodes that are able to perform the required task much faster.

5.3 Limitations

The experiments with external query engines based on the DOM interface showed that this interface severely limits the performance of data-centric applications as shown in Figure 8. This result, together with some new ideas about more efficient, adapted compression algorithms have led to a general overhaul of the developed structure. We are reviewing the representation of the document structure and content and the dynamic node generation mechanism. Partial approaches to these problems are described in the following section.

6 Future Research

We plan to exploit the experience gained from the implementation of the compressed relational and semistructured systems for future research in the area of compressed in-memory databases. We anticipate focusing our effort on several issues needed to transform our compressed DOM to a main-memory database for semistructured data. We need a high-performance query system that works with and utilises the advantages of a compressed representation. This will be based on an adaptive architecture that encodes content dependent on its characteristics. This should further help to improve compression ratios. We also need to support queries that run over more than one document at a time. Indexing is an important influence on query performance and we have started to address this issue. A review of available techniques and analysis of their possible uses seems to be necessary. All these conceptual changes will depend on a different representation of the document structure.

6.1 High-Performance Query Systems

To achieve better query performance we plan to develop an interface to the compressed data at a lower level than that provided by the DOM. The need to instantiate many nodes of a DOM tree during traversal could be overcome by the use of a cursor to access the data. As in the relational case, higher level, non-procedural query languages should allow for optimisation of the query execution. We will therefore aim to produce an optimised query system that bridges the gap between low-level data representation and high-level query language. The question as to which query language to support is not decided yet. XQuery will

become the recommended standard of the W3C but has not yet reached this state yet. The expressive power of other query languages such as XQL or XPath is slightly more limited but may be sufficient for the targeted application area. These alternatives are currently more mature and widely accepted.

6.2 Domain Specific Compression Methods

Currently the compression mechanism is closely coupled with the DDOM implementation and it is hard to separate the compression algorithm from the rest of the implementation. Recently we have separated the dictionary part using a pluggable interface to allow experiments with different dictionary implementations. Experience from the relational architecture has shown that significant memory is taken by the dictionaries. New research in this area [9] has shown that useful improvements in compression can be achieved by using compressed domain dictionaries. It is our goal to separate the compression mechanism used from the external interface to allow experimentation with different implementations.

We further hope to bridge the division between data-centric and document-centric XML by selecting different compression methods for different documents or even fragments of a document. Adapted compression methods are widely used in other information storage systems [27] and should be beneficial for our purposes too. Choosing appropriate compression methods could be done using emerging metrics defined for XML data [11]. An framework for adaptive compression of query results has already been described in [3]. This will need to be modified in order to cope with the properties of semistructured data.

6.3 Support for Document Collections

Although it is currently possible to work with multiple compressed documents in parallel, the advantages of applying the compression techniques across such a collection are quite limited. Compression and querying is handled on a per document level. Especially in the targeted context of data-centric applications, many documents share a common structure and may share a high degree of common content. Applying the compression technique across such a collection of documents will result in an improved compression ratio compared to the separate compression of individual documents. This is mainly caused by the relatively smaller overhead of the dictionaries. The merged dictionaries of two similar documents are typically smaller than the size of the two separate dictionaries.

Spanning the compression across several documents becomes even more important for querying than it is for compression. NXDs typically allow documents to be queried collectively. If compression is applied on document level, the compressed form of the same content may have different representations in different documents. Thus a query across several compressed documents would have to be transformed for every document to be queried. This can be avoided if documents that are typically queried together are also stored together in a common compressed domain.

6.4 Indexing

Indexing is an important building block of high performance database operations. However, indexing in the context of XML is complicated by the fact that domains are often unknown and the structure contained within a document may or may not be of importance. So far we have not analysed this issue far enough to give a conclusive estimation of its importance for our research. We already have experimented with value based indexing within a domain given by the identified dictionaries. Initial results are encouraging but incomplete without further investigation of structural indexing. This however is closely related with the structural representation of the documents. We will aim to close this gap in the future and refer to literature such as [12] and [5] for a first impression of this important topic.

6.5 Structural Representation

Because the current structural representation is the main cause of performance limitation, both in terms of compression ratio and query performance, we will investigate the use of other data structures. At the time of writing, a representation based on the idea of DataGuides as introduced in [8] seems to be most likely. However, unlike the Lore database system [14] for which this technique was developed and which uses DataGuides as a means of querying, we want to store the document data within the DataGuide. This approach separates data domains from each other and allows statistical information to be stored together with the data itself. This can be used to improve compression and also as a structural index for the contained data, consequently further enhancing query performance.

7 Conclusion

The DDOM prototype implementation demonstrates a significant space saving compared with standard DOM implementations. The evidence suggests that the entropy of typical data-centric XML documents is such that considerable savings beyond those we have achieved should be possible. Because of the wide variety of XML-based applications, the influence of the type of data needs to be analysed more closely. Adaptive techniques may be possible that compress domain specific data more effectively. At the lowest level, we have represented tokens as integers rather than minimal bit-strings and further savings in space may be achieved by optimised data structures. Many of the issues surrounding the efficient handling of data centric XML applications are not solved yet. The compression algorithm we have used needs further refinement and other algorithms need to be tested out and compared with the current technique.

In the relational domain, we have previously demonstrated that our approach to compression provides significant benefits in terms of enhanced query performance. The results we report here suggest that similar benefits can be achieved

for querying XML data structures if the queries can be resolved in the compressed domain. We have yet to developed a suitable strategy for querying the compressed XML data directly and expect to be able to make further significant improvements in the performance.

The performance of future Internet-based applications will be determined in part by the ease with which semistructured data can be queried and transferred. The work we describe suggests that compression has the potential for enhancing these processes and accelerating application performance in a resource-hungry environment.

References

[1] Peter M. G. Apers, Paolo Atzeni, et al., editors. *Proceedings of the 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Morgan Kaufmann, 2001.

[2] Angela Bonifati and Stefano Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, 2000.

[3] Zhiyuan Chen and Praveen Seshadri. An algebraic compression framework for query results. In *ICDE 2000*, pages 177–188, San Diego, California, USA, 2000. IEEE Computer Society.

[4] W. Paul Cockshot, Douglas McGregor, and John Wilson. High-performance operations using a compressed database architecture. *Computer J.*, 41(5):283–296, 1998.

[5] Brian Cooper, Neal Sample, et al. A fast index for semistructured data. In Apers et al. [1], pages 341–350.

[6] Daniela Florescu and Donald Kossmann. Storing and querying XML data using an RDBMS. *Data Engineering Bulletin*, 22(3):27–34, Sep 1999.

[7] Hector Garcia-Molina and Kenneth Salem. Main memory database systems: An overview. *IEEE Trans. Knowledge Data Eng.*, 4:509–516, 1992.

[8] Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In Matthias Jarke, Michael J. Carey, et al., editors, *VLDB 1997*, pages 436–445. Morgan Kaufmann, 1997.

[9] Abu Sayed M. L. Hoque, Douglas R. McGregor, and John N. Wilson. Database compression using of-line dictionary methods. Technical report, Department of Computer and Information Science, University of Strathclyde, Glasgow, Scotland, UK, 2002.

[10] Meike Klettke and Holger Meyer. XML and object-relational database systems. In Suciu and Vossen [23], pages 151–170.

[11] Meike Klettke, Lars Schneider, and Andreas Heuer. Metrics for XML document collections. In Akmal Chaudri and Rainer Unland, editors, *XMLDM Workshop*, pages 162–176, Prague, Czech Republic, Mar 2002.

[12] Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In Apers et al. [1], pages 361–370.

[13] Hartmut Liefke and Dan Suciu. XMill: An efficient compressor for XML data. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD 2000*, volume 29 of *SIGMOD Record*, pages 153–164, 2000.

[14] J. McHugh, S. Abiteboul, et al. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, Sep 1997.

[15] Mathias Neumüller and John N. Wilson. Compact in-memory representation of XML data. Technical report, Department of Computer and Information Science, University of Strathclyde, Glasgow, Scotland, UK, 2002.

[16] P. Pucheral, J.-M. Thevenin, and P. Valduriez. Efficient main memory data management using the DBGraph storage model. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *VLDB 1990*, pages 683–695. Morgan Kaufmann, 1990.

[17] Albrecht Schmidt, Martin Kersten, et al. Efficient relational storage and retrieval of XML documents. In Suciu and Vossen [23], pages 137–150.

[18] Albrecht Schmidt, Florian Waas, et al. Why and how to benchmark XML databases. *SIGMOD Record*, 30(3):27–32, 2001.

[19] Harald Schöning. Tamino – a DBMS designed for XML. In *ICDE 2001*, pages 149–154, Heidelberg, Germany, 2001. IEEE Computer Society.

[20] Takeyuki Shimura, Masatoshi Yoshikawa, and Shunsuke Uemura. Storage and retrieval of XML documents using object-relational databases. In Trevor J. M. Bench-Capon, Giovanni Soda, and A. Min Tjoa, editors, *DEXA' 99*, volume 1677 of *LNCS*, pages 206–217. Springer, 1999.

[21] Kimbro Staken. Introduction to dbXML. *XML.com*, Nov 2001. `http://www.xml.com/pub/a/2001/11/28/dbxml.html`.

[22] Kimbro Staken. Introduction to native XML databases. *XML.com*, Oct 2001. `http://www.xml.com/pub/a/2001/10/31/nativexml.html`.

[23] Dan Suciu and Gottfried Vossen, editors. *The World Wide Web and Databases, Third International Workshop WebDB 2000, Dallas, Texas, USA, May 18-19, 2000, Selected Papers*, volume 1997 of *LNCS*. Springer, 2001.

[24] D. Tsichritzis and F. H. Lochovsky. Hierarchical data-base management: A survey. *ACM Computing Surveys*, 8(1):105–123, 1976.

[25] WAP Forum, Ltd. *Binary XML Content Format Specification*, version 1.3 edition, Jul 2001. `http://www.wapforum.org`.

[26] T. A. Welch. A technique for high performance data compression. *IEEE Computer*, 17(6):8–20, Jun 1984.

[27] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, second edition, May 1999.

Comparison of Schema Matching Evaluations

Hong-Hai Do, Sergey Melnik, and Erhard Rahm

Department of Computer Science, University of Leipzig
Augustusplatz 10-11, 04109 Leipzig, Germany
{hong, melnik, rahm}@informatik.uni-leipzig.de
dbs.uni-leipzig.de

Abstract. Recently, schema matching has found considerable interest in both research and practice. Determining matching components of database or XML schemas is needed in many applications, e.g. for E-business and data integration. Various schema matching systems have been developed to solve the problem semi-automatically. While there have been some evaluations, the overall effectiveness of currently available automatic schema matching systems is largely unclear. This is because the evaluations were conducted in diverse ways making it difficult to assess the effectiveness of each single system, let alone to compare their effectiveness. In this paper we survey recently published schema matching evaluations. For this purpose, we introduce the major criteria that influence the effectiveness of a schema matching approach and use these criteria to compare the various systems. Based on our observations, we discuss the requirements for future match implementations and evaluations.

1 Introduction

Schema matching is the task of finding semantic correspondences between elements of two schemas [11, 14, 16]. This problem needs to be solved in many applications, e.g. for data integration and XML message mapping in E-business. In today's systems, schema matching is manual; a time-consuming and tedious process which becomes increasingly impractical with a higher number of schemas (data sources, XML message formats) to be dealt with. Various systems and approaches have recently been developed to determine schema matches (semi-)automatically, e.g., Autoplex [2], Automatch [3], Clio [1, 18], COMA [7], Cupid [14], Delta [6], DIKE [19], EJX [10][1], GLUE [9], LSD [8], MOMIS (and ARTEMIS) [4, 5], SemInt [11, 12, 13], SKAT [17], Similarity Flooding (SF) [15], and TranScm [16]. While most of them have emerged from the context of a specific application, a few approaches (Clio, COMA, Cupid, and SF), try to address the schema matching problem in a generic way that is suitable for different applications and schema languages. A taxonomy of automatic match techniques and a comparison of the match approaches followed by the various systems is provided in [20].

[1] The authors did not give a name to their system, so we refer to it in this paper using the initials of the authors' names.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 221–237, 2003.

For identifying a solution for a particular match problem, it is important to understand which of the proposed techniques performs best, i.e., can reduce the manual work required for the match task at hand most effectively. To show the effectiveness of their system, the authors have usually demonstrated its application to some real-world scenarios or conducted a study using a range of schema matching tasks. Unfortunately, the system evaluations were done using diverse methodologies, metrics, and data making it difficult to assess the effectiveness of each single system, not to mention to compare their effectiveness. Furthermore, the systems are usually not publicly available making it virtually impossible to apply them to a common test problem or benchmark in order to obtain a direct quantitative comparison.

To obtain a better overview about the current state of the art in evaluating schema matching approaches, we review the recently published *evaluations* of the schema matching systems in this paper. For this purpose, we introduce and discuss the major criteria influencing the effectiveness of a schema matching approach, e.g., the chosen test problems, the design of the experiments, the metrics used to quantify the match quality and the amount of saved manual effort. We intend our criteria to be useful for future schema matching evaluations so that they can be documented better, their result be more reproducible, and a comparison between different systems and approaches be easier. For our study, we only use the information available from the publications describing the systems and their evaluation.

In Section 2, we present the criteria that we use in our study to contrast the evaluations described in the literature. In Section 3, we review the single evaluations by giving first a short description about the system being evaluated and then discussing the methodology and the result of the actual evaluation. In Section 4, we compare the evaluations by summarizing their strengths and weakness. We then present our observations concerning the current situation of the match systems as well as the challenges that future match implementations and evaluations should address. Section 5 concludes the paper.

2 Comparison Criteria

To compare the evaluations of schema matching approaches we consider criteria from four different areas:

- *Input*: What kind of input data has been used (schema information, data instances, dictionaries etc.)? The simpler the test problems are and the more auxiliary information is used, the more likely the systems can achieve better effectiveness. However, the dependence on auxiliary information may also lead to increased preparation effort.
- *Output*: What information has been included in the match result (mappings between attributes or whole tables, nodes or paths etc.)? What is the correct result? The less information the systems provide as output, the lower the probability of making errors but the higher the post-processing effort may be.
- *Quality measures*: What metrics have been chosen to quantify the accuracy and completeness of the match result? Because the evaluations usually use different

metrics, it is necessary to understand their behavior, i.e. how optimistic or pessimistic their quality estimation is.
- *Effort*: How much savings of manual effort are obtained and how is this quantified? What kind of manual effort has been measured, for example, pre-match effort (training of learners, dictionary preparation etc.), and post-match effort (correction and improvement of the match output)?

In the subsequent sections we elaborate on the above criteria in more detail.

2.1 Input: Test Problems and Auxiliary Information

To document the complexity of the test problems, we consider the following information about the test schemas:
- *Schema language*: Different schema languages (relational, XML schemas, etc.) can exhibit different facets to be exploited by match algorithms. However, relying on language-specific facets will cause the algorithms to be confined to the particular schema type. In current evaluations, we have observed only homogeneous match tasks, i.e. matching between schemas of the same type.
- *Number of schemas and match tasks*: With a high number of different match tasks, it is more likely to achieve a realistic match behavior. Furthermore, the way the match tasks are defined can also influence the problem complexity, e.g. matching independent schemas with each other vs. matching source schemas to a single global schema.
- *Schema information*: Most important is the number of the schema elements for which match candidates are to be determined. The bigger the input schemas are, the greater the search space for match candidates will be, which often leads to lower match quality. Furthermore, matchers exploiting specific facets will perform better and possibly outperform other matchers when such information is present or given in better quality and quantity.
- *Schema similarity*: Intuitively, a match task with schemas of the same size becomes "harder" if the similarity between them drops. Here we refer to schema similarity simply as the ratio between the number of matching elements (identified in the manually constructed match result) and the number of all elements from both input schemas [7].
- *Auxiliary information used*: Examples are dictionaries or thesauri, or the constraints that apply to certain match tasks (e.g., each source element must match at least one target element). Availability of such information can greatly improve the result quality.

2.2 Output: Match Result

The output of a match system is a mapping indicating which elements of the input schemas correspond to each other, i.e. match. To assess and to compare the output quality of different match systems, we need a uniform representation of the correspondences. Currently, all match prototypes determine correspondences between

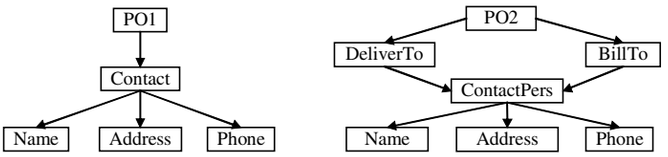


Fig. 1. Schema examples for a simple match task

schema elements (element-level matches [20]) and use similarity values between 0 (strong dissimilarity) and 1 (strong similarity) to indicate the plausibility of the correspondences. However, the quality and quantity of the correspondences in a match result still depend on several orthogonal aspects:

- *Element representation*: Schema matching systems typically use a graph model for the internal representation of schemas. Hence, schema elements may either be represented by nodes or paths in the schema graphs which also impacts the representation of correspondences. Figure 1 shows a simple match problem with two small (purchase order) schemas in directed graph representation; a sample correspondence between nodes would be *Contact*↔*ContactPers*. However, shared elements, such as *ContactPers* in *PO2*, exhibit different contexts, i.e. *DeliverTo* and *BillTo*, which should be considered independently. Thus, some systems return matches between node paths, e.g., *PO1.Contact*↔*PO2.DeliverTo.ContactPers*. Considering paths possibly leads to more elements, for which match candidates can be individually determined, and thus, possibly to more correspondences. Furthermore, the paths implicitly include valuable join information that can be utilized for generating the mapping expressions.
- *Cardinality*: An element from one schema can participate in zero, one or several correspondences (*global cardinality* of 1:1, 1:n/n:1, or n:m). Moreover, within a correspondence one or more elements of the first schema may be matched with one or more elements of the second schema (*local cardinality* of 1:1, 1:n/n:1, n:m) [20]. For example, in Figure 1, *PO1.Contact* may be matched to both *PO2.DeliverTo.ContactPers* and *PO2.BillTo.ContactPers*. Grouping these two match relationships within a single correspondence, we have 1:n local cardinality. Representing them as two separate correspondences leads to 1:n global and 1:1 local cardinality. Most automatic match approaches are restricted to 1:1 local cardinality by selecting for a schema element the most similar one from the other schema as the match candidate.

2.3 Match Quality Measures

To provide a basis for evaluating the quality of automatic match strategies, the match task first has to be manually solved. The obtained real match result can be used as the "gold standard" to assess the quality of the result automatically determined by the

match system. Comparing the automatically derived matches with the real matches results in the sets shown in Figure 2 that can be used to define quality measures for schema matching. In particular, the set of automatically derived correspondences is comprised of B, the *true positives*, and C, the *false positives*. *False negatives* (A) are correspondences needed but not automatically identified, while false positives are correspondences falsely proposed by the automatic match operation. *True negatives*, D, are false correspondences, which have also been correctly discarded by the automatic match operation. Intuitively, both false negatives and false positives reduce the match quality.

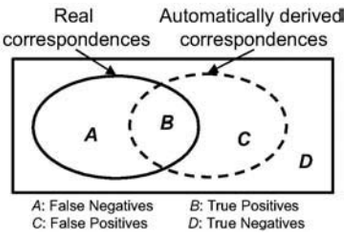


Fig. 2. Comparing real and automatically derived correspondences

Based on the cardinality of these sets, two common measures, *Precision* and *Recall*, which actually originate from the information retrieval field, can be computed:

- $Precision = \frac{|B|}{|B|+|C|}$ reflects the share of real correspondences among all found ones
- $Recall = \frac{|B|}{|A|+|B|}$ specifies the share of real correspondences that is found

In the ideal case, when no false negatives and false positives are returned, we have *Precision*=*Recall*=1. However, neither *Precision* nor *Recall* alone can accurately assess the match quality. In particular, *Recall* can easily be maximized at the expense of a poor *Precision* by returning as many correspondences as possible, e.g. the cross product of two input schemas. On the other side, a high *Precision* can be achieved at the expense of a poor *Recall* by returning only few (correct) correspondences.

Hence it is necessary to consider both measures or a combined measure. Several combined measures have been proposed so far, in particular:

- $F\text{–}Measure(\alpha) = \frac{|B|}{(1-\alpha)*|A|+|B|+\alpha*|C|} = \frac{Precision * Recall}{(1-\alpha)*Precision+\alpha*Recall}$, which also stems from the information retrieval field [21]. The intuition behind this parametrized measure ($0 \le \alpha \le 1$) is to allow different relative importance to be attached to *Precision* and *Recall*. In particular, *F-Measure*(α)$\rightarrow$*Precision*, when $\alpha \rightarrow 1$, i.e. no importance is attached to *Recall*; and *F-Measure*(α)$\rightarrow$*Recall*, when $\alpha \rightarrow 0$, i.e. no importance is attached to *Precision*. When *Precision* and *Recall* are considered equally important, i.e. α=0.5, we have the following combined measure:
- $F\text{–}Measure = \frac{2*|B|}{(|A|+|B|)+(|B|+|C|)} = 2*\frac{Precision*Recall}{Precision+Recall}$, which represents the harmonic mean of *Precision* and *Recall* and is the most common variant of *F-Measure*(α) in information retrieval. Currently, it is used in [3] for estimating match quality.

- $Overall = 1 - \frac{|A| + |C|}{|A| + |B|} = \frac{|B| - |C|}{|A| + |B|} = Recall * \left(2 - \frac{1}{Precision}\right)$, which has been introduced in [15][2] and is also used in [7]. Unlike *F-Measure*(α), *Overall* was developed specifically in the schema matching context and embodies the idea to quantify the post-match effort needed for adding false negatives and removing false positives.

To compare the behavior of *F-Measure* and *Overall*, Figure 3 shows them as functions of *Precision* and *Recall*, respectively. Apparently, *F-Measure* is much more optimistic than *Overall*. For the same *Precision* and *Recall* values, *F-Measure* is still much higher than *Overall*. Unlike the other measures, *Overall* can have negative values, if the number of the false positives exceeds the number of the true positives, i.e. *Precision*<0.5. Both combined measures reach their highest value 1.0 with *Precision*=*Recall*=1.0. In all other cases, while the value of *F-Measure* is within the range determined by *Precision* and *Recall*, *Overall* is smaller than both *Precision* and *Recall*.

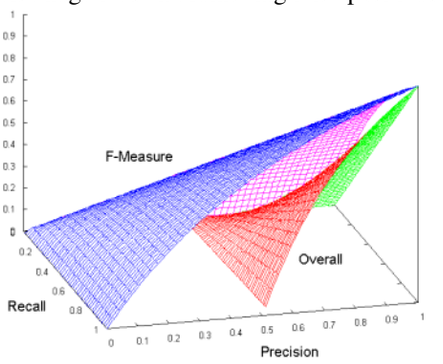


Fig. 3. *F-Measure* and *Overall* as functions of *Precision* and *Recall*

2.4 Test Methodology: What Effort Is Measured and How

Given that the main purpose of automatic schema matching is to reduce the amount of manual work quantifying the user effort still needed is a major requirement. However this is difficult because of many subjective aspects involved and thus a largely unsolved problem. To assess the manual effort one should consider both the *pre-match effort* required before an automatic matcher can run as well as the *post-match effort* to add the false negatives to and to remove the false positives from the final match result.

Pre-match effort includes:

- Training of the machine learning-based matchers
- Configuration of the various parameters of the match algorithms, e.g., setting different threshold and weight values
- Specification of auxiliary information, such as, domain synonyms and constraints

In fact, extensive pre-match effort may wipe out a large fraction of the labor savings obtained through the automatic matcher and therefore needs to be specified precisely. In all evaluations so far the pre-match effort has not been taken into account for determining the quality of a match system or approach.

---

[2] Here it is called *Accuracy*

The simple measures *Recall* and *Precision* only partially consider the post-match effort. In particular, while 1–*Recall* gives an estimate for the effort to add false negatives, 1–*Precision* can be regarded as an estimate for the effort to remove false positives. In contrast, the combined measures *F-Measure*(α) and *Overall* take both kinds of effort into account. *Overall* assumes equal effort to remove false positives and to identify false negatives although the latter may require manual searching in the input schemas. On the other hand, the parameterization of *F-Measure*(α) already allows to apply individual cost weighting schemes. However, determining that a match is correct requires extra work not considered in both *Overall* and *F-Measure*(α).

Unfortunately, the effort associated with such manual pre-match and post-match operations varies heavily with the background knowledge and cognitive abilities of users, their familiarity with tools, the usability of tools (e.g. available GUI features such as zooming, highlighting the most likely matches by thick lines, graying out the unlikely ones etc.) making it difficult to capture the cost in a general way.

Finally, the specification of the real match result depends on the individual user perception about correct and false correspondences as well as on the application context. Hence, the match quality can differ from user to user and from application to application given the same input schemas. This effect can be limited to some extent by consulting different users to obtain multiple subjective real match results [15].

3 Studies

In the following, we review the evaluations of eight different match prototypes, Autoplex, Automatch, COMA, Cupid, LSD, GLUE, SemInt, and SF. We have encountered a number of systems, which either have not been evaluated, such as Clio, DIKE, MOMIS, SKAT, and TranScm, or their evaluations have not been described with sufficient detail, such as Delta, and EJX. Those systems are not considered in our study. For each system, we shortly describe its match approach and then discuss the details of the actual evaluation. According to the taxonomy presented in [20], we briefly characterize the approaches implemented in each system by capturing

- The type of the matchers implemented (schema vs. instance level, element vs. structure level, language vs. constraint based etc.)
- The type of information exploited (e.g., schema properties, instance characteristics, and external information)
- The mechanism to combine the matchers (e.g., hybrid or composite [20, 7]).

3.1 Autoplex and Automatch

System description. Autoplex [2] and its enhancement Automatch [3] represent single-strategy schema matching approaches based on machine learning. In particular, a Naive Bayesian learner exploits instance characteristics to match attributes from a relational source schema to a previously constructed global schema. For each source attribute, both match and mismatch probability with respect to every global attribute are determined. These probabilities are normalized to sum to 1 and the match prob-

ability is returned as the similarity between the source and global attribute. The correspondences are filtered to maximize the sum of their similarity under the condition that no correspondences share a common element. The match result consists of attribute correspondences of 1:1 local and global cardinality.

Evaluation. In both Autoplex and Automatch evaluation, the global schemas were rather small, containing 15 and 9 attributes, respectively. No information about the characteristics of the involved source schemas was given. First the source schemas were matched manually to the global schema, resulting in 21 and 22 mappings in the Autoplex and Automatch evaluation, respectively. These mappings were divided into three portions of approximately equal content. The test was then carried out in three runs, each using two portions for learning and the remaining portion for matching.

The Autoplex evaluation used the quality measures *Precision* and *Recall*,[3] while for Automatch, *F-Measure* was employed. However, the measures were not determined for single experiments but for the entire evaluation: the false/true negatives and positives were counted over all match tasks. For Autoplex, they were reported separately for table and column matches. We re-compute the measures to consider all matches and obtain a *Precision* of 0.84 and *Recall* of 0.82, corresponding to an *F-Measure* of 0.82 and *Overall* of 0.66. Furthermore, the numbers of the false/true negatives and positives were rather small despite counting over multiple tasks, leading to the conclusion that the source schemas must be very small. For Automatch, the impact of different methods for sampling training data on match quality was studied. The highest *F-Measure* reported was 0.72, so that the corresponding *Overall* must be worse.

3.2 COMA

System description. COMA [7] follows a composite approach, which provides an extensible library of different matchers and supports various ways for combining match results. Currently, the matchers exploit schema information, such as element and structural properties. Furthermore, a special matcher is provided to reuse the results from previous match operations. The combination strategies address different aspects of match processing, such as, aggregation of matcher-specific results and match candidate selection. Schemas are transformed to rooted directed acyclic graphs, on which all match algorithms operate. Each schema element is uniquely identified by its complete path from the root of the schema graph to the corresponding node. COMA produces element-level matches of 1:1 local and m:n global cardinality.

Evaluation. The COMA evaluation used 5 XML schemas for purchase orders taken from www.biztalk.org. The size of the schemas ranged from 40 to 145 unique elements, i.e. paths. Ten match tasks were defined, each matching two different schemas. The similarity between the schemas was mostly only around 0.5, showing that the schemas are much different even though they are from the same domain. Some pre-match effort was needed to specify domain synonyms and abbreviations.

---

[3] Here they are called *Soundness* and *Completeness*, respectively

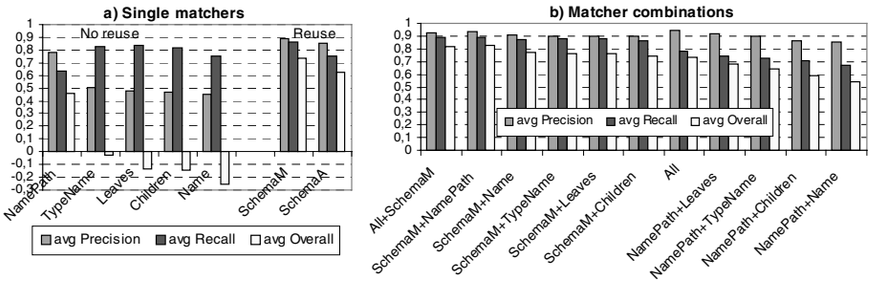


Fig. 4. Match quality of COMA [7]

A comprehensive evaluation was performed with COMA to investigate the impact of different combination strategies on match quality and to compare the effectiveness of different matchers, i.e. single matchers vs. matcher combinations, with and without reuse. The entire evaluation consisted of over 12,000 test series, in each of which a different choice of matchers and combination strategies was applied. Each series in turn consisted of 10 experiments dealing with the (10) predefined match tasks. The quality measures *Precision*, *Recall*, and *Overall* were first determined for single experiments and then averaged over 10 experiments in each series (average *Precision*, etc.). Based on their quality behavior across the series, the best combination strategies were determined for the default match operation.

Figure 4a shows the quality of the single matchers, distinguished between the no-reuse and reuse-oriented ones. The reuse matchers yielded significantly better quality then the no-reuse ones. Figure 4b shows the quality of the best matcher combinations. In general, the combinations achieved much better quality than the single matchers. Furthermore, a superiority of the reuse combinations over the no-reuse ones was again observed. While the best no-reuse matcher, *All*, combining all the single no-reuse matchers, achieved average *Overall* of 0.73 (average *Precision* 0.95, average *Recall* 0.78), the best reuse combination, *All+SchemaM*, reached the best average *Overall* in the entire evaluation, 0.82 (average *Precision* 0.93, average *Recall* 0.89). These combinations also yielded the best quality for most match tasks, i.e. high stability across different match tasks. However, while optimal or close to optimal *Overall* was achieved for the smaller match tasks, *Overall* was limited to about 0.6-0.7 in larger problems. This was apparently also influenced by the moderate degree of schema similarity.

3.3 Cupid

System description. Cupid [14] represents a sophisticated hybrid match approach combining a name matcher with a structural match algorithm, which derives the similarity of elements based on the similarity of their components hereby emphasizing the name and data type similarities present at the finest level of granularity (leaf level). To

address the problem of shared elements, the schema graph is converted to a tree, in which additional nodes are added to resolve the multiple relationships between a shared node and its parent nodes. Cupid returns element-level correspondences of 1:1 local and n:1 global cardinality.

Evaluation. In their evaluation, the authors compared the quality of Cupid with 2 previous systems, DIKE and MOMIS, which had not been evaluated so far. For Cupid, some pre-match effort was needed to specify domain synonyms and abbreviations. First, the systems were tested with some canonical match tasks considering very small schema fragments. Second, the systems were tested with 2 real-world XML schemas for purchase order, which is also the smallest match task in the COMA evaluation [7]. The authors then compared the systems by looking for the correspondences which could or could not be identified by a particular system. Cupid was able to identify all necessary correspondences for this match task, and thus showed a better quality than the other systems. In the entire evaluation, no quality measures were computed.

3.4 LSD and GLUE

System description. LSD [8] and its extension GLUE [9] use a composite approach to combining different matchers. While LSD matches new data sources to a previously determined global schema, GLUE performs matching directly between the data sources. Both use machine-learning techniques for individual matchers and an automatic combination of match results. In addition to a name matcher, they use several instance-level matchers, which discover during the learning phase different characteristic instance patterns and matching rules for single elements of the target schema. The predictions of individual matchers are combined by a so-called meta-learner, which weights the predictions from a matcher according to its accuracy shown during the training phase. The match result consists of element-level correspondences with 1:1 local and n:1 global cardinality.

Evaluation. LSD was tested on 4 domains, in each of which 5 data sources were matched to a manually constructed global schema, resulting in 20 match tasks altogether. To match a particular source, 3 other sources from the same domain were used for training. The source schemas were rather small (14-48 elements), while the largest global schema had 66 attributes. GLUE was evaluated for 3 domains, in each of which two website taxonomies were matched in two different directions, i.e. A→B and B→A. The taxonomies were relatively large, containing up to 300 elements. Both systems rely on pre-match effort on the one side to train the learners, and on the other side, to specify domain synonyms and constraints.

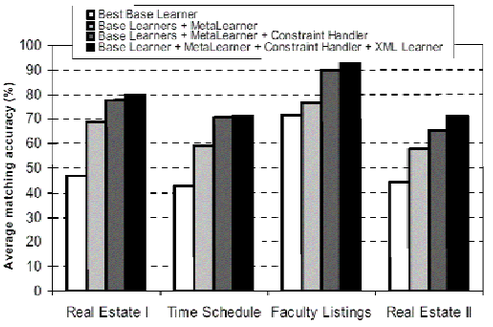


Fig. 5. Match quality of LSD [8]

For both LSD and GLUE, different learner combinations were evaluated. For LSD, the impact of the amount of available instance data on match quality was also studied. Match quality was estimated using a single measure, called *match accuracy*, defined as the percentage of the matchable source attributes that are matched correctly. It corresponds to *Recall* in our definition due to one single correspondence returned for each source element. Furthermore, we observe that at most a *Precision* equal to the presented *Recall* can be achieved for single match tasks; that is, if all source elements are matchable. Based on this conclusion, we can derive the highest possible *F-Measure* (=*Recall*) and *Overall* (=2**Recall*-1) for both LSD and GLUE. Figure 5 shows the quality of different learner combinations in LSD. The best quality was usually achieved when all learners were involved. In the biggest match tasks, LSD and GLUE achieved *Recall* of around 0.7, i.e. *Overall* of at most 0.4. In the case of GLUE, this quality is quite impressive considering the schema sizes involved (333 and 115 elements [9]). On average (over all domains), LSD and GLUE achieved a *Recall* of ~0.8, respectively. This corresponds to an *Overall* of at most 0.6.

3.5 Similarity Flooding (SF)

System description. SF [15] converts schemas (SQL DDL, RDF, XML) into labeled graphs and uses fix-point computation to determine correspondences of 1:1 local and m:n global cardinality between corresponding nodes of the graphs. The algorithm has been employed in a hybrid combination with a simple name matcher, which suggests an initial element-level mapping to be fed to the structural SF matcher. Unlike other schema-based match approaches, SF does not exploit terminological relationships in an external dictionary, but entirely relies on string similarity between element names. In the last step, various filters can be specified to select relevant subsets of match results produced by the structural matcher.

Evaluation. The SF evaluation used 9 match tasks defined from 18 schemas (XML and SQL DDL) taken from different application domains. The schemas were small with the number of elements ranging from 5 to 22, while showing a relatively high similarity to each other (0.75 on average). Seven users were asked to perform the manual match process in order to obtain subjective match results. For each match tasks, the results returned by the system were compared against all subjective results to estimate the automatic match quality, for which the *Overall* measure was used.

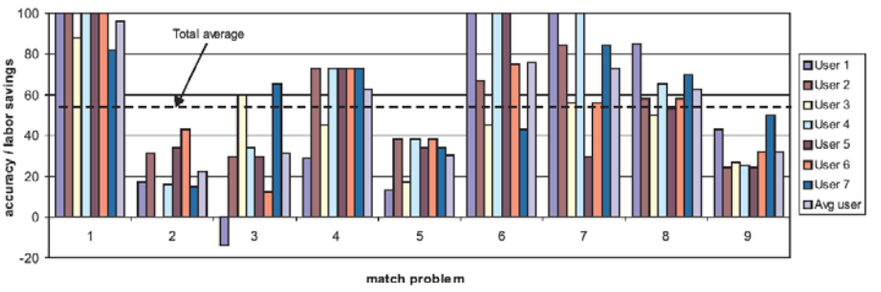


Fig.6. Match quality of Similarity Flooding Algorithm [15•]

Other experiments were also conducted to compare the effectiveness of different filters and formulas for fix-point computation, and to measure the impact of randomizing the similarities in the initial mapping on match accuracy. The best configuration was identified and used in SF. Figure 6 shows the *Overall* values achieved in the single match tasks according to the match results suggested by the single users. The average *Overall* quality over all match tasks and all users is around 0.6.

3.6 SemInt

System description. SemInt [11, 12] represents a hybrid approach exploiting both schema and instance information to identify corresponding attributes between relational schemas. The schema-level constraints, such as data type and key constraints, are derived from the DBMS catalog. Instance data are exploited to obtain further information, such as actual value distributions, numerical averages, etc. For each attribute, SemInt determines a signature consisting of values in the interval [0,1] for all involved matching criteria. The signatures are used first to cluster similar attributes from the first schema and then to find the best matching cluster for attributes from the second schema. The clustering and classification process is performed using neural networks with an automatic training, hereby limiting pre-match effort. The match result consists of clusters of similar attributes from both input schemas, leading to m:n local and n:1 global match cardinality. Figure 7 shows a sample output of SemInt. Note that each cluster may contain multiple 1:1 correspondences, which are not always correct, such as in the first two clusters.

```
(Database1.Faculty.SSN, Database1.Student.Stud_ID, Database2.Personnel.ID,
        similarity = 0.98)
(Database1.Faculty.Facu_Name, Database1.Student.Stud_Name,
        Database2.Personnel.Name, similarity = 0.92)
(Database1.Student.Tel#, Database2.Personnel.W_phone#, similarity = 0.94)
(Database1.Student.Tel#, Database2.Personnel.H_phone#, similarity = 0.95)
```

Fig. 7. SemInt output: match result [12]

Evaluation. A preliminary test consisting of 3 experiments is presented in [11]. The test schemas were small with mostly less than 10 attributes. However, the achieved quality for these experiments was only presented later in [12, 13]. In these small match tasks, SemInt performed very well and achieved very high *Precision* (0.9, 1.0, 1.0) and *Recall* (1.0). In [12, 13], SemInt was evaluated with two further match tasks. In the bigger match task with schemas with up to 260 attributes, SemInt surprisingly performed very well (*Precision* ~0.8, *Recall* ~0.9). But in the smaller task with schemas containing only around 40 elements, the quality dropped drastically (*Precision* 0.20, *Recall* 0.38).

On average over 5 experiments, SemInt achieved a *Precision* of 0.78 and *Recall* of 0.86. Using the *Precision* and *Recall* values presented for each experiment, we can also compute the average *F-Measure*, 0.81, and *Overall*, 0.48. On the other hand, it is necessary to take into consideration that this match quality was determined from match results of attribute clusters, each of which possibly contains multiple 1:1 corre-

spondences. In addition to the match tasks, further tests were performed to measure the sensitivity of the single match criteria employed by SemInt [12]. The results allowed to identify a minimal subset of match criteria, which could still retain the overall effectiveness.

4 Discussion and Conclusions

We first summarize the strengths and weaknesses of the single evaluations and then present our conclusions concerning future evaluations.

4.1 Comparative Discussion

Table 1 gives a summary about the discussed evaluations. The test problems came from very different domains of different complexity. While a few evaluations used simple match tasks with small schemas and few correspondences to be identified (Autoplex, Automatch, SF), the remaining systems also showed high match quality for more complex real-world schemas (COMA, LSD, GLUE, SemInt). Some evaluations, such as Autoplex and Automatch, completely lack the description of their test schemas. The Cupid evaluation represents the only effort so far that managed to evaluate multiple systems on uniform test problems. Unlike other systems, Autoplex, Automatch and LSD perform matching against a previously constructed global schema.

All systems return correspondences at the element level with similarity values in the range of [0,1]. Those confined to instance-level matching, such as Autoplex, Automatch, and SemInt, can only deliver correspondences at the finest level of granularity (attributes). In all systems, except for SemInt, correspondences are of 1:1 local cardinality, providing a common basis for determining match quality.

Only the SF evaluation took into account the subjectivity of the user perception about required match correspondences. Unlike other approaches, SemInt and SF do not require any manual pre-match effort. In several evaluations, e.g. COMA, LSD, GLUE, SemInt and SF, different system configurations were tested by varying match parameters on the same match tasks in order to measure the impact of the parameters on match quality. Those results have provided valuable insights for improving and developing new match algorithms.

Usually, the quality measures were computed for single match experiments. Exceptions are Cupid with no quality measure computed, and Autoplex, Automatch with quality measures mixing the match results of several experiments in a way that does not allow us to assess the quality for individual match tasks. Whenever possible, we tried to translate the quality measures considered in an evaluation to others not considered so that one can get an impression about the actual meaning of the measures. Still, the computed quality measures cannot be used to directly compare the effectiveness of the systems because of the great heterogeneity in other evaluation criteria. Only exploiting schema information, COMA seems quite successful, while the LSD/GLUE approach is promising for utilizing instance data.

Table 1. Summary of the evaluations

	Autoplex & -match	COMA	Cupid	LSD & GLUE	SemInt	SF
References	[2] & [3]	[7]	[14]	[8] & [9]	[11, 12, 13]	[15]
Test problems						
Tested schema types	relational	XML	XML	XML	relational	XML, relational
#Schemas / #Match tasks	15/21 & 15/22	5/10	2/1	24/20 & 3/6	10/5	18/9
Min/Max/Avg schema size	-	40/145/77	40/54//47	14/66/- & 34/333/143	6/260/57	5/22/12
Min/Max/Avg schema similarity	-	0.43/0.8/0.58	-	-	-	0.46/0.94/0.75
Match result representation						
Matches	element-level correspondences with similarity value in range [0,1]					
Element repr.	node (attr.)	path	path	node	node (attr.)	node
Local/global cardinality	1:1/1:1	1:1/m:n	1:1/n:1	1:1/n:1	m:n/m:n (attr. cluster)	1:1/m:n
Quality measures and test methodology						
Employed quality measures	Precision, Recall & F-Measure	Precision, Recall, Overall	none	Recall	Precision, Recall	Overall
Subjectivity	1 user					7 users
Pre-match effort	training	specifying domain synonyms	specifying domain synonyms	training, specifying domain synonyms, constraints	none	none
Studied impact on match quality	Automatch: methods for sampling instance data	matchers, combination, reuse, schema characteristics	none	learner combinations; LSD: amount of data listings	constraints (discriminators)	filters, fix-point formulas, randomizing initial sim
Best average match quality						
Prec./Recall	0.84/0.82	0.93/0.89	-	~0.8/0.8	0.78/0.86	-
F-Measure	0.82 & 0.72	0.90	-	~0.8	0.81	-
Overall	0.66	0.82	-	~0.6	0.48	~0.6
Evaluation highlights						
		Big schemas, Systematic evaluation	Comparative evaluation of 3 systems	Big schemas	Big schemas, No pre-match effort	User subjectivity, No pre-match effort

4.2 Conclusions

The evaluations have been conducted in so different ways that it is impossible to directly compare their results. While the considered match problems were mostly simple, many techniques have proved to be quite powerful such as exploiting element and structure properties (Cupid, SF, COMA), and utilizing instance data, e.g., by Bayesian and Whirl learners (LSD/GLUE) or neural networks (SemInt). Moreover, the combined use of several approaches within composite match systems proved to be very successful (COMA, LSD/GLUE). On the other side, there are still unexploited opportunities, e.g. in the use of large-scale dictionaries and standard taxonomies and increased reuse of previous match results (COMA). Future match systems should integrate those techniques within a composite framework to achieve maximal flexibility.

Future evaluations should address the following issues:

- *Better conception for reproducibility*: To allow an objective interpretation and easy comparison of match quality between different systems and approaches, future

evaluations should be conceived and documented more carefully, if possible, including the criteria that we identified in this paper.

- *Input factors – test schemas and system parameters*: All evaluations have shown that match quality degrades with bigger schemas. Hence, future systems should be evaluated with schemas of more realistic size, e.g. several hundreds of elements. Besides the characteristics of the test schemas, the various input parameters of each system can also influence the match quality in different ways. However, their impact has rarely been investigated in a comprehensive way, thus potentially missing opportunities for improvement and tuning. Similarly, previous evaluations typically reported only some peak values w.r.t. some quality measure so that the overall match quality for a wider range of configurations remained open.
- *Output factors – match results and quality measures*: Instead of determining only one match candidate per schema element, future systems could suggest multiple, i.e. *top-K*, match candidates for each schema element. This can make it easier for the user to determine the final match result in cases where the first candidate is not correct. In this sense, a top-K match prediction may already be counted as correct if the required match candidate is among the proposed choices.
  Previous studies used a variety of different quality measures with limited expressiveness thus preventing a qualitative comparison between systems. To improve the situation and to consider precision, recall and the degree of post-match effort we recommend the use of combined measures such as *Overall* in future evaluations. However, further user studies are required to quantify the different effort needed for finding missing matches, removing false positives, and verifying the correct results. Another limitation of current quality measures is that they do not consider the pre-match effort and the hardness of match problems.

Ultimately, a *schema matching benchmark* seems very helpful to better compare the effectiveness of different match systems by clearly defining all input and output factors for a uniform evaluation. In addition to the test schemas, the benchmark should also specify the use of all auxiliary information in a precise way since otherwise any hard-to-detect correspondences could be built into a synonym table to facilitate matching. Because of the extreme degree of heterogeneity of real-world applications, the benchmark should not strive for general applicability but focus on a specific application domain, e.g., a certain type of E-business. Alternatively, a benchmark can focus on determining the effectiveness of match systems with respect to specific match capabilities, such as name, structural, instance-based and reuse-oriented matching. Currently we are investigating how such benchmarks could be generated.

5 Summary

Schema matching is a basic problem in many database and data integration applications. We observe a substantial research and development effort in order to provide semi-automatic solutions aiding the user in this time-consuming task. So far, many systems have been developed and several of them evaluated to show their effectiveness. However, the way the systems have been tested varies to a great extent from

evaluation to evaluation. Thus it is difficult to interpret and compare the match quality presented for each system.

We proposed a catalog of criteria for documenting the evaluations of schema matching systems. In particular, we discussed various aspects that contribute to the match quality obtained as the result of an evaluation. We then used our criteria and the information available in the literature to review several previous evaluations. Based on the observed strengths and weaknesses, we discussed the problems that future system implementations and evaluations should address. We hope that our criteria provide a useful framework for conducting and describing future evaluations.

Acknowledgements. We thank Phil Bernstein and AnHai Doan for many useful comments. This work is supported by DFG grant BIZ 6/1-1.

References

1. Andritsos, P. et al: Schema Management. IEEE Bulletin on Data Engineering 25: 3, 2002
2. Berlin, J., A. Motro: Autoplex: Automated Discovery of Content for Virtual Databases. CoopIS 2001
3. Berlin, J., A. Motro: Database Schema Matching Using Machine Learning with Feature Selection. CAiSE 2002
4. Bergamaschi, S., S. Castano, M. Vincini, D. Beneventano: Semantic integration of heterogeneous information sources. Data & Knowledge Engineering 36: 3, 2001
5. Castano, S., V. De Antonellis: A Schema Analysis and Reconciliation Tool Environment. IDEAS 1999
6. Clifton, C., E. Housman, A. Rosenthal: Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases. IFIP 2.6 Working Conf. Database Semantics 1996
7. Do, H.H., E. Rahm: COMA – A System for Flexible Combination of Schema Matching Approach. VLDB 2002
8. Doan, A.H., P. Domingos, A. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. SIGMOD 2001
9. Doan, A.H., J. Madhavan, P. Domingos, A. Halevy: Learning to Map between Ontologies on the Semantic Web. WWW 2002
10. Embley, D.W., D. Jackman, L. Xu: Multifaceted Exploitation of Metadata for Attribute Match Discovery in Information Integration. WIIW 2001
11. Li, W.S., C. Clifton: Semantic Integration in Heterogeneous Databases Using Neural Networks. VLDB 1994
12. Li, W.S., C. Clifton: SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Network. Data and Knowledge Engineering 33: 1, 2000
13. Li, W.S., C. Clifton, S.Y. Liu: Database Integration Using Neural Networks: Implementation and Experiences. Knowledge and Information Systems 2: 1, 2000
14. Madhavan, J., P.A. Bernstein, E. Rahm: Generic Schema Matching with Cupid. VLDB 2001
15. Melnik, S., H. Garcia-Molina, E. Rahm: Similarity Flooding: A Versatile Graph Matching Algo-rithm. ICDE 2002
16. Milo, T., S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB 1998, 122–133

17. Mitra, P., G. Wiederhold, J. Jannink: Semi-automatic Integration of Knowledge Sources. Fusion 1999
18. Naumann, F., C.T. Ho, X. Tian, L. Haas, N. Megiddo: Attribute Classification Using Feature Analysis. ICDE 2002 (Poster)
19. Palopoli, L., G. Terracina, D. Ursino: The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. ADBIS-DASFAA 2000
20. Rahm, E., P.A. Bernstein: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10: 4, 2001
21. Van Rijsbergen, C.J.: Information Retrieval. 2nd edition, 1979, London, Butterworths.

Indexing Open Schemas

Neal Sample[1,2] and Moshe Shadmon[2]

[1]Department of Computer Science,
Stanford University, Stanford, CA 94305, USA
[2]RightOrder Incorporated
3850 N. First St., San Jose, CA 95134, USA
nsample@stanford.edu, moshes@rightorder.com

Abstract. Significant work has been done towards achieving the goal of placing semistructured data on an equal footing with relational data. While much attention has been paid to performance issues, far less work has been done to address one of the fundamental issues of semistructured data: schema evolution. Semistructured indexing and storage solutions tend to end where schema evolution begins. In practice, a real promise of semistructured data management will be realized where schemas evolve and change. In contrast to fixed schemas, we refer to schemas that grow and change as *open schemas*. This paper addresses the central complications associated with indexing open and evolving schemas: we specify the features and functionality that should be supported in order to handle evolving semistructured data. Specific contributions include a map of the steps for handling open schemas and an index for open schemas.

1 Introduction

Storing and managing semistructured data has received attention in both industrial and academic circles. Significant work has been done to achieve the goal of placing XML data (among other semistructured formats) on an equal footing with relational data. Some research projects and commercial products have the explicit objective of bringing the expanding body of semistructured data inside relational databases to achieve the goal [1,2,3], while others have attempted to build high-performance "native" semistructured databases and indexes [4,5,7].

While much attention has been paid to performance issues, far less work has been done to address one of the fundamental issues of semistructured data: *schema evolution*. In relational databases, the schema represents the logical structure of data elements. It is typically an expensive operation to alter schemas, and may have consequences that bubble up from the relational store to the application layer. Semistructured indexing and storage solutions tend to end where schema evolution begins. Data stores and indexes are built assuming that the data may enter and leave the system as some form of XML, but that the XML's schema will be somewhat static, most likely corresponding to a fairly rigid DTD. This is an overly limiting assumption.

In practice, the real promise of semistructured data management will be realized when schemas evolve and change. In contrast to fixed schemas, we refer to schemas that grow and change as *open schemas*. For example, in the most well defined industries there are still competing standards and representations for information. Even when there are standard record formats, records can be incomplete due to

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 238–252, 2003.

missing or incorrectly entered information. However, these semi-rigid and incomplete data sets are actually the easiest available sources to integrate. In practice, data records and content objects vary widely among entities in both scope and format. This type of data provides a challenge for traditional databases to store and search, and the complications grow as actors enter and leave. While DTDs and DDML/XSchema can accurately *describe* evolving semistructured data sets, they say nothing about effectively storing and indexing such data [14].

This paper addresses the central complications associated with indexing open and evolving schemas. Our objective is twofold, both theoretical and practical. We specify the features and functionality that should be supported in order to handle evolving semistructured data. Some features are well understood because they are required with or without an open schema, so we will just give a cursory examination of those details. Our specific contributions:

1. Map out steps for handling open schemas – we cover the fundamental requirements for managing data with an evolving schema. These steps are generally applicable and may be interpreted broadly for any semistructured data format.
2. An index for open schemas – we illustrate the Index Fabric, an index well suited for evolving schemas. The Index Fabric is compact, high-performance, and balanced over semistructured and irregular data sources, making it ideal for schemas that change over time.

This rest of this paper consists of three main sections. First, we explain the steps for indexing an open schema, from key generation to query support. Second, we present the Index Fabric as a substrate for supporting indexes over data with an open schema. Finally, we summarize our contributions and close with a few remarks about open issues.

2 Managing Open Schemas

In this section we cover the steps required to index data with an open schema. These steps cover each level of the process, from generating keys to changing data to updating indexes after source data changes.

2.1 Key Encoding

Interesting data items are located by searching an index for matching keys. Keys consist of attribute names and data values for corresponding attributes. In a relational database, attributes are the same as columns in a table. With semistructured data, there is no assumption that there is a regular or complete mapping to a relational table. If there is such an obvious mapping, either to a relational table or to a fully specified and static DTD, then indexing is not difficult. However, an evolving schema places new requirements on the indexing methodology. In order to build the index, there must be an association between attribute names (e.g., XML tags) and data values (e.g., XML PCDATA). We propose to map attribute names to *designators,* and to keep information about designators in a dictionary. This designator dictionary maps designators (used in an index) to and from arbitrary attribute names [5].

Keys are an encoding of designators and data values derived from objects in the database.

2.1.1 Designator Creation

Any semistructured indexing strategy must support dynamic creation of new designators. As new attributes names (e.g., previously unseen XML tags) enter the system they should be integrated into the existing framework with a minimum of effort. This means that an in-flight index with an existing set of designators should support the definition of new designators, and the insertion of keys tagged with those designators should not require a complete index rebuild. Creating new designators is the first requirement when supporting evolving schemas.

2.1.2 Designator Dictionary

There must be an explicit mechanism to map between the self-describing labels (e.g., tags) in the data objects and designators representing those labels. This mechanism, the *designator dictionary*, must support the following functions:

- *Semantic marking of designators*: when the system encounters two objects from different domains and has knowledge of the distinction, it should create distinct designators for syntactically similar but semantically different attributes (e.g. as attribute names (label, tag, etc.) versus designators, as from one domain versus another). For example, "<record>" in one domain may refer to a collection of data, while "<record>" in another domain refers to an album of music.
- *Lookup of an existing attribute name*: to find the corresponding designator(s) and associated semantic markings. If the attribute name does not exist, an error condition should be returned.
- *Creation of new designators*: for new attribute names. This function should support marking the new designators with semantic information.
- *Lookup of an existing designator*: to find the corresponding attribute name and associated semantic markings.
- *Pattern matching over attribute names*: ideally, the dictionary would support regular expression lookup to find attribute names.

The designator dictionary is the bridge between the external view of data and any internal representations used for indexing/storage/etc. It can also play a crucial role in relating semantically equivalent concepts and searching over data types that do not adhere to identical schemas. For instance, augmenting the designator dictionary with semantic information can assist when searching for attributes that do not have a common name, but do have a common meaning. The designator dictionary overcomes one of the limitations of existing databases, namely the fixed name space. With the designator dictionary identical and different names are mapped to designators simplifying schema evolution and avoiding name conflicts when data is integrated.

2.1.3 Encoding Keys

Data elements in a semistructured data document may be of any length, and attribute names are fundamentally unlimited as well. Ideally, a key of any length should be supportable. With keys over semistructured data, it is needed that the keys should allow search and still reflect hierarchy of any depth. Obviously, the index implementation of these long keys that embed relationships should not impact performance. Moreover, keys may include multiple designators with no corresponding data. For example, the path "***invoice.buyer.name.***'Cisco'" could be represented as "**I.B.N.**Cisco" where "**I**," "**B**," and "**N**" are designators. At the same

time a key such as "**I**.325.**B.N**.Cisco" allows search for the buyer name of invoice number "325." These are examples of raw and refined paths explained in more detail in [5].

2.2 Data Interface

There should be an abstract "pointer" type, which could be a unique document identifier, a pair (docid, offset), a physical row identifier, or any type of pointer appropriate for a particular data storage manager. If the index is designed generically to handle arbitrary pointer types, then any data storage mechanism can be used. Separating the indexing mechanism from the storage mechanism is important for generality, at the cost of potential performance. (However, performance results presented in [5] indicate that external indexing can be quite effective compared to less-flexible native indexes.)

This requirement implies a modular separation between any index and the data manager. In other words, different data storage managers can be used without modifying the index. This means that schemas may evolve, and that even new storage managers may be introduced without altering existing stores. The index itself is agnostic to storage and treats keys generated from different data stores uniformly. This level of indirection is critical for complete flexibility, but would likely be somewhat limited in installations where performance is favored at the cost of fixing the underlying storage manager. The features supported by the data layer should be:

- Insert document; returns a document ID (pointer)
- Accept a pointer from the index and return the corresponding document or fragment

2.3 Inserts, Updates, and Deletions

We have omitted the following sections from this shortened version: 2.3 Inserts, updates, and deletions, 2.3.1 Parser, 2.3.2 Insertion, 2.3.3 Deletion, and 2.3.4 Updates. These sections may be found in the complete paper [15].

2.4 Query Support

Querying an open schema can be problematic for certain applications that make assumptions about underlying data. For instance, if query expects to discover data with the form ***A.B.C***, but the schema has evolved to include data with ***A.B.C.D***, how should that be handled? An obvious solution is to push filtering logic into the client application such that it accepts any ***A.B.C.****, but discards anything uninteresting. This type of filtering has been touted as an advantage of using semistructured data, but is not always so clearly desirable in practice. For performance reasons, it may be much better to return just the portion ***A.B.C***. Furthermore, if clients are brittle and validate returned values against a restricted DTD, previously legitimate results may be rejected. This indicates that query support should include filter operations at the server and mechanisms that avoid retrieving irrelevant data. This may be standing filters for specific clients, XSLT, or some other mechanism.

2.4.1 Query Language

An index should support any front-end semistructured query language, as even query languages are in a state of flux and evolution. This means there should be a well-defined internal language for query operations. Query compilers specific to front-end languages will translate queries into internal language queries. A key step will be the translation of attribute names (e.g. tags) to designators.

2.4.2 Query Operators

There are basic operators (which provide simple index lookups) and more complicated operators (to provide joins, unions, etc.). We begin by defining a complete but simple set of operators (e.g., able to answer queries but not necessarily in the most efficient way) and build on that set as necessary.

Simple Operators

- *Single key lookup*: given a designator encoded key, find the pointer associated with that key.
- *Prefix key lookup*: given a prefix of a designator-encoded key, find the pointers associated with all keys that have that prefix.
- *Data retrieval*: given a pointer, retrieve the data (e.g., document or fragment) associated with that pointer.

Complex Operators

- *Set union*: given two sets of pointers, return the set union.
- *Set intersection*: given two sets of pointers, return the set intersection.
- *Set difference*: given two sets of pointers, return the set difference.
- *Projection*: given a document, extract the desired information. (As mentioned above, this can use a standard tool, like XSLT.)
- *Join*: given two sets of pointers, join the associated data according to a specific condition.
- *Select*: given a set of pointers, filter out those that do not meet a specific condition.
- *Discover structure*: retrieve unknown structure subordinated to a given path.

2.4.3 Query Optimization

There are many different ways to use an index to answer even simple queries. For example, a search for a document with a buyer "X" (e.g. <buyer>X</buyer>) and seller "Y" (e.g. <seller>Y</seller>) can be supported in several ways, including:

- Lookup documents with buyer "X." Lookup documents with seller "Y." Take the intersection.
- Lookup documents with buyer "X." Select for documents with seller "Y."
- Lookup documents with seller "Y." Select for documents with buyer "X."

A query optimizer can select an efficient plan for evaluating queries. An optimizer is especially important for searches over root-to-leaf paths, since naïve plans for complex queries (e.g., queries with wildcards) can be very bad. The optimizer may take into account statistics about the data. Statistics about attribute name

distribution/structure can be especially important for optimizing queries over root-to-leaf paths.

However, query optimization is a complex problem and still an active research problem in the area of semistructured data. As a result, it is desirable to begin with a query planner that produces good query plans given simple indexing. While investments in query planning can occur in the engine, the indexing strategy outlined so far makes it easy to directly leverage information about expected queries. For instance, the query for "documents with a buyer 'X' and seller 'Y'" may be easily handled with a single key lookup if the parser knows to generate ***buyer.seller*** composite keys whenever a document has both components.

3 Indexing Open Schemas

Proper indexing is an important component of any database; indexes may access particular elements quickly. There are many trade-offs to consider when choosing which indexes to build. With relational data sources, indexing decisions are made based on expected queries, index update costs, space requirements, and expected benefit, among many other criteria.

When dealing with open schemas, making the right decisions about indexing is more difficult. In this section, we present the Index Fabric, a novel indexing structure suited to both semistructured and hierarchical data. The Index Fabric is also uniquely suited to open schemas. First, the index does not depend on a schema or DTD, and second because it explicitly maintains all element relationships and does so at a very low cost. This means that queries with path-expressions, a crucial component of semistructured queries, are supported in the Index Fabric.

3.1 Piecewise Indexing

We have omitted the following sections from this shortened version: 3.1 Piecewise indexing, 3.1.1 Breaking apart the document, and 3.1.2 Static edge map performance. These sections may be found in the complete paper [15].

3.2 Complete Indexing – Index Fabric

Section 3.1.2 puts forward the argument that the Index Fabric is a fast index for semistructured data, and that it was shown to be faster than one of the best recent hybrid relational approaches. We conclude section 3 with a discussion about why the Index Fabric is also uniquely suited for use with evolving schemas.

3.2.1 Complete Tree Indexing

At the heart of the Index Fabric is a Patricia trie string index [5, 7]. Other string-based indexes have been proposed using Patricias, including Sybase's Compact B-tree Index in the iAnywhere product [8] and the String B-tree [9]. These alternative indexes are built like B-trees with very long strings, but the B-tree blocks are individually compressed using Patricia structures. This is a subtle yet fundamental difference between them and the Index Fabric. Because the Index Fabric maintains a full Patricia over the entire key set, it can be used like Lore's DataGuide [10]. Vertical tree

traversals from the root of the Index Fabric over the tree representing the data in the index are possible in an efficient manner. In the String B-tree or iAnywhere B-tree, such a traversal is not efficient in many cases. A fuller description of the Index Fabric can be found in [5,7,11].

This fully connected vertical Patricia is important for constraining search when the structure of the search results is not known at search time. For example, in hybrid edge map approach like STORED, finding matches over even a simple XPath query like `/catalog/cd/*` could be very expensive if the schema has changed. Imagine every document looked something like document in Figure 1, when the STORED schema was mined.

```
<catalog>
        <cd country="USA">
                <title>Empire Burlesque</title>
                <artist>Bob Dylan</artist>
                <price>10.90</price>
        </cd>
        <cd country="UK">
                <title>Hide your heart</title>
                <artist>Bonnie Tyler</artist>
                <price>9.90</price>
        </cd>
        <cd country="USA">
                <title>Greatest Hits</title>
                <artist>Dolly Parton</artist>
                <price>9.90</price>
        </cd>
</catalog>
```

Fig. 1. Sample XML document

When the schema changes because even one of the `<cd>` tags adds a `<saleprice />`, the overflow edge map table kicks in for every query like `/catalog/cd/*`. The complete Patricia at the core of the Index Fabric prevents the need for an overflow edge map, and also explicitly indicates what tags can reside along any given path during a single lookup, without any joins. This property can be leveraged for inexpensive tests for existence of long paths as well. Looking for something like `/catalog/cd[saleprice<10.0]` can be very expensive when few `<cd>`s in the catalog have that property, but it cannot be determined until each join is performed and the overflow edge map is consulted for every `<cd>`. A visual example of this property can be seen in Figure 2(a) and (b).

In Figure 2 (a), we see the original document structure represented in the Index Fabric's Patricia, before the addition of any `<saleprice>` tags. In Figure 2 (b), we see the new document structure represented in the Index Fabric's Patricia, after the addition of some `<saleprice>` tag. With keys encoded in the Index Fabric, XPath queries for `/catalog/cd[saleprice<10.0]` are immediately directed along the appropriate path in Figure 2(b), as soon as such keys enter the index. With an edge map or hybrid approach, a significant portion of the document base needs to be checked before results are returned. (We should note that researchers have

implemented better search techniques than edge maps in particular instances. Lore's DataGuide can be searched top-down, bottom-up, or in a hybrid fashion to reduce the expense of consulting the index [10]. However, such techniques are not typically available in the logic of commercial systems.)

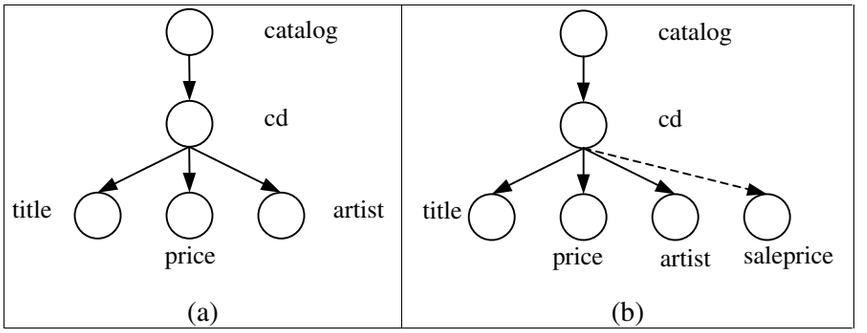


Fig. 2. XML structural summary

3.2.2 Index Fabric Properties

The index Fabric is a layered vertically in the same manner that a B-tree is layered vertically. While a detailed description is beyond the scope of this paper, this horizontal balancing process makes it possible to access either "shallow" or "deep" paths with equal effort. There is a single I/O per horizontal layer in the Index Fabric just as there is in a B-tree tree.The Index Fabric is significantly smaller than a corresponding B-tree because the Patricia at the core is a lossy string index. The high-compression of the Patricia means that the overall size of the index is low, but also means that there is a high fan-out in upper layer blocks, ensuring that the index remains shallow. (In the Index Fabric, because of the lossy Patricia, there is a constant cost per key, regardless of key length. This means that the Index Fabric is not sensitive to the *size* of the keys, only the *number* of keys.) These properties combine to yield the performance advantages presented in [5]. More recently, we have demonstrated near linear speedups for the same queries when parallelizing and distributing portions of the index [11].

There is always the question of performance when indexing hierarchical data. With semistructured and hierarchical data, root-to-leaf paths tend to be long, raising difficulties for complete path keys using standard indexing. The use of designators in the index allows the unification of paths that share the same prefix, even if the indexed data is of different types and stored differently. For example, a query for particular **attributes** of a product in a catalog where the indexed data is ***catalog.product.attribute.value*** allows a query to follow the ***catalog.product*** path and navigate to the needed attributes according to the designators found at the end of the ***catalog.product*** path. If a desired attribute exists (or is missing) for the particular path, this information is readily available.

The Index Fabric is a self-describing index. This means that properties of the indexed elements are stored per element rather than in an external schema. This makes the description of the data always complete regardless of the diversity between the data elements and obviates the need to update a schema when new elements are

introduced. The indexed data reflects the schema, allowing the schema to evolves with the insertion of new types of keys. This feature, along with the fact that the index is able to treat all keys in a uniform manner, with the ability to create a balanced and small index, makes the Index Fabric perfectly suited to index data where schemas are not fixed.

3.2.3 Index Fabric, a Complete Indexing System

In order for the Index Fabric to leverage its performance advantages in the face of open schemas, we show that it meets the requirements of the indexing scheme set forth in section 2. First, the Index Fabric should be compatible with an open process for key encoding, including the creation of new designators for new attribute names entering the system. Second, the Index Fabric should support a generic data interface, not restricted to a single table, source, or set of documents. Third, update operations should be handled gracefully, without requiring significant downtime or index rebuilding as schemas change. Finally, the Index Fabric should support many different query patterns, including those mentioned in sections 2.4.2 and 2.4.3.

Key Encoding

Adding new keys to the Index Fabric is a straightforward process derived from the one outlined in section 2.1. The Index Fabric is a string index that encodes arbitrary bit-strings. These strings can be any length, and the index is content agnostic. Because there are no formatting or content restrictions placed on indexed string, new designators may be added at any point to represent new attribute names. The Index Fabric maps all keys to designated strings regardless of their storage data types. Thus, the data can be very different but paths assigned with the same designators are comparable as if the data was normalized to fit a single schema. In addition, the bit-strings are uniform allowing comparison of keys by their value rather than simply by type. This is in stark contrast to a fully structured environment, like a relational system, where the data needs to be normalized and the addition of new attributes means the creation of new indexes and often implies altering tables.

Data Interface

The Index Fabric manages long, arbitrary keys. In the index, there is some portion of the key materialized by the Patricia structure coupled with an arbitrary pointer object. This pointer may be of any size, and may eventually refer to any object. This works in an obvious way through indirection. B-trees in relational systems frequently have (*value*, *pointer*) pairs in the leaves, where a pointer is something specific to that implementation, like a “row ID.” After searching a B-tree for some values, a set of homogenous pointers is returned. The Index Fabric also returns a set of pointers, however the pointers themselves may be of multiple types. Recall that keys appended with pointers are inserted into the Index Fabric, without restriction on the pointer’s content or structure. The translation of arbitrary pointers to data happens outside the index.

Handling Updates

Updates to the Index Fabric are based directly on output from a document parser. When new documents are added to a database, they pass through a parser that generates the initial set of keys. The initial set of keys is inserted into the index along with a pointer to the original document or appropriate fragment. An update to the

index may be triggered by an update to a document, and is modeled as two operations: a delete followed by an insert. Deletions are not value-based; they require a key string and a pointer. This is standard requirement of any non-unique index.

Query Support
Section 2.4 outlines a set of functional requirements to support a structured query language. The Index Fabric efficiently supports each of these operations. The simple operators are *single key lookup, prefix key lookup,* and *data retrieval*. As noted previously, single key lookup is done in one probe of the index, regardless of key complexity. Prefix key lookup is handled in a similar way. A single horizontal probe into the index locates the specified portion (the prefix) of the query in the Patricia. From this point, a vertical traversal collects all subordinate elements. Referring to Figure 2 (b), a query for `/catalog/cd` would then return a set of four key types, {***catalog.cd.title***, ***catalog.cd.price, catalog.cd.artist***, ***catalog.cd.saleprice***}. The Index Fabric directly supports single and prefix key lookups directly.

Complex operations should be supported by the index as well. Some operations are the direct responsibility of the index (like set operations), while others are used in result post-processing (such as projection). Compared to other indexing alternatives, set operations can be somewhat expensive to perform in the Index Fabric because the Index Fabric's allowance for multiple pointer types. For example, set intersection in a B-tree is a simple process. It can be done by taking two sets of pointers, sorting each, then scanning the pointer sets in order to find the elements that exist in each set. This process is simplified in B-trees by the assumption that all pointers are of the same type (and size).

In the Index Fabric, pointers may be different types (and potentially sizes) to support federated indexes over multiple sources. One solution to the set intersection problem is to partition the sets of pointers by types, essentially sorting by "pointer type" first, and then sorting those partitions by value. This partitioned solution is a reasonable way to perform operations over sets of multiple pointer types.

Optimizing Queries Using the Index Fabric
It has already been shown that the Index Fabric is a fast and small index for semistructured data [5]. We further present that the Index Fabric is appropriate for specialty optimizations. In section 2.4.3, we presented the case of generating special key strings that aid in evaluating specific queries. The example presented was the query for "documents with a buyer 'X' and seller 'Y'." By generating new composite keys of the form ***buyer.seller*** whenever a document has both components, the search can be optimized. The Index Fabric is not limited to any particular key type or structure.

The real insight to adding this support comes from the designator dictionary. It may not be obvious as to why arbitrarily adding a key like ***buyer.seller*** is problematic. But what happens when a user enters an XPath query like `/buyer/seller`? Does this path correspond to a real data fragment, or does it exist in the index simply as an optimized key? The designator dictionary makes it possible to distinguish between keys that represent actual document structure and those built for the purpose of optimized access. The optimized ***buyer.seller*** can be represented with ***buyer'.seller'***, distinguished from an actual path, but appropriate for the optimized query.

3.3 Open Schema Performance

When the schemas change, the utility of the *a priori* analysis done with an approach like STORED is diminished. The STORED algorithm chooses materializations based on high support within the training data set, which can change radically with schema evolution. Where the schema is a moving target, we are left with edge map approaches to storage and indexing. The authors of [13] point out that conventional horizontal row representations fail when schemas constantly evolve. They advance the alternative approach of a vertical representation where data is stored as tuples consisting of object identifier and an attribute/value pair.

Figure 3 shows a horizontal table and its corresponding representation in the vertical format as explained in [13]. (The symbol ∅ represents a null value.)

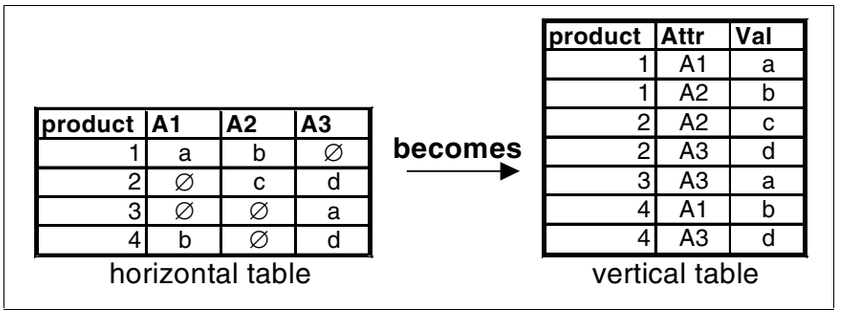


Fig. 3. Horizontal and vertical table representations

With the vertical approach to data storage, the schema is completely open even in a relational store – one simply adds new tuples corresponding to new attributes. According to [13], the best approach to indexing of the vertical table is to build B-tree indexes over each of the three columns.

We compared query times of the vertical table approach for handling open schemas with the Index Fabric. Our approach to storage for the Index Fabric was similar to [13], with one alteration; the data was stored in 3 tables shown in Figure 4: An attribute name table, an attribute value table, and a product ID table. The Index Fabric engine was embedded transparently within a commercial relational DBMS.

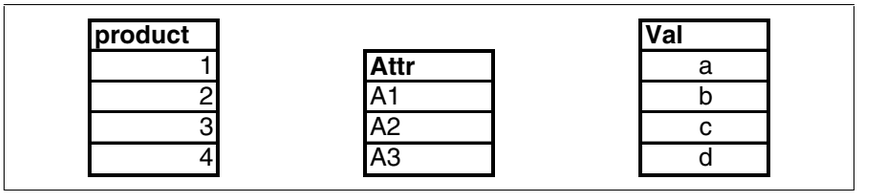


Fig. 4. Index Fabric approach - the data tables

The keys in the Index Fabric are the concatenation of 3 fields: ***Attr.Val.product*** (these keys are shown in the "key string" table embedded in Figure 5). This storage

and indexing structure facilitates schema flexibility; one simply adds new attributes, new values and new products, while maintaining the relationships between the products and the attribute/value pairs through the index.

As noted in earlier publications, the leaf layer of the Index Fabric has a Patricia structure over the data keys [5, 7, 11]. The key strings are not materialized explicitly, rather the index points to the data from which the keys are constructed. Every path in the index has 3 pointers: (a) at the attribute name position in the trie there is a link to the attribute name column in the attribute name table, (b) at the attribute value position in the trie there is a link to the attribute value column in the attribute value table, and (c) at the product ID position in the trie there is a link to the product column in the product table.

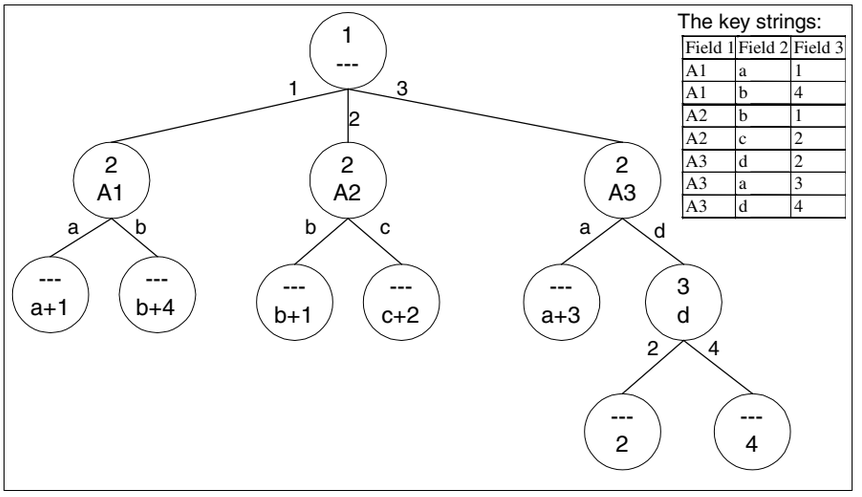


Fig. 5. The Index Fabric structure

Figure 5 shows the Patricia structure over this sample set of keys (these keys come from the data table presented in Figure 3). The circles in Figure 5 represent elements in the index. The links between the circles represent the labels of the Patricia links. Each circle contains two values. The top value stands for a position in a key string. The bottom value stands for a link to a data element; rather than show the pointers here, we show the key value of the data element. Two data links are represented by two key values. If the top value is a dashed line, the circled element represents a link to data (but is not a node). If the bottom value is a dashed line, the circled element represents a node without a link to the data. For example, the leftmost circles of the tree include the root node having the value 1 and no data link, a node with a value 2 and a link to a data record whose key value is "A1" (the first attribute name in the attribute name table). The third left-most circle represents two links to data elements: a link to a data element whose key is "a" (the first value in the attribute value table) and a link to the data element whose key is "1" (the first product in the product table).

Searching in the trie is simple: to search for the key ***A2.b.1***, we follow the root node having the value "1", the link labeled "2", the node having the value "2". We also follow a link to "A2" in the attribute name table, the link labeled "b" to the value

"b" in the attribute value table, and the product "1" in the product table. A query to retrieve product IDs by attribute name "A3" and attribute value "d" would follow the path ***A3.d.**** to the sub-trie rooted at the node with the value "3".

Our experiments run on a 700 MHz dual processor Intel Pentium III machine with 2GB physical memory. The operating system was RH Linux 6.2. The installed database is one of the most widely used commercial databases[1]. For the vertical partition approach (similar to [13]), the database cache was set to 320 MB. For the Index Fabric, the database cache was set to 100MB and the embedded engine code was running in a JVM configured with 128 MB of memory from which 15MB was used for caching[2] (only 243 MB total, compared to 320MB). The size of the 3 indexes created for the vertical layout was 5.6 GB; the Index Fabric size was 1.6 GB.

The data in both setups consisted of 1 million products, with 100 attributes per product[3]. The queries were for product ID, by attribute/value pairs. For the vertical layout, the queries had from 1 to 4 different attribute value pairs. In order to see the trend with Index Fabric we ran queries with 1, 2, 5, and 10 attribute value pairs. We queried for both common and uncommon attributes, assuming an evolving schema. It should be noted that these relatively "flat" multiple attribute queries are equivalent to contiguous path queries (no intervening wildcards) often seen in semistructured queries.

With the Index Fabric, searching for increased numbers of attributes scales linearly. The same search over the vertical table rapidly degrades to unusable, even with a small number of attributes. This is an indicator of the high cost for multiple attribute queries, and also for long document path construction. Searching a completely open schema then degrades quickly, especially as the number of queried attributes grows.

Several differences combine to account for the results seen in Figure 6, primarily that the vertical approach joins every attribute name with an attribute value. This generates, in many cases, operations on large intermediate result sets. For example, a query for products with 'price=100' first considers all products that have a price attribute. With the Index Fabric, the composite index eliminates the join. Maintaining a similar index with relational DBMS is not practical with an evolving schema; the space requirements balloon since attribute values would be duplicated for every relevant attribute name. Since the attribute value field is large (contains names of products, suppliers, descriptions etc.), it constrains the options for the system and effective indexes. The size of the indexes built over the vertical table can potentially be *two orders* of magnitude larger than the indexes built over the horizontal table. The Patricia structure of the Index Fabric offers high compression, minimizing the size of the index.

As mentioned above, the size of the indexes created for the vertical layout was 5.6 GB, the Index Fabric size was 1.6 GB. In addition, with Index Fabric, the attribute names, the attribute values and the products are each only stored once in different tables. With the relational DBMS, in order to maintain composite indexes, the attribute values (and the attribute names) would need to be stored adjacent to the

---

[1] Our licensing agreements do not allow us to publish the DBMS brand.

[2] This cache was used to cache the non-leaf blocks of the Index Fabric as well as some of the leaf blocks and data records.

[3] We tested different data sets with different numbers of attributes and received similar results.

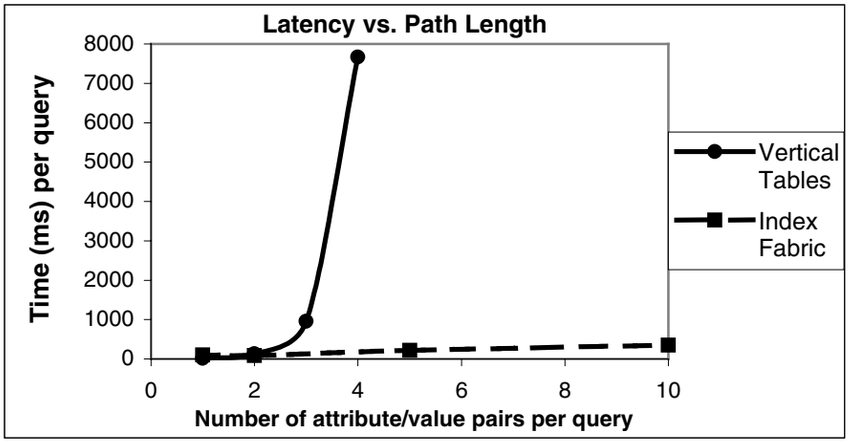


Fig. 6. Latency results

product IDs, causing the system to revert to the horizontal orientation. With the Index Fabric all the keys addressing the same product have the same row IDs, whereas with the vertical schema attribute/value pairs of the same product address different rows in the vertical table. With vertical tables, once the row IDs for each attribute/value pair were found (by a potentially costly join operation), the actual tuples are read to find if multiple values belong to the same product. Said differently, each attribute/value pair returns a set of row IDs. To decide which of the row IDs point to tuples satisfying the query, the system needs to retrieve, for each row ID, the product ID from the vertical table. This is what caused the vertical system to behave exponentially with increased number of attribute/value pairs.

4 Conclusions

We have addressed some of the complications associated with indexing open and evolving schemas at both a theoretical and practical level. We mapped out the steps for handling open schemas, covering each of the major issues: key encoding, data interfaces, updating the data, and query support. These steps are generally applicable and may be interpreted broadly for any semistructured data format.

We also present an index well suited for use with open schemas, the Index Fabric. The Index Fabric is a self-describing index. It is based on two cardinal advantages: the use of designators to store properties and structure per object's key, augmented by a compact, high-performance index structure [5]. These features combine to make the Index Fabric an ideal substrate for semistructured and irregular data sources that change over time. Previous experiments demonstrated that the Index Fabric is a successful index for semistructured data, and the preliminary experiments presented here demonstrate that the Index Fabric is an effective index for data with an evolving schema.

References

1. A. Deutsch, M. Fernandez and D. Suciu. Storing semistructured data with STORED. SIGMOD, 1999.
2. J. Cheng, J. Xu. XML and DB2. ICDE 2000.
3. "XML Support in Oracle8i and Beyond." Oracle technical whitepaper, November 1998. http://otn.oracle.com/tech/xml/htdocs/xml_twp.html
4. "Tamino White Paper." A Software AG report, October 2001. http://www.softwareag.com/tamino/download/tamino.pdf
5. B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A fast index for semistructured data. In Proc. VLDB, 2001.
6. J. Clark, ed. XSL Transformations (XSLT). November 1999. http://www.w3.org/TR/xslt
7. B. Cooper and M. Shadmon. The Index Fabric: A mechanism for indexing and querying the same data in many different ways. Technical Report, 2000. http://www.rightorder.com/technology/overview.pdf
8. P. Bumbulis and I. Bowman. A Compact B-tree. SIGMOD, 2002.
9. P. Ferragina and R. Grossi. "The String B-Tree: A New Data Structure for String Search in External Memory and its Applications." Journal of the ACM, 1998.
10. R. Goldman and J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. VLDB, 1997.
11. B. Cooper, N. Sample, and M. Shadmon. A parallel index for semistructured data. ACM SAC, 2002.
12. D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. INRIA Technical Report 3684, 1999.
13. R. Agrawal, A. Somani, and Y. Xu: Storage and Querying of E-commerce Data. In Proc. VLDB 2001.
14. Notes on the Xschema/DDML W3C submission process, http://purl.oclc.org/NET/ddml
15. N. Sample and M. Shadmon. Indexing Open Schemas. Web Databases 2002. http://www.rightorder.com/webdbws021.pdf (full version of this paper)

On the Effectiveness of Web Usage Mining for Page Recommendation and Restructuring

Hiroshi Ishikawa, Manabu Ohta, Shohei Yokoyama, Junya Nakayama, and Kaoru Katayama

Tokyo Metropolitan University

Abstract. The more pages Web sites consist of, the more difficult the users find it to rapidly reach their own target pages. Ill-structured design of Web sites also prevents the users from rapidly accessing the target pages. In this paper, we describe two complementary approaches to Web usage mining as a key solution to these issues. First, we describe an adaptable recommendation system called the system L-R, which constructs user models by classifying the Web access logs and by extracting access patterns based on the transition probability of page accesses and recommends the relevant pages to the users based both on the user models and the Web contents. We have evaluated the prototype system and have obtained the positive effects. Second, we describe another approach to constructing user models, which clusters Web access logs based on access patterns. We also have found that the user models help to discover unexpected access paths corresponding to ill-formed Web site design.

1 Introduction

The systems supporting the users in navigation of the Web contents such as [9] are in high demand. This is because it is more difficult for the users to rapidly reach their own target pages as an increasing number of Web sites consist of an increasing number of pages. Furthermore, ill-structured design of Web sites prevents the users from rapidly accessing the target pages. The support for detecting access paths contrary to the Web site administrators' expectations is also in high demand. The key solution to these issues is Web usage mining. In this paper, we describe two complementary approaches to Web usage mining. First, we describe an adaptable recommendation system called the system L-R (Log-based Recommendation), which constructs user models by mining the user access logs (e.g., IP addresses and access time) and recommends the relevant pages to the users based both on the user models and the Web contents [6]. We construct the user models by classifying Web access logs and by extracting access patterns based on the transition probability of page accesses. Second, we describe another approach to constructing user models, which clusters Web access logs based on access patterns. The user models help to discover unexpected access paths corresponding to ill-formed Web site design as well as play a basic role in recommendation.

In this paper, we explain the functional aspects of the system L-R as a first approach to Web usage mining in Section 2, describe the experiment and evaluation

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 253–267, 2003.

of the system in Section 3, and describe user model construction based on clustering as a second approach to Web usage mining in Section 4.

2 First Approach to Web Usage Mining: System L-R

Indeed, the recommendation systems [9] are very helpful in directing the users to the target pages. However, applying recommendation systems to existing Web sites requires a large amount of work and knowledge. So we propose a recommendation system that assumes no detailed knowledge about the recommendation system and that can be adapted to any existing Web sites without rewriting the contents.

The proposed system has the following merits:

(1) It is easy for the Web site administrator to incorporate the recommendation facility because the system utilizes only Web usage logs. The system doesn't impose the users to explicitly input their profiles.
(2) It is possible for the Web site administrator to provide multiple recommendation methods by considering the objective of the site and the tendency of the users.
(3) It is possible for the users to interactively choose their favorite recommendation methods.

The system is intelligent in that it can automatically mine the Web log data and construct the user model based on the Web usage and recommend the pages fit for the user.

2.1 Extraction of Web Usage Logs

Our system recommends the relevant pages to the users based on the probability of the user's transition from one page to another, which is calculated by using the Web access logs. First, we delete the unnecessary log data left by so-called Web robots (i.e., sent by search engines) by heuristically finding them in Web access logs [4]. In general, robots are recommended to have access to a file "robots.txt" created by the Web site administrator as the Web policy about the robot access. Moreover, since robots leave their own names in the Web log as host names or user agents, we prepare the list of robots in advance. Thus, we can find most robots in the Web logs by checking access to the robots text or searching in the robot list. Please note that there is small possibility that non-human data log, such as brand-new robots and page scanning tools, remain in the Web.

Second, we collect a sequence of page accesses by the same user within a predetermined time interval (e.g., 30 minutes) into a user session by using the genuine (i.e., human) Web logs. At the same time, we correct the session by deleting the transition caused by pushing the back button in Web browsers. For example, assume that the user browses the page A and the page B and browses again the page A by pushing the back button and then the page C. We change the transition from B to A to C into the transition from B to C (See Fig.1).

Lastly, we calculate the probability of the transition from the page A to the page B denoted by $P_{A->B}$ based on the modified Web log as follows:

$P_{A->B}$ = (the total number of transitions from A to B) / (the total number of transitions from A).

Note that the transition probability of the path from A to B to C which we denote by A->B->C (where the length of the path is two) is basically calculated as follows: $P_{A->B->C} = P_{A->B} * P_{B->C}$.

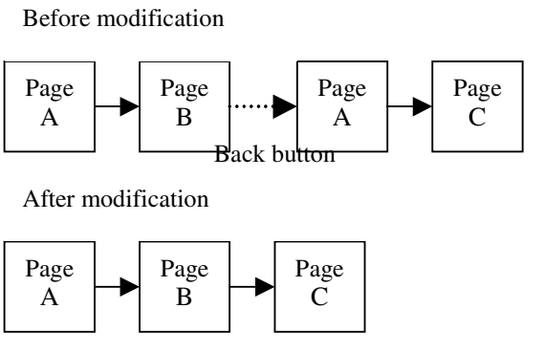


Fig. 1. Log modification w.r.t. back transition

2.2 Classification of Users

We classify the Web users by using information extracted from the Web access logs. The information includes IP addresses, domain names, host names, access time, OS, browser names, refers (i.e. the site just previously visited by the user), and keywords for retrieval. We describe the methods of classification based on such kinds of information. We construct user models by applying the methods of classification to the modified Web log data and realize the recommendation fit for the users based on the user models. We show the viewpoints of classification based on the Web log data as follows:

(1) IP address. We count the same IP address to get the number of access times. We recommend pages to the user according to their frequency of access to the Web site. For example, the frequent users are likely to browse the frequently updated pages. New comers are likely to browse "getting started" pages.
(2) Domain and host name. The names of user domains and hosts indicate the place from which the users are accessing the Web site. We classify the users according to the places. For example, the users from companies may browse the company information of EC sites while the general users may browse the product pages
(3) Access time. The users have different tendencies according to the time when the users are accessing the Web site. During the daytime, the housewives and businesspersons often access the site. At midnight, students often access the site.
(4) Day of the week. The weekday and weekend indicate different tendencies. For example, the pages about the leisure information are more often accessed on weekends than weekdays.

(5) Month and season. Month and season indicate different tendencies. For example, the pages about traveling are more often accessed during holiday seasons.
(6) OS. As a general rule, most users use Windows 98 and 2000. Business users often use Windows NT and Unix. It is possible to recommend different products according to OS.
(7) Browser. From the browser information, we can know the language that the user uses. We can automatically change the display language according to the browser.
(8) Keyword. We can conjecture the pages more correctly that the user intends to access by user-input keywords.
(9) Refer. We can make recommendation more fit to the user by knowing the site that the user visited just before coming to the current site.

Of course, we can know other information such as the types of requests (e.g., get, put) and the size of transferred data from the Web logs, too. However, for the moment, we don't know the effect of the classification based on these kinds of information.

2.3 Recommendation

We provide various methods for recommendation, which can be classified into two groups: pure probability-based methods and weighted probability-based methods.

(1) Pure-probability-based methods. The following recommendation methods are based solely on the transition probabilities calculated from the Web log data.

– **Path recommendation.** The whole path with high transition probability such as pages A, B, and C is recommended to the user (See Fig.2). The recommended path reflects the series of pages, which are usually browsed in sequence such as instructions stretching over manual pages. Of course, the path longer than two can also be recommended.

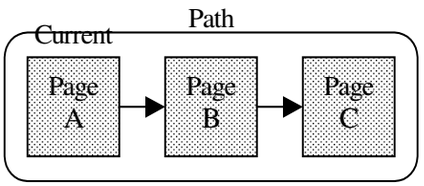


Fig. 2. Path recommendation

– **History-based recommendation.** Based on the pages visited before coming to the current page, the next page with high path transition probability is recommended. Please note that the path transition probability only in this case is calculated by counting the whole path (i.e., A->B->C) differently from the previous definition. In other words, even if the current page is the same, the next recommended pages differ depending on such histories. For example, either C if A->B or E if D->B is recommended (See Fig.3). Of course, the history path older than two can be utilized too.

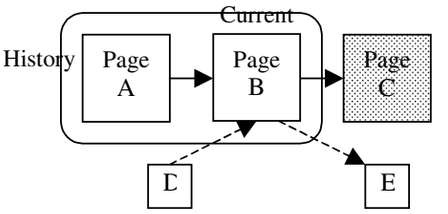


Fig. 3. History-based recommendation

– **Future prediction-based recommendation.** A page ahead by more than one link is recommended based on the path transition probability. This can let the user know what exists ahead in advance. Any link ahead (i.e., any future path farer than two) can be recommended, but it is also possible that the user may miss their true target between the current page and the page far ahead. For example, if the path A->B->C has the highest probability, then the user at A is recommended C (See Fig.4).

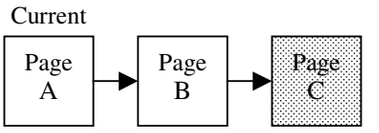


Fig. 4. Future prediction-based recommendation

– **Shortcut recommendation.** The transition by the back button is viewed insignificant and then the path is shortcut. For example, there is a path A->B->A->C where B->A is done by the back button. The path is shortened to A->C.

(2) Weighted probability-based methods. The following recommendation methods are done based on the probability weighted by other aspects such as the time length of stay, the Web policy, and the Web content structures:

– **Weighted by time of stay.** The page where the user stays longest is recommended. It can be known by using the Web log data. For example, if there is a path A->B->C where C is longest referenced, the weight of A and B with respect to C is increased by one. Then C is recommend at A or B.
– **Weighted by most recent access.** The probability is weighted by the importance of the target pages that we judge by most recent access. For example, if there is a path A->B->C where C is most recently referenced, the weight of A and B with respect to C is increased by one. C may be the page with detailed description of products. Then C is recommend at A or B.
– **Weighted by number of references.** If n pages reference a page within the same site, then the weight of the referenced page is increased by n. This method is validated by the observance that many pages link the important pages, which is similar to authorities and hubs in WWW [3].

3 Experiment and Evaluation of System L-R

We have implemented an experimental recommendation system and have evaluated the system by using the logs of the Web server of our department. In this section, we describe the statistical data of the Web logs, the problems and the method for evaluation, the experimental system, and the system evaluation in sequence.

3.1 Statistics

First, we show the statistical information about the Web log data used for the experiments as follows:

Access logs: 384941 records; Web Pages: about 170 HTML files; Log duration: from June to December 2000; Subjects: five In-campus (knowledgeable) students and five out-campus (less knowledgeable) students

Please note that we have excluded non-HTML files such as GIF, JPEG because they need to be handled separately.

3.2 Problems and Evaluation Method

Before evaluation, we have prepared problems corresponding to the possible objectives of the users visiting the department Web site. They include the following problems:

Problem 1. Find the major field of Prof. Ishikawa.
Problem 2. Find the contents of the course “electric circuit I”.
Problem 3. Find the contact information of the member of faculty doing the research on “efficient implementation of pipeline-adaptable filter”.
Problem 4. Find the contents of Phd theses for 1995.
Problem 5. Find the contact information of the member of faculty teaching the course on “communication network theory ”.

Next, we describe the method for evaluating the experimental system. We instruct the subjects to write down the number of transitions (i.e., clicks) and the time to reach the target pages as answers to the above problems. We evaluate the experimental system based on the figures. Each subject solves different problems by different recommendation methods.

3.3 Experimental System

For this time of evaluation, we have implemented the following four recommendation methods effective for our department site:

- Recommendation with back button transitions
- Recommendation without back button transitions
- Future prediction-based recommendation
- Recommendation Weighted by number of references

Please note that the first two methods are simple methods which recommend the next page based on pure transition probability. They are used just for the purpose of comparison with the last two methods as our proposal. We use two user groups based on IP addresses: In-campus students and out-campus students.
We have implemented the recommendation system by using frames (See Fig. 5). The upper frame displays the original Web page and the lower frame displays the recommendation page. Both of the frames change in an interrelated manner. When the user moves from one page to another based on the recommendation, the recommendation page is changed accordingly. The home page allows the user to select favorite recommendation methods. The recommendation page frame displays titles, URLs, and ranks of the recommended pages. The ranks are based on the transition probability calculated from the Web logs. At most five pages are recommended to the user. The user either may or may not accept the recommendation.

Fig. 5. User interface of recommendation system

3.4 Evaluation

We illustrate the graphs indicating the average clicks and time required for solving the five problems (See Fig.6). 1, 2, 3, 4, and 5 indicate no recommendation, recommendation with back button transitions, recommendation without back button transitions, future prediction-based recommendation, and recommendation weighted by number of references, respectively.

First of all, the result shows the positive effect of recommendation because all methods (2-5) decrease the clicks and time in comparison with no recommendation (1). Recommendation without back button transitions (3) is better than

recommendation with them (2), so we think that the modified Web logs reflect user access histories more correctly. Future prediction-based recommendation (4) is a little bit worse than recommendation without back button transitions (3). This is probably because the former can skip the user's target mistakenly. Recommendation weighted by number of references (5) is the best of all although recommendation without back button transitions (3) is a little bit better in the number of clicks.

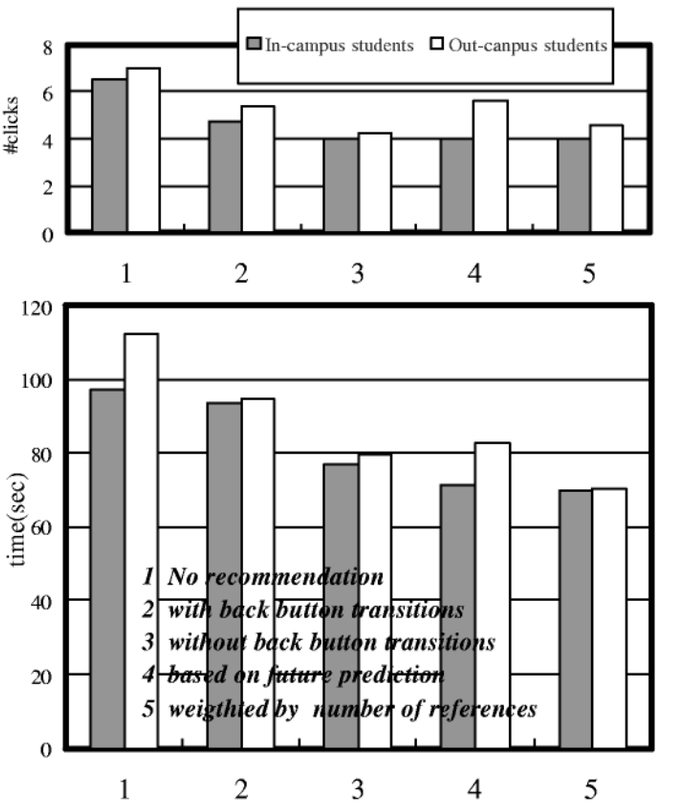


Fig. 6. Number of clicks and time for types of recommendation

The fact that there is difference between in-campus students and out-campus students suggests that we need to classify the user groups. The difference can be shortened by our recommendation system. This indicates that our system is more effective for less-knowledgeable people such as out-campus students.

4 Second Approach to Web Usage Mining: Clustering

4.1 User Model Based on Clustering

We summarize our first approach described earlier. First, we have classified the Web access log data according to the attributes such as in-campus/out-campus students in advance. Then we have extracted access patterns recurring in them and have constructed the user models according to them and have recommended relevant pages based on the user models.

Here we describe a complementary approach to Web usage mining. First, we cluster Web access log data based on access patterns. Each cluster corresponds to a specific user model with access patterns as its features. Then, we can use the models for recommendation. We can also use the clusters by extracting unexpected access patterns when we redesign Web sites to remedy such inappropriate structures.

Cluster	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Number of sessions	195	171	144	135	67
Most frequently accessed page/ratio(%)	Road show 17	Road show 26	Road show 18	I-congestion 14	Floor 16
i-mode page ratio (%)	27	20	47	61	33

(a) collection type

Cluster	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Number of sessions	214	161	124	120	93
Most frequently accessed page/ratio(%)	Road show 17	Road show 24	Road show 16	Floor 11	Lineup 16
i-mode page ratio (%)	53	20	36	42	25

(b) set type

Fig. 7. The result of page reference clustering

We describe how we utilize clusters as the user models for recommendation in more detail. First of all, we match clients with user models (i.e., clusters) and determine the most likely model. Then we recommend the clients relevant pages according to frequent access patterns in the selected cluster. At this time, we can use the various strategies for recommendation described earlier as our first approach. In matching the clients with user models, we can use access patterns (i.e., IP addresses, access date and time, access pages), which we use in clustering user models. In matching, of course, we can also use the other attributes of the Web access log not used for clustering itself. We extract characteristics of such attributes from the constructed clusters by combining them with the original Web access log data.

Next we describe how we use clusters for redesigning Web site structures. We can find access patterns contrary to the expectations of the Web site administrators. It is possible that they indicate ill-formed structures of the Web sites. Thus, we can refine Web sites by resolving such ill-formed structures discovered in cluster analysis.

4.2 Experiments

(1) Statistics

We have done experiments to validate this complementary approach described above. First, we describe the experiment settings. We use another Web log: The Web site is one for a cinema complex at Tachikawa City. The Web log contains access log records for Sunday, October 7th, 2001 consisting of 6402 records accessing distinct 138 html pages. We extracted 712 user sessions. We determined one session of a consecutive access from the same IP whose intervals are within 40 minutes for this site.

2) Session Representation

We provide two types of representation to sessions: Page reference and page transition vectors. We represent each page reference vector as a 138-dimensional vector plus IP and last access time, whose each component corresponds to a distinctive referenced page. Further, we provide two sub types of representation to page reference vectors: Set type and collection type. The set type of reference vectors indicate the existence of references to each page (i.e., each component contains either 0 or 1) while the collection type of reference vectors keep records of the times of references to each page (i.e., each component contains 0 or a larger integer). We exclude accesses to only one page as we consider them noisy.

We represent each page transition vector as a 19044 (= 138*138)-dimensional vector plus IP and last access time, whose each component corresponds to a transition from one page to another (i.e., directed one). We also provide two types of representation: Set and collection. Similar to page reference vectors, they indicate the existence and counts of a specific transition, respectively. In other words, we have four different types of clustering, that is, set- and collection-typed page reference clustering and set- and collection-typed page transition clustering. They are selectively used depending on specific cases of clustering. Of course, we exclude accesses to only one page.

(3) Ward Clustering

We use Ward's method for clustering sessions: We start with a node in a separate cluster with the node as its center of gravity. In each step, the two clusters that are closest according to a distance measure (e.g., Euclidean distance) are merged and a new center of gravity is computed for this new cluster by using those of the two merged clusters. We repeat this step until we get the target number of clusters.

In Ward's method, we have to determine the number of target clusters. To this end, we provide a visual aid for the user to choose the appropriate number of clusters. The aid graphically plots the minimum values of all cluster-distances vs. the number of

Cluster	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Number of sessions	250	192	83	82	76	29
Most frequently accessed time	10	11	12	18	8	19
Most frequently accessed page/ ratio(%)	Road show-> Lineup 13	Road show-> Next road show 6	I-congestion ->City2 7	Road show-> Lineup 12	Road show-> Lineup 7	Next road show-> Road show 17

(a) collection type

Cluster	Cluster1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Number of sessions	279	133	110	71	64	55
Most frequently accessed time	23	12	11	14	13	12
Most frequently accessed page/ ratio(%)	Road show-> Lineup 11	I-congestion ->City1 5	Road show-> Lineup 7	Road show-> Lineup 14	Road show-> Lineup 12	Next road show-> Road show 10

(b) set type

Fig. 8. The result of page transition clustering

clusters. With this help, we have detected a big change of the minimum distance when the number of clusters changes from 5 to 4 at page reference clustering. So we choose 5 as the number of target clusters at page reference clustering. For the same reason, we choose 6 as the target number of clusters at page transition clustering.

(4) Clustering Result

First, we illustrate the result of page reference clustering (See Fig. 7). We counted the total number of sessions, the time of the most frequent accesses, the most frequently accessed page and its ratio, and the "i-mode" page (i.e., internet access through mobile phones) ratio for each cluster. We focus on collection-typed clustering (See Fig.7 (a)). The cluster 4 contains the largest number of "i-mode" users with 61%. On the contrary, the cluster 2 contains the largest number of PC users. The users in the cluster 5 most often visit the "floor" page, where the theater facilities are described as expertise. We can use these clusters as the user models on which recommendation is based. For this specific case of clustering, we prefer collection to set because the characteristics extracted from collection-typed clustering are more emphasized. For example, the cluster 2 of set-typed clustering has 161 sessions, "road show" with 24%, and "i-mode" page with 20% (See Fig.7(b)).

Next we also focus on the result of collection-typed page transition clustering. We counted the total number of sessions, the time of the most frequent transitions, the most frequently transitioned page pairs and its ratio for each cluster. At this time, the collection-type also emphasizes the characteristics more clearly than the set-type.

The most frequent transition in the cluster 6 of collection-typed page transition clustering has "next road show->road show" with 17% while the corresponding cluster of set-typed clustering has the same transition with 10% (See Fig. 8). We can interpret this observation as follows: The day of the Web log is late Sunday. Therefore the users have access to the "next road show" page for the next week schedule. However, the page on Sunday is empty and is updated on next Tuesday. Until then, the current schedule page "road show" contains the schedule from Saturday to Friday. Then the users notice the wrong visit and go to the right page, which causes this anomalous transition. Of course, this indicates the inappropriate design of Web pages contrary to the users' expectations. In other words, the observation suggests that the Web site administrator should redesign the Web site structures more understandably.

5 Conclusion

5.1 Summary

Our main contribution is that we have validated the effectiveness of Web usage mining for page recommendation and restructuring through empirical studies. First, we have proposed an experimental recommendation system L-R based on both Web usage mining (i.e., transition probability calculation) and Web site structures. From the result of the evaluation of the system L-R, we have been able to indicate the following findings:

(1) The recommendation based on Web usage mining is positively effective. All recommendations based on modified Web logs in respect to back button transitions are more effective than that based on pure Web logs.
(2) The recommendation based on user classification is positively effective. We have found that the amount of knowledge about the Web site affects the number of clicks and the time to the destination pages. Recommendation is more helpful for the less-knowledgeable people. So the classification with respect to the point is valid.
(3) The recommendation based on Web page structures is positively effective. The one based on both the modified Web logs and Web page structures (i.e., hybrid approach) is the most effective. This finding is one of our contributions.

Second, we have described user model construction based on automatic clustering. We have been able to indicate the following findings:

(1) Collection-typed page reference clustering extracts features of each cluster more obviously than set-typed one. The result can play a basic role in recommendation.

(2) Collection-typed page transition clustering extracts features more obviously than set-typed one. The result can help to redesign ill-structured Web sites. Our findings about collection-typed clustering techniques are also among our contributions.

However, we also have found remaining issues, in particular, in the implementation of the System L-R as follows.

(1) We have copied the original Web site and Web logs to the virtual site from the experiment and evaluation of the system. So we must rewrite the contents for the virtual site, such as modification of URL. Moreover, the contents in the virtual site are not completely synchronized with those in the original site. However, simple solution such as just incorporating the recommendation system to the original site is also problematic because the user access according to the recommendation modifies the access logs.

(2) The size of the Web logs is no so large that we could not implement the other methods such as path recommendation. This is because the multiplication of the probabilities tends to be quite small in the small-sized Web logs.

(3) The number of problems and subjects in evaluation of the System L-R is not so large enough to give very confident results. In particular, it is difficult to prepare as distinctive types of problems as possible.

5.2 Related Work

We will compare our work with relevant works. Cooley et al. [1] have clarified the preprocessing tasks necessary for Web usage mining. Our approach basically follows their steps to prepare Web log data for mining.

Srivastava et al. [10] have surveyed techniques for Web usage mining comprehensively. Among various techniques, we use transition probabilities as a sequential variation of association rules for page recommendation and collection-typed clustering as page restructuring.

Spiliopoulou et al. [11] have proposed Web Utilization Miner (WUM) to find interesting sequential access patterns, which is similar to our approach for page recommendation. Unlike them, however, we construct various types of recommendation from the same set of sequential access patterns.

Perkowits et al. [8] have proposed Adaptable Web Sites as a method for providing index pages suitable for the users based on Web access logs. They have implemented the algorithm to extract user access patterns from the access logs. They provide virtual sites adapted to individual users by determining the links and their ranks (order) in the index pages based on the algorithm. The Adaptable Web Sites automatically recommend different pages depending individuals while our system recommends pages in several ways chosen by both the Web site administrator and the users.

Mobasher et al. [5] have proposed a method for recommending pages weighted by mining the user access logs. The system recommends pages according to the user access recorded in “cookies” while our system allows the user to choose among several recommendation methods and takes page structures into consideration. The recommended pages change depending on the current page but not on the cookies.

Kiyomitsu et al. [2] [12] have proposed a system for displaying different link structures according to the user access logs and a mechanism for the administrator to

easily change the contents. They change pages according to the user access logs while we keep the original contents intact and allow any site to add on recommendation. Their mechanism is solely intended for the Web site administrators. Our recommendation is intended for both the users and the Web site administrators in that several recommendation methods can be chosen by both of them.

Ohura et al. [7] have proposed an approach, which clusters access patterns based only on "set-typed page reference session vectors" and uses them to expand users' queries. Our approach provides four types of clustering to serve different purposes such as page recommendation and restructuring.

5.3 Future Work

First, we will improve the evaluation result by the following ways without changing the system:

(1) Increasing the problems and subjects
(2) Increasing the size of the Web logs by using the longer duration of the logs
(3) Classifying users into groups form different viewpoints

Second, we will generalize our recommendation system as follows:
(1) Adapting the system to non-HTML contents such JPEG, MPEG, MP3
(2) Adapting the system to different sites
(3) Using the user models based on automatic clustering of the users

The above three ways to generalization can require the system to change more or less.

Acknowledgements. This work is partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Japan under a grant 14019075. It is also partially supported by the Tokyo Metropolitan University under a president special grant 142.

References

1. R. Cooley, B. Mobasher, and J. Srivastava: Data Preparation for Mining world Wide Web Browsing Patterns, Journal of Knowledge and Information systems, vol.1, no.1, pp.5–32, 1999.
2. H. Kiyomitsu, A. Takeuchi, and K.Tanaka.: Dynamic Web Page Reconfiguration Based on Active Rules, IPSJ Sig Notes, vol.2000, no.69 (2000-DBS-122), pp.383-390, 2000 (in Japanese).
3. J. Kleinberg: Authoritative sources in a hyperlinked environment. Proc. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998.
4. T. Mizuhara, T. Nakajima, M. Ohta, and H. Ishikawa: Web Log Data Analysis for Recommendation System, Proc. IPSJ National Convention, 4W-3, 2001 (in Japanese).
5. B. Mobasher, R. Cooley, and J. Srivastava: Automatic Personalization Based on Web Usage Mining, CACM, vol.43, no.8, pp.142–151, 2000.

6. T. Nakajima, T. Mizuhara, M. Ohta, and H. Ishikawa: Recommendation System Using User Models based on Web Logs, Proc. IPSJ National Convention, 4W-4, 2001 (in Japanese).
7. Y. Ohura and M. Kitsuregawa: Examination of the expansion of users' search queries based on clustering Web access logs: Experiments on the i-townpage, IEICE Data Engineering Workshop, A2-1, 2002 (in Japanese).
8. M. Perkowittz and O. Etzioni: Adaptive Web Sites, CACM, vol43, no.8, pp.152–158, 2000.
9. J. Schafer, J. Konstan, and J. Riedl: Recommender Systems in E-Commerce. Proc. ACM Conference on Electronic Commerce (EC-99), pp.158–166, 1999.
10. M. Spiliopoulou, L.C. Faulstich, K.Winkler: A Data Miner Analyzing the Navigational Behavior of Web Users, Proc. Workshop on MachineLearning in User Modeling of ACAI99, 1999.
11. J. Srivastava, R. Cooley, M. Deshpande, P. Tan: Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, SIGKDD Explorations, vol.1, no.2, pp12–23, 2000.
12. A. Takeuchi, H. Kiyomitsu, and K.Tanaka.: Access Control of Web Content Based on Access Histories, Aggregations and Meta-Rules, IPSJ Sig Notes, vol.2000, no.69 (2000-DBS-122), pp.315–322, 2000 (in Japanese).

XML Fragment Caching for Small Mobile Internet Devices

Stefan Böttcher and Adelhard Türling

University of Paderborn
Fachbereich 17 (Mathematik-Informatik)
Fürstenallee 11 , D-33102 Paderborn , Germany
{stb,Adelhard.Tuerling}@uni-paderborn.de

Abstract. Whenever small mobile devices with low bandwidth connections to a server execute transactions on common XML data, then efficient data exchange between client and server is a key problem to be solved. However, a solution should also consider client-side cache management for the XML data, synchronization of concurrent access to the XML data, and lost connections during transaction execution. In order to reduce data exchange between client and server, our protocol reuses data stored in the client's memory instead of reloading data into the client's memory wherever possible. A key idea is that the server keeps a 'living copy' of the XML fragments in the client's memory for efficient cache management. Furthermore, our protocol integrates well with a validation based scheduler in such a way that offline work and transaction completion after lost connections are supported. Finally, we present some optimizations that further reduce client-server communication.

1 Introduction

1.1 Problem Origin and Motivation

There is a growing interest in XML as a data representation language and data exchange format for different web based enterprise applications. Whenever complex applications include small mobile devices in distributed transactions, then resources like communication bandwidth and available memory are known to be bottlenecks that make mobile transactions slow or even impossible.
Our work is motivated by a web-database application [2], that now shall be ported to small mobile devices, which communicate via low bandwidth connections with a server. Within our application, several client transactions read or modify (their local copy of) a virtual XML document, which is stored on a server.
Our approach optimizes the data communication between client and server needed to support client transactions, i.e. to keep the client's copy of the server's XML document up to date.

1.2 Relation to Other Work and Our Focus

Our work is related to contributions in different fields ranging from XML databases, over cache management in distributed databases and transaction management for mo-

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 268–279, 2003.

bile devices, and focuses on optimization of client-server communication over low bandwidth connections.
Recently there has been much research in the area of XML databases, e.g. [8,17], with an increasing interest in query optimization for XML data (e.g. [13]) and efficient storage of XML data (e.g. [11]). While other contributions on cache management and query optimization assume, that there is a fixed and fast connection from the main memory used by the client application to the XML database, we focus on special optimizations needed for small bandwidth connections between a client application and a server-side XML document.
Additionally, many contributions have proposed ideas for cache management in distributed DBMS, e.g. [4,7,12]. While these contributions integrate the cache in the query processing environment by caching the results of queries, other contributions improve cache management by the storage of deltas, e.g. for OODBMS [6] and for XML [14]. Common to the last approach, we compute deltas, however, we compute delta XML fragments on the server side and our focus is to restrict the communication between server and client to the exchange of delta XML fragments.
Furthermore, mobile transactions are considered to be an additional challenge due to limited bandwidth and frequent disconnection [16,21]. Compatible to related contributions on mobile transactions (e.g. [3,5]), we assume an optimistic approach to transaction synchronization. However additionally, we integrate optimized cache management with validation.
Finally, while there has been much excellent work on scaling up XML database technology to large databases, our work can be characterized as downsizing database technology, which is considered to be very important but quite difficult [1,9]. But different from other work that downsizes database technology, e.g. [10,15,18], we ”downsize” the exchange of XML fragments which leads to different requirements.
Different from all other contributions, we focus on cache management and reduced client-server data exchange tailored to the specific needs of XML databases which are accessed from mobile clients via small bandwidth connections.

2 Underlying Concepts and Problem Description

We offer two possible ways for clients in our transaction approach to access fragments of an XML document stored on a server: *direct requests* use XPath [20] expressions to access XML fragments, whereas *indirect requests* are transparently initiated by calling DOM (document object model) [19] methods of our client API. We summarize the underlying techniques (XPath, DOM) used for these requests, before we describe the problem.

2.1 Direct Requests Use XPath Expressions

As mentioned, XPath, the query language of XML, is used to describe which fragment of the server-side document shall be transferred to the client’s memory. A read operation, e.g.:

```
doc.read( ”/BookingService/Connections/Connect[ @id=’3’ ]//* ” )
```

has as argument an XPath expression that describes a set of nodes of the underlying XML document. The path '/BookingService/Connections/Connect' starts at the root '/' of the XML document and uses the child axe, i.e. it identifies 'Connect' elements under 'Connections' elements under a 'BookingService' element. The predicate filter [@id='3'] restricts the set of selected 'Connect' nodes to the node with the attribute 'id' set to the value of '3'. Finally, '//*' expresses that all descendent nodes of the node(s) selected by '/BookingService/Connections/Connect[@id='3']' are included in the request.
XPath expressions are used to communicate, log, and analyze not only the client's read operations, but also the client's write operations (e.g. in the context of transaction validation), i.e. all XML fragments read or written by clients are described by XPath expressions on the server side.

2.2 Indirect Access through the DOM Interface

In addition to direct requests based on XPath, the client can use a mobile DOM interface ($MDOM_{tr}$). This interface supports the main methods of the DOM [19] like reading or writing node values, appending or deleting nodes, and navigating on the document object model using inter-child and parent-child relations. Because we focus on data exchange, the API allows the user to query for node-names and to generate nodes as defined[1] for the document. $MDOM_{tr}$ also differs between node-types of the DOM (like attribute, element, or comment), but leaves the document validation to the server.

2.3 Problem Description

Within our application scenario, several clients share access to a *virtual XML document*, i.e., the client's view on the data source is the complete XML document no matter what is the real representation on the server - and ignoring that the client only works on parts (fragments) of the XML document and concurrently with other clients. The virtual XML document is stored or made available on a server and XML fragments are transferred to clients, where they are read or modified. Clients are expected to use a DOM API for their access to the server-side XML document and to use XPath expressions to preselect or focus an interconnected fragment of the XML document they want to access in a transactional context.
Our clients face two partially contrary main problems: First, since these clients use low bandwidth connections to a server, data exchange shall be reduced to a minimum. Second, efficient use of the client's memory is a key requirement, because the client's data memory may be too small for the data needed for client applications.
More specific, we expect that most but not all of the client operations access rather small XML fragments that fit into the client's memory. We further expect that a considerable part of a single client's transactions use the same (personal) subset of the XML data again and again. For these transactions, it makes sense, to reuse data already stored in the client's memory wherever possible, even across the boundaries of

[1] E.g. defined by a server-side DTD or XML schema

a single transaction. Therefore, our focus is to optimize memory management of DOM applications running on small internet devices not only beyond the boundaries of a single operation, but also beyond the boundaries of a single transaction.
Since clients are mobile, they may loose their connection to the server, but transactions should get the opportunity to complete after the connection is reestablished in order to be as smart on resources as possible. Furthermore, each transaction involves only a single client (and the server) and each client runs at most one transaction at a time. While synchronization of such concurrent transactions can be done on the server as described e.g. in [3], here, we discuss the transaction's benefit of reused data from previous transactions left in the client's memory.

3 Overview and Key Concepts

In this section, we outline $MDOM_{tr}$ a mobile client's DOM interface which allows the client to interact with a server-side XML document concurrently with other clients. Our approach allows the client to operate on arbitrary parts of a DOM of the server's XML document, while data loading form the server to the client and transaction management are performed by our protocol transparently to the client application. The presented protocol uses available client memory to cache data and optimizes data exchange between client and server. For this purpose, our approach extends the server-side XML database system with server-side memory areas for each client, where the server keeps an identical physical copy of each client's data memory and logical representations of each client's transactional operations. In Section 4, we discuss the displacement strategy used on both, the client's memory and the server's copy, and the achievable communication reduction. Section 5 presents the transaction management adapted to the needs of our mobile clients. We end up with Summary and Conclusions in chapter 6.

3.1 The Data Structure for $MDOM_{tr}$

As mentioned in Section 2.2, we implement our approach as a small mobile-client DOM API ($MDOM_{tr}$) that accesses a server-side XML document in a transactional context. We reserve a fixed amount of the client's memory (called the *client's cache*) at application initiation time and use it for the storage of the XML fragment the client is currently working on. The cache size usually differs from client to client (it may be e.g. 256 KB) and often represents the client's maximum available (data-) memory, i.e. that part of the client's memory that is not needed for application program code, etc..
The $MDOM_{tr}$ data structure consists of a pointer connected tree[2] that represents the XML fragment of accessed data and associated inter-connected access time stamps which are used for displacement. The cache is organized in memory blocks with a

[2] Within our implementation, each node object is represented by 10 pointers, the data value and a time stamp. When each pointer, time stamp and decision criteria consumes 4 byte and each date value consumes 32 byte, we need 80 bytes/node, i.e. we can store 3200 nodes in 256 KB cache.

fixed precalculated[3] size. Such a memory block is able to hold any expected node making it easy to organize and displace it. Possible overflows of memory blocks caused by arbitrary input values are handled by using additional memory blocks. If no such block is available, other nodes have to be displaced as described in chapter 4. When a client accesses the DOM tree, it is transparent to the client application, whether data is already available in the client's cache or it has to be downloaded from the server when a node-object is accessed. The API achieves this transparency by using additional pointers to indicate whether siblings or children of a node are missing and have to be downloaded or are already available in the cache.

3.2 Our Client-Server Communication Protocol

Our protocol supports two main strategies for clients to access data. As introduced for read operations in chapter 2, we call these strategies *direct* and *indirect*. Whereas an indirect request initiated through the MDOM is possible for all operations (read, update, insert, delete) on the virtual XML document, direct requests are only supported for read and delete operations.
Direct requests use XPath expressions to identify the fragment the operation accesses. For example, a direct read access of the client transfers an XPath expression to the server, where the corresponding interconnected XML fragment is extracted from the virtual XML document. Only those parts of the fragment that are not yet stored in the client's cache are transferred to the client in order to save communication bandwidth. When the data is stored in the client's cache, the client can use $MDOM_{tr}$ operations in order to navigate within the retrieved fragments or to initiate any indirect request. - For the delete operation, it is sufficient to send the XPath expression to the server, because the XML fragment to be deleted can be extracted from the server's cache copy. Both, read and delete operations have in common that they are often applied to fragments and that the fragment itself is the only parameter that can be represented as a short (XPath) expression. This makes a direct request the best choice for fragment based read and delete operations.
On the other hand, indirect requests are used by iterated read, insert, update or delete operations on single nodes. Here, the strategy is to only transfer the physical data that have been changed back to the server. In the case of individual insert and update operations, transferring the new values for inserted or updated data is an optimum w.r.t. the amount of data that must be transferred[4], because the new data has to be exchanged anyway.
The server's response to a client's indirect write request or the direct delete request is a simple acknowledgement to synchronize the protocol if the request was well formed and no constraints where violated. Otherwise, the server refuses the write operation and repairs inconsistent data.
The protocol can follow different strategies (heuristics) to respond to indirect read or write requests. One strategy, optimizing the low bandwidth bottle-neck, is that the server returns each accessed node immediately and the client sends written data for each operation immediately. Other strategies allow the server or the client to send

---

[3] E.g. a memory block should be able to hold 90% of all existing nodes in the document.
[4] We focus on single data updates here and do not discuss mass updates that could be done by stored procedures.

fragments containing multiple nodes for read operations at once or data of multiple write operations at once. Notice that strategy may depend on different optimization goals like: *small bandwidth* or *minimal online time.*

3.3 The Server's Copy of the Client's Cache

One of our key ideas is to hold an identical physical copy of the client's cache on the server (called the *server's cache copy*) and to duplicate the client's operations on it. This is possible because any operation of the client changing the cache is communicated from the client to the server and because we use an identical prearranged displacement strategy[5] and an identical fixed prearranged cache size for both, the client's cache and the server's cache copy. Therefore, the server's *cache copy* is able to displace data exact the same way the client cache does without any additional client-server interaction. Because keeping the server's cache copy up do date doesn't involve any additional communication, its costs are only server memory and CPU time. Since the client's memory is rather small, the server's load on memory resources and CPU time to keep track of the client's operations is acceptable in our application[6]. As we'll see in chapter 4 the server uses its *cache copy* to optimize the data communication with the client.

4 Efficient Memory Management

Small bandwidth for the communication between client and server requires to avoid unnecessary exchange of large XML fragments or to exchange the same data repeatedly. Instead, reduced data exchange containing only client requests and cache modifications (on client or server) is preferred. Therefore, still available client-side data belonging to older and actual transactions should be reused whenever possible.

4.1 Flexible Displacement Strategy

As mentioned, the basic idea in order to reduce the data exchange between client and server is, that client and server have identical copies of the client's cache and use local, identical, and independent displacement strategies for their copy. Note that we do not require the client and the server to use a specific displacement strategy (e.g. LRU or FIFO), but we only require that they use an identical strategy during a transaction. In fact, the client can even choose the displacement strategy from a set of predefined displacement strategies – depending on the application.
In principle, a displacement strategy has to obey the following rules:

1. Nodes not accessed in the actual transaction have to be displaced before nodes accessed in the actual transaction.

---

[5] e.g. LRU or FIFO, limitations are discussed later.

[6] e.g. this would be one displacement-step per client interaction and 256 Mbytes of memory for 1000 clients

2. Only displace as few nodes as needed.
3. Displacement always starts at leaf nodes, i.e. it is not allowed to displace an inner node as long as its successor nodes are not displaced.
4. The displacement strategy has to obey locks.

4.2 Fixed Nodes and Override Rules

Additionally, client and server are allowed to fix and unfix fragments of nodes in the cache. Such nodes are not allowed to be displaced. This override rule is used by the client-side $MDOM_{tr}$ in order to prevent misleading pointers. Since the client-side MDOM decides locally to fix (or unfix) a node, it informs the server of its newly fixed (or unfixed) nodes together with the next client request. Similarly in the *delta refresh scenario* (as described in section 4.3), the server informs the client that some nodes arefixed, i.e. can not be displaced.
Be aware that although fix operations have to be communicated and fixed nodes decrease the cache available for displacement, fixed nodes can be used to enhance the offline workability of a client, because the application can be sure that the needed data is not displaced and no further read request to the server is needed for this fragment. As we will see in the *delta refresh scenario* described in section 4.3, a temporary fix operation initiated by the server is needed for a proper working protocol.

4.3 The Benefit of Caching Former Operations' Data

The goal of caching former operations' data is to reuse the data instead of reloading it wherever possible. First, read operations are performed locally (we will call this *zero refresh scenario*) whenever the client knows, that it has already read all the required data in the same transaction. If not, the client contacts the server.
Whenever a read or insert request is initiated by a mobile client, the limited capacity of the cache may require data replacement which leads to two additional scenarios which we call the delta refresh and the overflow scenario. Whereas update requests only affect time stamps and data values, they do not displace any nodes but should increase the chance of the updated nodes to stay in the cache. A delete request of the client only releases memory that can be used by data of the next operation.

Zero Refresh Scenario
As mentioned before, the client cache associates access time stamps with each stored node of the XML fragment, such that the client can compute whether the data has been loaded in the current transaction or in a previous transaction. In order to avoid unnecessary read requests for fragments that are completely in the client's cache, the client API first searches data in its local cache. If the needed fragment is completely in the client's cache and has a time stamp that belongs to the current transaction, then no read request is submitted to the server[7]. Note however, if the data found in the client's cache has a time stamp that belongs to an older transaction or the client does not contain all data it needs in its cache, then the client asks the server for actual data.

[7] A list of updated time stamps is sent to the server with the next communication step.

Delta Refresh Scenario
What we expect to occur most often in this protocol is the *delta refresh scenario*. This scenario takes place whenever a client requests to read a fragment, a part of which is already stored in the client's cache (called *existing fragment*). Then the server only has to send the missing data (called the *delta fragment*).
If the server now would send the *delta fragment* without any additional information, then the displacement strategy used for server and client might displace some nodes of the *existing fragment* and the client's processing might fail. To avoid this, the server applies the *delta fragment* first to the server's *cache set,* monitoring if any node of the *existing fragment* is effected. If an *existing fragment* node would be displaced, then the server overrides this by a fix operation on this node. All fixed node IDs are collected and assembled to a temporary fixed node list representing as a fixed-fragment. This temporary fixed fragment is sent to the client in the *delta update* response. The client now can follow the server's steps by first fixing the same nodes and thereafter applying the *delta fragment* to the cache. After that, the client automatically unfixes the temporary fixed fragment (as the server does after sending the temporary lock fragment) and client and server have the same state in their caches again.
Note that the delta refresh scenario not only covers the situation that a previous query of the same transaction left actual data in the client's cache, but also that data from a query of a previous transaction left valid data in the client's cache.

If the client requests to read a fragment of the XML document and none of the fragment's nodes is available in the clients cache, then obviously the complete fragment has to be transferred. Note that only in this special case the cache is no advantage or disadvantage w.r.t. the attempted reduction of client-server communication.

4.4 Treatment of Large Results (the Overflow Scenario)

Whenever the result to a single XPath query of the client will not fit into the client's cache, there are several alternative ways to iterate over the query result.
A first approach is, that the server follows a heuristics to fill the client's cache with data of the queried XML fragment that is expected to be used by the client's application first. If the client application wants to access data of the requested fragment that is not available in the cache, then the $MDOM_{tr}$ API transparently loads it on demand (indirect load request). Such a heuristics can be driven by parameters provided by the client, e.g. a preferred access sequence (e.g. depth first traversal) and a preferred fragment size to be transferred at once. This allows an application to *implicitly* iterate over a fragment of any size without implementing any additional code that takes care of such a situation. The price for this advantage is that reload and displacement volume depends on whether or not the displacement and overflow heuristics are appropriate for the given application.
In the special case where it is known at design time of the client application which amount of data a single client operation will access, a second approach is to implement an *explicit* iterator. Explicit iterators are stored at the server side and optimize data access for this query, such that the client application only has to pass a reference to the appropriate iterator with the query. Notice that the protocol is still able to fall

back to indirect iteration (on each iteration fragment), if the client's cache size is too small for a whole explicitly defined iteration fragment. If more than one fragment has to be iterated, the available space in the client's cache is computed and the server sends as many fragments as the client can hold.
After such a large iteration has been processed, the client cache will be used by the protocol and chosen replacement techniques as before. Since typical read operations in our applications currently use an XML fragment that is considerably smaller than the client's cache, we consider the need for the management of large results to occur not so often.

4.5 Advantages of Our Memory Management

The main goal of our approach is to allow for transactions on small mobile devices to access a server-side large XML document over a low bandwidth connection. Our approach reduces (the costs for) client-server communication in several ways. First, read operations are performed only locally (zero refresh scenario) whenever the client knows that it has read all the required data before and within the same transaction. Second, the delta update scenario reduces the size of returned fragments. Third, the ability of the client to use an XPath expression and to retrieve a larger XML fragment at once reduces the number of further client requests. Forth, the reuse of old transaction's data still kept in the client's cache instead of reloading the data is an additional advantage. Finally, since in the *zero refresh scenario* the protocol needs no server interaction, the client has the opportunity to work offline for a long time.

5 Transaction Management

5.1 Integration of Our Approach with Validation

As stated before, each client access to the XML document must be done in the context of a transaction. Like most transaction synchronization protocols for mobile clients [5,16], we use an optimistic approach to transaction synchronization on the server-side, because it is not acceptable to block XML fragments for a mobile client that may loose the connection to the server. Since our approach to optimistic transaction synchronization is described in [3], we do not outline it here, but instead mention the following advantages.
Our approach to validation needs no extra communication between the client and the server - except that the client signals that it wants to commit a transaction, and the server informs the client about the commit status of this transaction. The reason is that for direct read or delete operations, our validation protocol [3] uses exactly the same XPath expression as communicated from the client to the server, i.e. validation applies them to modified XML fragments of older concurrent transactions. Furthermore, indirect read or write access can be translated on the server side to the corresponding XPath expression, i.e. this also needs no extra communication between client and server for the purpose of validation.
Note that this communication optimization for the purpose of validation is compatible with the reuse of data that was loaded by old transactions into the client's cache for

the following reason. Before old data in the client's cache can be reused for a new XPath query, the client asks the server whether or not the data is still up to date (delta refresh scenario), i.e. submits the XPath expression to the server. This XPath expression is used in the validation phase, i.e. validation has to consider only the actual transaction's XPath expressions (even if data exchange is reduced to those XML fragments that have changed from older transactions to the actual transaction).

5.2 Offline Work and Lost Connections

Whenever the client uses an XPath expression that covers all the data it needs in a transaction and the corresponding XML fragment fits into the client's cache, then the client needs no further communication with the server (zero refresh scenario) until the end of the transaction, i.e. it could work offline. Furthermore, when the client-server connection gets lost during transaction execution, the client can continue transaction execution as long as it has all needed data in its local cache. Thereafter, it has to wait until the connection is reestablished, and can continue its transaction. Finally, if the connection gets lost during a data exchange step between client and server, client and server can recover by repeating the data exchange operation during which the connection got lost.

5.3 Further Advantages of Our Approach after a Failing Validation

If validation fails, the server sends an abort response to the client. After such an abort, the client can decide to redo the whole transaction. Especially for the frequent case of personal transactions that work only on a small (individually different) fragment of the XML document that fits in the client's cache, we expect that most of the client's cache will contain data that is still up to date. In such a case, submitting the same query as in the previous transaction will require that only a small part of the client's cache has to be refreshed by the server, i.e. again we can reduce the data transfer between client and server. The same advantage holds, if a transaction has to be restarted, because it lost a connection.

6 Summary and Conclusions

A key requirement for transactions on small mobile devices with a low bandwidth server connection is to reduce (the costs for) data exchange and communication between client and server. Our approach contributes to this goal in two ways, i.e. by reusing cache data instead of reloading cache data, and by reducing the communication steps needed in order to check whether client and server cache are up to date.
In order to reuse the client's cache data, we keep a server-side cache copy, that is used to compute those (smaller) delta fragments that are needed to update the client's cache. We transfer only the delta fragments and avoid to reload existing fragments of the client's cache.

In order to reduce communication steps between client and server, we have contributed the following ideas. First, using XPath expressions to read (or to delete) a

larger fragment at once needs fewer data request and data transfer steps between client and server. Second, we use an identical replacement strategy in the client's and the server's cache that avoids extra communication steps to synchronize both caches, i.e. all information needed to synchronize the caches can be taken from the read or write operations that the client performs on the server-side XML document. Third, the zero refresh scenario avoids a server call, whenever the client finds all the required data in its cache and knows that the data has been read from the server (or written by a client operation) within the same transaction. Forth, transaction synchronization (in our case implemented by validation) needs no extra communication between client and server (except that the client asks the server to complete the current transaction and that the server responds with the commit status). Fifth, even in the case of a failed validation, our approach may profit from the data of the failed transaction that is kept in the client's cache, whenever most of this data is still up to date and only a small delta refresh fragment is needed. Finally, our approach integrates well with lost connections.

We consider the presented approach to be a significant step towards new application areas which use XML on small mobile devices. Further research on optimized heuristics for both, replacement and prefetching of XML fragments, with a focus either on small bandwidth connections or on minimized connect times, may be a challenging direction to extend our approach. We are currently implementing a prototype to prove our protocol and to investigate the efficiency of different strategies for different application behavior.

References

[1] Bobineau, C., Bouganim, L., Pucheral, P., Valduriez, P.: PicoDBMS: Scaling down Database Techniques for the Smartcard. Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

[2] Böttcher, S., Türling, A.: Transaction Synchronization for XML Data in Client-Server Web Applications. Informatik 2001, Jahrestagung der GI, Wien, 2001.

[3] Böttcher, S., Türling, A.: Optimized XML Data Exchange for Mobile Concurrent Transactions. Workshop MDBIS, Jahrestagung der GI, Dortmund, 2002.

[4] Dar, S., Franklin, M., Jonsson, B., Srivastava, D., Tan, M.: Semantic data caching and replacement. In Proc. 22nd VLDB, Bombay, 1996.

[5] Ding Z., Meng, X., Wang, S.: O2PC-MT: A Novel Optimistic Two-Phase Commit Protocol for Mobile Transactions. DEXA 2001: 846-856

[6] Doherty, M., Hull, R., Rupawalla, M.: Structures for manipulating proposed updates in object-oriented databases. In SIGMOD 1996.

[7] Franklin, M., Jonsson, B., Kossmann, D.: Performance tradeoffs for client-server query processing. In *Proceedings of the ACM-SIGMOD Conference on Management of Data* (Montreal, Que., June). ACM, New York, NY, 1996.

[8] Goldman, R., McHugh, J., Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Proc. of the 2nd Int. Workshop on the Web and Databases (WebDB), Philadelphia, June, 1999

[9] Graefe, G.: The New Database Imperatives. *Int. Conf. on Data Engineering (ICDE)*, 1998.

[10] IBM Corporation. *DB2 Everywhere – Administration and Application Programming Guide*. IBM Software Documentation, SC26-9675-00, 1999.

[11] Kanne, C.-C., Moerkotte, G.: Efficient Storage of XML Data. Proc. Of the 16th Int. Conf. On Data Engineering (ICDE), San Diego, March, 2000
[12] Kossmann, D., Franklin, M.J., Drasch, G.: Cache Investment: Integrating Query Optimization and Distributed Data Placement. ACM ToDS, Vol. 25, No. 4, Dec. 2000.
[13] Li, Q., Moon, B.: Indexing and Querying XML Data for Regular Expressions. Proc. of the 27th VDLB, Roma, 2001.
[14] Marian, A., Abiteboul, S., Mignet, L.: Change-Centric Management of Versions in an XML Warehouse. Proc. of the 27th VDLB, Roma, 2001.
[15] Oracle Corporation. *Oracle 8i Lite – Oracle Lite SQL Reference*. Oracle Documentation, A73270-01, 1999.
[16] Rasheed, A., Zaslavsky, A.B.: A Transaction Model to Support Disconnected Operations in a Mobile Computing Environment. OOIS 1997: 120–130
[17] Schöning, H., Wäsch, J.: Tamino – An Internet Database System. Proc. of the 7 th Int. Conf. on Extending Database Technology (EDBT), Springer, LNCS 1777, Konstanz, March, 2000
[18] Sybase Inc. *Sybase Adaptive Server Anywhere Reference*. CT75KNA, 1999.
[19] W3C: Document Object Model (DOM) Level 1 Specification. http://www.w3c.org/TR/1998/REC-DOM-Level-1-19981001/
[20] W3C: XML Path Language (XPath) Version 1.0 . http://www.w3.org/TR/xpath
[21] Yeo, L.H., Zaslavsky, A.B.: Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. ICDCS 1994: 372–379

Support for Mobile Location-Aware Applications in MAGNET

Patty Kostkova[1] and Julie McCann[2]

[1] Department of Information Science, Institute of Health Sciences,
City University, London, UK
patty@soi.city.ac.uk
[2] Department of Computing, Imperial College London, UK
jamm@doc.ic.ac.uk

Abstract. A key characteristic of mobile applications is the need for up-to-date location-dependent information, while physical locations change frequently. Recent improvements in wireless communication and hardware technology and Internet-based data exchange has created a new type of mobile application whose requirements are not met by traditional relational and object database systems. In this paper we describe MAGNET, a tuplespace-based framework for dynamic information storage and retrieval, addressing the needs of applications in frequently changing mobile environments, and discuss how this approach could enable flexible data exchange. In addition to type-free data storage and user-customized requests for data records, MAGNET enables adaptation to changing environments by supporting constant monitoring of selected information.

1 Introduction

In the business climate, an increasing number of people are expected to perform complex work-related tasks while on the move. Change in physical location results in the volatility of location-dependent information (e.g., the local time, the nearest library). Therefore, a key requirement of mobile users[1] is the retrieval of dynamically updated location-dependent information. The need for database support for *mobile applications*[2] has become important since frequent travelling has become commonplace. Key advances in hardware technologies allowing the current boom in mobile computing include: improvements in reliability, speed and coverage of wireless communication, decreasing hardware size, the invention of the colour LCD display, the track-ball and the touch-pad, and, the rapidly decreasing size and weight of mobile phones. The timely combination of these achievements has enabled the widespread of PDAs designed for specific mobile applications running specialized

[1] By 'mobile users' we mean weakly-connected (wireless) users frequently changing their location (e.g. ,taxi-drivers, tourists) typically using portable computers or PDAs.

[2] A 'mobile application' we define as a distributed application run by mobile users (e.g., users with portables working while in transit, tourists while sightseeing, taxi-drivers etc. dealing with location-dependent information (e.g., a local resource, local 'data', a location-based request) in a changing environment.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 280–294, 2003.

software. Also, the Internet phenomenon has become ubiquitous as it enables format-free data storage, retrieval, search and dynamic exchange without restrictive data modelling and with the freedom to query its content by looser means than relational algebra could offer.

That is, as a result of the combination of the availability of the Internet and the affordability of wireless communications, new types of applications have emerged requiring new types of database support. Traditional database applications are still commonplace at enterprises and organizations where there is a need for relational data modelling, SQL-processing, traditional strong consistency and two-phase transactions, however, there is also a need for support of the new type of applications which find traditional relational and object databases too restrictive.

In this paper, we focus on the problem of retrieving dynamically updated location-aware information, rather than "classical" mobile problems dealing with the fluctuation in quality of a wireless communication network, or change in the degree of connectivity. This paper describes an information exchange model, MAGNET, which supports dynamically updated information among mobile users who frequently change location. MAGNET is investigating a different approach to information storage and retrieval, and is not based on a traditional database framework. It is based on a shared information pool permitting operations for data insertion and their user-customized location-aware withdrawal – that is a query. In addition, it supports the monitoring of information placed in the pool, and user-defined adaptations to changes in the environment.

In the next section, we discuss our motivations, and define the requirements for a dynamic information-sharing infrastructure. Section 3 presents an overview of the MAGNET model. Section 4 describes the support for information monitoring in greater depth and section 5 demonstrates the use of MAGNET using an example of a taxi navigation system. Section 6 summarizes work relevant to MAGNET; section 7 discusses the project's current status and directions for future research. Finally, section 8 contains concluding remarks.

2 Motivations

In this section, we start with summarizing current problems attributable to traditional database systems, and then give a brief outline of the mobile environment typical for the applications requesting up-to-date location-dependent data. Then, we discuss characteristics of this class of applications, and elaborate on the requirements for database support.

2.1 Shortcomings of Traditional DBMS

DBMS have matured to become large, expensive and lumbering pieces of systems software. Consequently, as closed systems, a DBMS not only has sole ownership of the data, but data access is restricted to the DBMS through either query-languages or programming interfaces. Below we list a set of DBMS shortcomings, which were mostly highlighted by participants of a previous ICDE conference [1]:

- multi-media and its content searching cannot be carried out by DBMS
- information retrieval techniques are not incorporated into current DBMS systems
- once a query is issued the user cannot make changes during processing
- self administration and self tuning are currently unfeasible
- data structures for mining need to be evaluated
- improved data structures for model migration are needed
- extensibility and reconfigurability could then provide e.g. 24 x 7 runtime support.

This combination of *shoehorning* and *wrapping-up* of add-on services reduces not only performance [2] but impairs core DBMS flexibility.

2.2 Characteristics of Location-Aware Applications

Typical location-aware applications include: mobile portable users requiring local resources in different offices, tourists running guide-like information software on PDAs, or taxi-drivers using PDAs to navigate to the next destination.

For mobile applications requiring dynamically updated location and time-dependent information, the crucial problem is the absence of service support for information sharing and run-time update. These applications, cannot be satisfied by traditional database engines, as currently these offer a too restrictive format, that is, relational algebra-based queries are too rigid, typically not providing any means for location and time based awareness and adaptation.

Owing to recent significant improvements in wireless communication, weakly connected applications no longer suffer from unreliability of the communication infrastructure (in terms of higher error-rate, frequent disconnections, or limited coverage). In addition, for some mobile users it is essential to be provided with local information. The high volatility of the local information encountered by mobile users' moving location necessitates specialised database support which is able to be tailored to their new requirements and needs as they change. The primary role of the database support should be enabling type-free information to be stored and exchanged, and different type of queries to be supported, which allow wider user-customized location and time aware searching.

2.3 Tuplespace-Based Database

The primary role of the database in this type of application is to enable the sharing of dynamic information (both local and location-independent). We term such an information-sharing infrastructure an *information pool*. That is, in order to avoid the confusion with the term database which is commonly used for a relational or object databases. The information pool, described above, holds data items termed *tuples* (i.e., structured records) which express information to be requested by mobile users. To achieve full generality, the information pool should not constrain the format or semantics of tuples it contains. This permits virtually unrestricted extensibility of existing services, data formats, in order to adapt application behaviour to changes in environment, or dynamically extend system functionality.

To query the pool, we use a mechanism we term tuple *matching*. Again, as the matching process semantically differs from traditional database queries, we would stick to the term matching to avoid confusion. In order to achieve generality, user-customisation of the requests for data stored in the pool is required. An addition, to distinguish our framework from other systems, we use the term binding to refer to the result of the tuple match (1:1 relation, by default). In other words, when the match function has found a tuple satisfying the request, a *binding* between the two components which inserted the matching tuples is established. To satisfy the crucial requirement of mobile applications – dynamic information exchange and update, a mechanism for automated monitoring is required in order to announce updates of monitored information provided by components themselves. Finally, updated information is fully utilized if the system can dynamically adapt to a new environment. This process is called *rebinding*.

3 Overview of the MAGNET Architecture

MAGNET is a high-level framework enabling applications in mobile frequently changing environments to store, update and query location-depended information. The full description of the architecture and its usage in dynamic resource management and other application areas could be found in [3]. Ideally MAGNET would be placed within a component-based systems architecture. This architecture would also reconfigure on demand and this too can be controlled by MAGNET (i.e., components can be activated and bound to other components at runtime). However in this paper we focus on information mapping mainly.

The key component of the framework is a *Trader* that collects information on services, records and all application data and dynamically matches requests against demands, in other words, performs user-customized search. One of the key features of the tuplespace is not to constrain the format or the semantics of stored information to allow type-free dynamically defined data to be stored and searched by a user-customized matching process. This provides *extensibility* in terms of enabling new records, service requests and actual services to be dynamically generated but also in terms of customizing the matching process itself. Further, to support runtime adaptation and system reconfiguration *dynamic rebinding* is required. That is, the old binding is dropped and a new binding is established in order to better meet application requirements. This may be as a result of client, server or a third party initiation. For example, a mobile client currently using its local disk may wish to join a new, more stable environment in an office to upload data. Therefore it will unbind from its current disk and rebind to the office disk. Information on client demands and service capabilities is maintained either manually (i.e., carried out by the components themselves) or automatically (by a monitoring process). The stateless nature of tuples saves the pool from having to provide a state-maintaining scheme, for example, check-pointing or recovery procedures. In addition, it improves the generality and reliability of the system. If state is required, it can be incorporated as a parameter of tuples. Finally, decoupling the server from the client (servers produce tuples of interest to any client) permits communication to proceed anonymously.

3.1 The Trader

The Trader is the key component in the MAGNET architecture. The Trader accesses a shared data repository available to all applications and objects represented by components. We call this data structure an information pool, its structure is similar to the tuplespace[3]. The Trader consists of three distinctive elements:

- The information pool (a tuplespace-like data structure),
- The Trader operations on tuples for their manipulation, and
- The tuple matching function (an operation providing the actual querying or communication).

Fig. 1 illustrates the structure of the Trader, and its three components. Darwin, an architecture-description language [4], provides a convenient formalism for defining bindings in distributed systems.

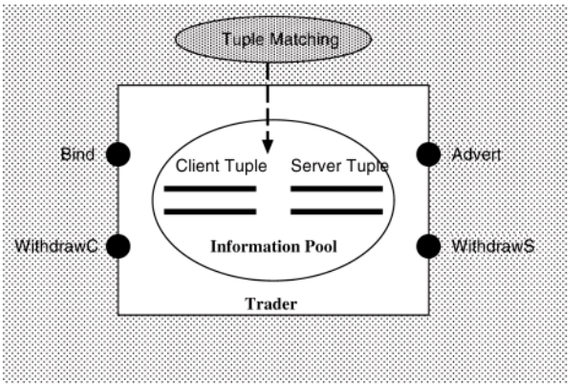


Fig. 1. The Trader Structure

3.2 The Information Pool and the Matching Function

The information pool is a distributed data structure accessible by all components using MAGNET. Tuples can be inserted in, or withdrawn from, the tuplespace by a set of clearly defined operations. Tuples describing mobile component data often contain additional information, such as interface references for accessing the component. These are all expressed as tuple elements. Therefore, the tuple distinguishes between the number of all tuple elements n and the number of matching elements m. This extension, which we have incorporated into traditional tuple matching, enables the restriction of the matching process to matching the first m elements.

---

[3] The information pool is actually a tuplespace. However, the term "tuplespace" is often associated with the Linda distributed programming language [13]; therefore, we decided to call our data structure `information pool' to avoid confusion.

We define a tuple as follows:

A **tuple T** is an ordered set of (n+2) elements $T=(n, m, p_1, p_2, \ldots,p_n), n>m$ where n represents the number of tuple elements and m is the number of "matchable" tuple elements pi are the values of tuple elements i.e., the actual parameters.

For example, to describe a component book Romeo and Juliet by William Shakespeare we may use the following server tuple:

A = (6, 5, 12345, William, Shakespeare, Romeo and Juliet, Penguin, ISBN 654321)

That is 6 tuple elements, 5 of which can be matched: 12345 (Author ID), William, Shakespeare, Romeo and Juliet, Penguin (publisher). ISBN is a reference to the book (service) described by this tuple. Naming for interface references is derived from the naming scheme used in the computing or application environment, e.g., ISBN, library identification.

An equivalent client tuple looking up Romeo and Juliet would be:

*B = (6, 5, *, William, Shakespeare, Romeo and Juliet, *, reader ID)*

requesting this book published by any publisher (* sign) and ignoring the Author ID. This tuple definition incorporates advanced operators, such as *. These are defined in details in [3].

3.3 The Matching Function

By matching, querying the data structure, as was explained above, we mean an equality of tuple elements, or a user-defined "match" enabling quality of service to be taken into account. (However, this is beyond the scope of this paper, further details can be found in [3]).

A client tuple $T_1 = (n_1, m_1, p_1, p_2, \ldots,p_n), n_1 > m_1$ and a server tuple $T_2 = (n_2, m_2, q_1, q_2, \ldots,q_n), n_2 > m_2$ match iff $m_1 = m_2$ *and* $(p_i = q_i)$ *for all* $i \in \{1, m_1\}$.

As incorporating non-matching values into tuples is optional, and may differ between a client and a server-tuple, the equality of tuple size ($n_1 = n_2$) is not a required matching condition.

The key focus of the architecture is to provide flexible dynamic matching of format-free records for mobile applications. Above all, the user-customized matching function enabling an extra flexibility and framework extensibility is a powerful mechanism needed in dynamic frequently changing environments.

3.4 The Trader Operation

A set of predefined operations exist to manipulate tuple data, e.g., insert and delete. MAGNET's Trader includes the operations: *Bind*, *Advert*, *WithdrawC*, and *WithdrawS*. These are described below in more detail.

Operation **Bind (T)**, T is a client-tuple. The Trader searches the information pool for a complementary matching tuple. If such a tuple is found, T is returned to the server component (which inserted the matching tuple) without being withdrawn from the pool. If no such tuple exists, the operation results in inserting tuple T into the information pool until a match becomes available and the request is fulfilled.

Operation **Advert (T)**, T is a server-tuple, which is inserted into the information pool. The trader also searches the pool for all complementary matching tuples. If such tuples are found, they are removed from the pool, and returned to the calling server component.

Operation **WithdrawC (T),** where T is a client-tuple, results in removing tuple T from the information pool; while operation **WithdrawS (T),** where T is a server-tuple, results in removing tuple T from the information pool.

3.5 Components for the MAGNET Architecture

Fig. 2. illustrates the structure of the MAGNET architecture. The system consists of four classes of component: *the Trader, Client, Server* and *Tree* (components performing the matching process). There is only a single instance of the Trader component per physical computing node, in contrast to multiple instances of Client, Server and Tree. In addition there are two types of subcomponent performing dedicated functions: these are a pair of Binders (the *Client-Binder* and the *Server-Binder*) present in all Clients and Servers; and the *GlueFactory* included in all Trees. The GlueFactory hands over a client tuple to the Server to initialise the establishment of the binding carried out by the Binders. Therefore, Binders in cooperation with the GlueFactory establish the resultant client-server binding. In order to provide scalability of the framework, it could be distributed into federations; more details could be found in [3, 5, 6].

4 Support for Information Monitoring

In order to enable adaptation to changes in system characteristics, service definitions, which are placed in the Trader, must be kept up-to-date. Therefore, MAGNET must monitor resource characteristics. For this reason, the framework presented in this paper is equipped with two additional components providing monitoring: *the Monitor* (monitoring server provisions), and *the Updater* (monitoring changing client requirements).

4.1 Components for Monitoring

As the MAGNET framework distinguishes between the roles of the client and server, it is necessary to approach their monitoring differently. Therefore, MAGNET has two monitoring components providing this functionality – the Monitor and the Updater –

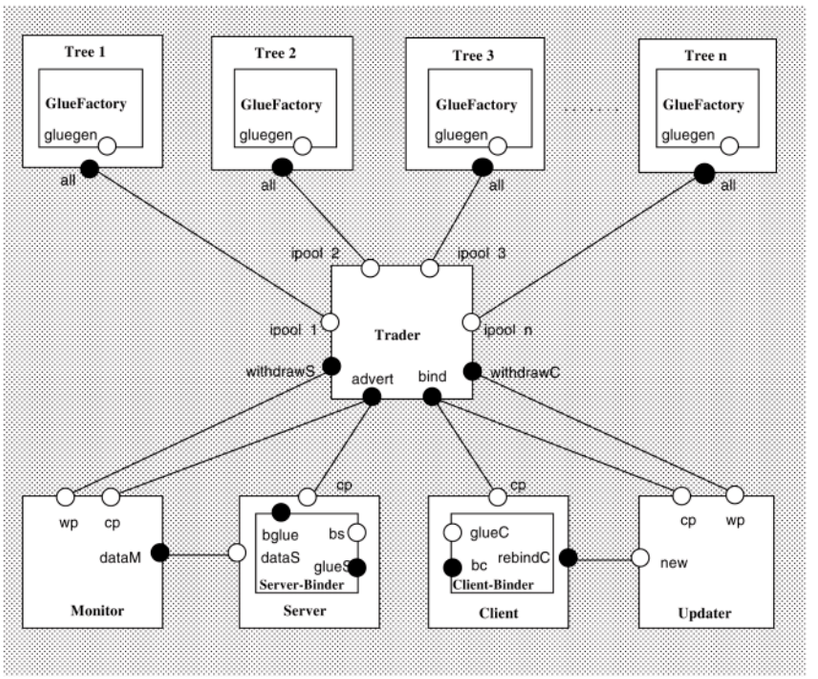


Fig. 2. The Architecture with the Monitor and the Updater

both these application-level components are attached to server or client respectively. They are created together with the components they serve, and are instructed by them to provide component-tailored functionality. Here we discuss their interface to MAGNET and expected functionality. The components are illustrated in Fig. 2.

4.1.1 The Monitor and the Updater

The task of the Monitor component is to observe changing characteristics of the server it is attached to, and keep the server tuple up-to-date. Tight cooperation with the server enables the Monitor to be informed about current service characteristics, so that it can periodically update relevant tuples in the pool (by removing them and replacing with updated ones). As there are not many clients requiring rebinding after having found a requested service, the monitoring of client requirements is less crucial. Also, client-tuples do not reside in the pool (once a match was found), and therefore there is no need to keep them up-to-date.

However, clients in systems with frequently changing characteristics may rely on a guaranteed level of service (e.g., network throughput). For those, adaptation to change in conditions is unavoidable (e.g., switching to lower-quality audio and video, etc.) The Updater is a dedicated component instructed by the client it is attached to. It searches the pool for a tuple meeting the client's current requirements more precisely, or looks for a different tuple if the client's requirements have changed (e.g., mobile users on the move need to update a requirement for the nearest server, etc.)

4.2 Monitoring Server Provisions and Client Requirements

In this section we describe the monitoring process, as provided by the dedicated components: the server-attached Monitor, and the client-attached Updater. Server-Monitor and Client-Updater interactions are established statically in advance by a system administrator, not using MAGNET.
The Monitor component is attached to the server by a binding established between service interfaces *dataS* and *dataM*. The server keeps the Monitor informed about relevant changes. Then, according to the granularity of update (how often it is performed), and the 'out-of-dateness' accepted (how much can a tuple in the pool differ from current characteristics), the Monitor decides when to perform the operations *WithdrawS* and *Advert*. That is, the actual update in the pool (through service interfaces *cp* and *wp*). From the Trader's point of view, monitoring is performed transparently, indistinguishable from a sequence of operations *WithdrawS* and *Advert* performed by the server itself.
The Updater component is instructed by a client about service requirements it should search for. These two components communicate through a statically established binding between service interfaces *new* and *rebindC*. In this case, the initiative is on the Updater component, in contrast to the Monitor that acts only when invoked by the server. The Updater calls the operation *Bind* on a tuple with higher requirements (through service interface cp), or performs *WithdrawC* and *Bind* operations when the requirements of the client have changed. The bind-tuple, inserted by the Updater, waits in the pool until it finds a match. According to the Updater protocol and the 'stage' of client interaction, the Updater decides if rebinding is beneficial (rebinding a client close to finishing might not be beneficial, when taking the overhead of the rebinding process into account). Therefore, the new server tuple can be ignored, or client rebinding can be performed.

4.3 Efficiency: Discussion

For applications requiring only course-grained monitoring strategies (with frequency of minutes), tuple updates performed by a withdrawal and reinsert (as discussed in this section) are sufficient. However, for applications requiring finer-grained updates of their data in the pool (with frequency of seconds and milliseconds), the complexity of the Trader operations must be added to the complexity of the update operation. In order to improve efficiency, specific trusted Monitors and Updaters might be authorized to have direct access to the Tree holding their tuples. However, this solution fundamentally violates protection of the information pool (encapsulating Trees behind the public Trader's operations). For the reason of protection of other data in the pool, and protection of Trees that might be misused by untrustworthy Monitors, this approach is not a part of the framework design.

5 Example: The Taxi Navigation System

In this section, we illustrate the use of MAGNET on a typical mobile application – a taxi navigation system. This simplistic example illustrates MAGNET's functionality in terms of the applications' dependency on dynamically updated location-dependent information, requiring monitoring and support for adaptations to the changed situations.

The taxi navigation system consists of a number of taxis that are characterized by their location [X,Y], and passengers, characterized by their location – place where they are waiting for a taxi. Passengers can hail a taxi on the street, call for it by phone, or wait for it at a city taxi-rank. After a taxi drops off a passenger at the destination, it returns to the nearest taxi-rank, if not directed to another pick-up location. If there are no waiting passengers, taxis remain at the taxi-ranks. We can imagine that information about the length of the queue and passengers waiting at the taxi-rank is provided by a camera placed above the taxi-rank recording the queue and updating the information pool accordingly. All taxis are equipped with small PDAs and GPSs and are directed to their destination by a special component called the *Navigator* which transmits directions to the destination. The Navigator to select the best route at runtime, according to the city plan, and in response to dynamic changes, such as traffic jams, road-works, road closures due to accidents, etc. Fig. 3 illustrates the situation.

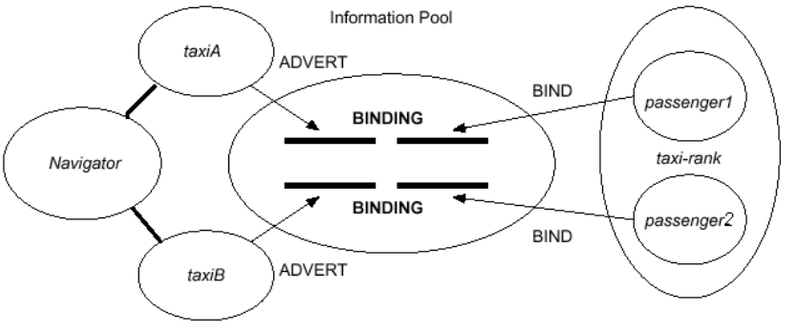


Fig. 3. Components in the Dynamic Taxi Navigation System

5.1 MAGNET Support for Dynamic Taxi Allocation

We will describe MAGNET functionality on a simple scenario. MAGNET is used to find the best matches between available taxis (expressed by placing their offer into the pool) and waiting passengers (also expressed by placing their request for a taxi into the pool). Also available taxis have tuples in the pool, one a customer is picked up, the tuple is withdrawn, as the taxi is no longer available.

1. **Taxi A and Taxi B available**

We start with a situation when there are two taxis (taxiA and taxiB) in the example, both currently available, being navigated by the Navigator to the taxi-rank. In MAGNET, the situation is described by two server tuples representing the taxis inserted into the pool by operation *Advert*:

TAXIA = (3,2,X1,Y1,TA)
TAXIB = (3,2,X2,Y2,TB)

Where the [X1, Y1] and [X2, Y2] are the coordinates of current location of the taxis. The Monitor components attached to both taxis ensure the tuples are updated in requested intervals, e.g., 5 minutes. TA and TB are references to the taxis, e.g., a direct contact to their Navigator components, or phone numbers of their drivers etc.

2. Passenger 1 arrives

Then, Passenger 1 arrives to the taxi-rank and waits for a taxi. The camera at the taxi-rank records the client and inserts a client tuple using operation *Bind*:

P1 = (3,2,X3,Y3,P1)

Where [X3, Y3] is the location of Passenger 1 – the coordinates of the taxi-rank and P1 is the identification of the passenger. As soon as the operation *Bind(P1)* is called the matching function finds the best available match (the closest taxi) and allocates the passenger to that taxi (lets Taxi A is the closer one). Then, the P1 tuple is automatically withdrawn (as this is the definition of the operation), and tuple TAXIA is manually withdrawn by calling *WithdrawS* operation as a taxi usually cannot drive more than one passenger.

3. Taxi hailed on the Street by Passenger 2

TAXIB, still 2 on its way to the taxi-rank, is hailed on the street by Passenger 2 – this 'binding' takes place without MAGNET to illustrate that in open systems components can establish binging also without the assistance of traders. Technically, this client did not insert any tuple into the pool, simply hailed a passing taxi. Consequently, the *WithdrawS (TAXIB)* operation is called by the TAXIB Monitor to reflect the change. Therefore, there are no tuples in the pool at the moment.

4. Passenger 3 calls for a taxi by phone

A passenger 3 calls for a taxi by phone, a client tuple P3 = (3,2,X4,Y4,P4) is inserted into the pool by operation *Bind* (we may imagine the automated phone operator calls the function). The tuple remains waiting in the pool as there is no taxi available. The, TAXI A drops off Passenger 1, the relevant Monitor reinserts the server tuple into the pool by operation *Advert (TAXIA).* Then, a match is achieved between TAXIA and Passenger 3 resulting in directing TAXIA to Passenger 3 location.

5.2 User-Customized Matching Function

The key feature enabling this dynamic taxi-passenger matching is the customized matching function. In this example, we have assumed their [X, Y] coordinates indicate locations of taxis and passengers. As for taxis, these are recorded by GPSs attached to taxi Monitors, however, for clients they need to be calculated from street names. The user-customized matching function selects the closest client for each taxi

(or vice versa). For example, there is a taxi tuple TAXI = (3,2,X1,Y1,T) and N waiting clients P_n = (3, 2, X_n2 ,Y_n2, P_n) in the pool. Then, the matching function finds the minimum distance (the closest client to the taxi):

TAXI matches P_i *iff Min* $i_{i \in N}$ ($|X1 - X_i2| + |Y1 - Y_i2|$)

This calculation assumes grid street topology. We have adopted this approach to illustrate the notion of user defined matching, however, in many US cities this distance definition would be perfectly appropriate. However, the function could be further extended to allow more complex distance definitions, based on real street maps, and include dynamic changes, such as traffic jams and road closures. However, further investigation of these issues is beyond the scope of this paper.

5.3 Dynamic Adaptation

A passenger waiting for a taxi at a taxi-rank, is a classical situation, not requiring adaptation, assuming there is only one taxi-rank in our example spares the system solving the problem of redirecting taxis from one taxi-rank to another in response to varying lengths of queues. Typical adaptation-requiring situations include a taxi being hailed in a street when this was to pick up a customer at the taxi-rank, or a passenger phoning for a taxi (according to the taxi system priority policy, closest taxi could be allocated to calling client despite the fact the taxi was going to pick up a different customer living further, etc.)

5.4 Discussion

For additional flexibility, the tuplespace structure as defined in MAGNET can allow the application to model time-constrained operations (e.g., a passenger urgently needs a taxi to get to the airport; but, if the taxi does not arrive within ten minutes, the plane will be missed, therefore, the passenger is not interested after ten minutes). Clients can choose how long they are willing to wait until their request is accepted. The time scale extends from zero (if the requested tuple is not available at that moment, an error status is returned to the client), through arbitrary time-out intervals (the tuple is waiting in the pool until the requested tuple is inserted or until the timeout expires), to unlimited waiting (the tuple persists in the pool forever if the required complementary tuple has never been inserted). In order to incorporate a “timeout” feature, the client can withdraw the tuple when he is no longer interested, or this could be provided automatically by an *Updater* component where the client sets a predefined threshold. This functionality can also be incorporated into the user-defined matching functions, if appropriate (e.g., in the previous example of a passenger travelling to the airport; the matching function can incorporate more flexible adjustment of the time interval according to the current traffic situation). Finally, the system could record the destination of clients in terms of extra tuple elements and their willingness to share a taxi with other clients. Then, the matching function could optimise taxi allocation so as clients travelling to destinations near each other could share a taxi, which would result in more efficient transport.

6 Related Work

There has been some work on adaptive query processing. Examples of this work are pipelined hash join [7], and the Xjoin [8]. Most of this work is with relational data and concerns aggregation queries as examples [9], however some have looked at XML [10]. Contemporary research in mobile computing has explored problems with mobility and the unreliability of wireless communication networks [11]. Fluctuations in quality of service (QoS) and changing degrees of connectivity have also been studied [12].

As for trading architectures, Linda was the first system to support a generative communication model [13], providing several important features, but its fixed tuple format and semantics do not provide the flexibility required by mobile applications. Also, JavaSpaces provide a Tuplespace-like distributed environment manipulating objects rather than data tuples. This enables global scalability and forms a base for the Jini technology.

Blair et al [14] investigated the tuplespace approach to QoS support in a mobile environment. It extends the traditional tuplespace with QoS management providing support for monitoring and adaptation for applications using heterogeneous networking environments. More examples of dynamic resource reconfiguration are discussed in [3,5,6].

7 Current Status and Further Work

In this section, we summarize our assumptions, outline our implementation experience and discuss further work.

7.1 Assumptions

Here we summarize the assumptions we used when designing the MAGNET system, and discuss their implications and possible solutions. We assume all system components maintain their own consistency. That is, we assume that rebinding can be performed only when the system is in a safe state and that when a component has finished its operation it must leave MAGNET in a consistent state. A related situation is where old tuples remain in the information pool. A periodic garbage collection routine can purge tuples that are marked as out-of-date by as defined by the originating component.

Further, user defined functions are assumed to be secure in that they return control back to the Trader. To overcome this we would have to extend the trader's functionality to finish any matching function by force after a timeout period. Also, we assume that unambiguous naming schemes are used. However, the former can be derived from common naming schemes, such as IP addresses.

As for performance, the estimated numbers of components in are in the region of tens and they have the potential to generate tens to hundreds tuples placed in the Trader. Likewise, the number of concurrent components accessing the Trader at one

time is estimated to be in the region of tens. A higher number of components can result in the Trader becoming a bottleneck. A possible solution would be to implement the information pool in a distributed shared memory.

Regarding change frequency, the framework is designed for components that will change their features with a frequency of minutes and hours, rather than seconds and milliseconds. Therefore the proposed support for monitoring and rebinding as a result of a change is adequate. The support for applications requiring finer grained updates (with a frequency of seconds and milliseconds) would not be viable.

7.2 Implementation

MAGNET has been implemented in C++ in Regis [15], an environment for constructing distributed systems. The complexity of the Trader operations was calculated and was found to be linear to the number of tuple matching elements. However, critical analysis of various features of the framework can be found in [3]. In addition, the MAGNET architecture also supports advanced QoS support. However, support for QoS is beyond the scope of this paper. Further details of our QoS model, its design and implementation can be found in [3].

7.3 Further Work

Also, another our project Go!, a component-based Operating System which has shown that fine-grained componentisation does not only provide 'lightweightness' and extensibility but also improvements in performance [16]. We believe that Go! is a good starting point for this research as it has already proven that it can improve performance and be lightweight, but combined with MAGNET we should be able to show its true power. To do this we are currently expanding the operating system, and extending our work on models to describe hardware abstractions, components and their interaction and resource management.

8 Conclusion

This paper has targeted a fundamental problem of mobile users requiring dynamically updated location-aware information. We have argued that the problem has become crucial, owing to a combination of recent improvements in wireless communication, and advances in hardware technology. As a result of these fundamental changes, there is a new class of applications requiring type-free data storage, frequent updates and modifications and user-defined flexible way to query these data. These applications need both flexibility and generality, and often no longer require the traditional database features, relational data modelling, transactions and security constrains.

As traditional database systems do not provide support for these types of mobile applications, we have investigated a tuplespace-based framework, MAGNET, allowing the searching and trading of information and data records in frequently changing mobile environments. This extends the notion of the tuplespace paradigm to

provide a universal solution, which interestingly is not tied to mobile environments only.

References

1. ICDE, 13th IEEE International Conference on Data Engineering, Birmingham, UK, (1997)
2. Stonebraker M., Brown P.: Objects in Middleware: How bad can it be?, Informix white paper, www.informix.com/informix/whitepapers/howbad/howbad.htm
3. Kostkova P.: MAGNET: Dynamic Resource Management Architecture. PhD Thesis. Dept. of Computing, The City University, London, (1999)
4. Magee J., Dulay N., Eisenbach S., Kramer J.: Specifying Distributed Software Architectures. Fifth European Software Engineering Conference, Barcelona, (1995)
5. Kostkova P., McCann J.A.: MAGNET: An Architecture for Dynamic Resource Allocation. Proc. of Int. Workshop on Data Engineering for Wireless and Mobile Access, ACM, (1999)
6. Kostkova P., McCann J.A.: Inter-federation Communication in the MAGNET Architecture. The Third Grace Hopper Celebration of Women in Computing, Mass. USA., (2000)
7. Wilschut A. N., Apers P..M. G.: 'Dataflow Query Execution in a Parallel Main-Memory Environment. In Proc. First International Conference on Parallel and Distributed Information Systems (PDIS), (1991) 68–77
8. Urhan T., Franklin M.J., Amsaleg L.: Cost-based query scrambling for initial delays. In Proc. ACM SIGMOD Int. Conference on Management of Data, (1998)
9. Hellerstein JM., Haas PJ., Wang HJ.: Online Aggregation, Tech. Paper, IBM (1997)
10. Ives Z G., Levy A Y., Weld D. S., Florescu D., Friedman M. Adaptive Query Processing for Internet Applications. IEEE Data Engineering Bulletin vol 23 no 2, (2000) 19–26
11. Forman G.H, Zahorjan J.: The Challenges of Mobile Computing. IEEE Computer, 27(4), April 1994, 38–47
12. Davies N., Blair G. S., Cheverst K., Friday A.: Supporting Adaptive Services in a Heterogeneous Mobile Environment. The 1st Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA (1994)
13. Gelernter D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1), (1985) 80–112
14. Blair G. S., Davies N., Wade A. P.: Quality of service support in mobile environment: an approach based on tuple spaces. The 5th IFIP International Workshop on Quality of Service, New York, USA (1997)
15. Crane J. S.: Dynamic Binding for Distributed Systems. PhD thesis, Dept. of Computing, Imperial College, London, UK (1997)
16. Law G., McCann J. A.: Decomposition of Pre-emptive Scheduling in the Go! Component-Based Operating System, ACM SIGOPS European Workshop (2000)

The XML Query Language Xcerpt: Design Principles, Examples, and Semantics

François Bry and Sebastian Schaffert

Institute for Computer Science, University of Munich
http://www.pms.informatik.uni-muenchen.de/

Abstract. Most query and transformation languages developed since the mid 90es for XML and semistructured data – e.g. XQuery [1], the precursors of XQuery [2], and XSLT [3] – build upon a path-oriented node selection: A node in a data item is specified in terms of a root-to-node path in the manner of the file selection languages of operating systems. Constructs inspired from the regular expression constructs $*$, $+$, ?, and "wildcards" give rise to a flexible node retrieval from incompletely specified data items.
This paper further introduces into Xcerpt, a query and transformation language further developing an alternative approach to querying XML and semistructured data first introduced with the language UnQL [4]. A metaphor for this approach views queries as patterns, answers as data items matching the queries. Formally, an answer to a query is defined as a simulation [5] of an instance of the query in a data item.

1 Introduction

Essential to semistructured data is the selection of data from incompletely specified data items. For such a data selection, a path language such as XPath [6] is convenient because it provides constructs similar to regular expressions such as $*$, $+$, ?, and "wildcards" that give rise to a flexible node retrieval. For example, the XPath expression `/descendant::a/descendant::b[following-sibling::c]` selects all elements of type `b` followed by a sibling element of type `c` that occur at any depth within an element of type `a`, itself at any depth in the document.

Query and transformation languages developed since the mid 90es for XML [6] and semistructured data – e.g. XQuery [6], the precursors of XQuery, and XSLT [6] – rely upon such a path-oriented selection. They use patterns (also called templates) for expressing how the selected data, expressed by paths, are re-arranged (or re-constructed) into new data items. Thus, such languages intertwine construct parts, i.e. the construction patterns, and query parts, i.e. path selectors.

Example 1. An example for this intertwining of construct and query parts is the following XQuery query from [7]. This query creates a list of book titles for each author in a bibliography database, like that of Example 2.

A.B. Chaudhri et al. (Eds.): Web Databases and Web Services 2002, LNCS 2593, pp. 295–310, 2003.

```
<results>
  {
    for $a in distinct-values(document("http://www.bn.com")//author)
    return
        <result>
            { $a }
            {
                for $b in document("http://www.bn.com")/bib/book
                where some $ba in $b/author satisfies deep-equal($ba,$a)
                return $b/title
            }
        </result>
  }
</results>
```

The XQuery expression is a construct pattern specifying the structure of the data to return. The query parts, i.e. the definition of the values for the variables `$a` and `$b`, are included in the construct pattern. Note that the (path-oriented) definitions of the variables `$a` and `$b` refer to a common subpath `document("http://www.bn.com")`. Note also the rather complicated condition relating values of `$a` and `$b`: `some $ba in $b/author satisfies deep-equal($ba,$a)`.

The same query can be expressed in Xcerpt as shown in Example 9. □

This intertwining of construct and query parts à la XQuery has some advantages: For simple query-construct requests, the approach is rather natural and results in an easily understandable code. However, intertwining construct and query parts also has drawbacks:

1. query-construct requests involving a complex data retrieval might be confusing,
2. unnecessarily complex path selections, e.g. XPath expressions involving both forward and reverse axes, are possible [8],
3. in case of several path selections, the overall structure of the retrieved data items might be difficult to grasp, as in Example 1.

Among the query and transformation languages, UnQL [4] is a noticeable exception. This language first considered using patterns instead of paths for querying semistructured data. UnQL query patterns may contain variables. Applying a kind of pattern matching algorithm, reminding of those pattern matching algorithms used in functional programming and in automated reasoning, to a UnQL query pattern and a (variable-free) data item binds the variables of the query pattern to parts of the data item. This paper further investigates this approach proposing the following new ideas:

1. Instead of pattern matching, a (non-standard form of) unification is considered using which two query patterns, both containing variables, can be made identical through bindings of their variables.

2. Within a query pattern, a variable might be constrained through a (sub-)pattern to be bound only to data conforming to this (sub-)pattern.
3. Instead of building upon the functional paradigm, as UnQL does, the paradigm of SQL and of logic programming is retained. Thus, a query might have several answers and the choice of some or all of the answers specified by a query can be expressed with language constructs reminding of the well-known set operators of elementary mathematics.
4. A chaining of queries, the answers to which are not necessarily sought for, makes it possible to rather naturally split complex queries into intuitive parts.

A metaphor for this approach is to see queries as forms, answers as form fillings yielding database items. With this approach, patterns are used not only in construct expressions, but also for data selection.

In the following, the basic concepts of a query language called Xcerpt are introduced. An answer to a query in this language is formalised as a simulation [5] of a ground instance of the query in a database item. This formalisation yields a compositional semantics.

2 Xcerpt Basic Constructs

This section introduces the essential constructs of the query language Xcerpt. Note that Xcerpt's terms are just "XML in disguise". However, aspects of XML, such as attributes and namespaces, that are irrelevant to this paper, are not explicitly addressed in the following.

Below, the following pairwise disjoint sets of symbols are referred to: A set $\mathcal{I}$ of identifiers, a set $\mathcal{L}$ of labels (or tags or strings), a set $\mathcal{V}_l$ of label variables, a set $\mathcal{V}_t$ of term (or data item) variables. Identifiers are denoted by id, labels (variables, resp.) by lower (upper, resp.) case letters with or without indices. The following meta-variables (with or without indices and/or superscripts) are used: *id* denotes an identifier, l denotes a label, L a label variable, X a term variable, t a term (as defined below), v a label or a term, and V a label or term variable.

2.1 Database Terms

A database is a set (or multiset) of database terms. The children of a document node may be either ordered (as in standard XML), or unordered. In the following, a term whose root is labelled l and has *ordered* (*unordered*, resp.) children $t_1, \ldots, t_n$ is denoted $l[t_1, \ldots, t_n]$ ($l\{t_1, \ldots, t_n\}$, resp.).

Definition 1 (Database Terms). Xcerpt Database Terms *are expressions inductively defined as follows and satisfying Conditions 1 and 2 given below:*

1. *If l is a label, then l is a (atomic) database term.*
2. *If id is an identifier and t is a database term neither of the form $id_0\colon t_0$ nor of the form $\uparrow id_0$, then id$\colon t$ is a database term.*

3. *If id is an identifier, then* $\uparrow id$ *is a database term.*
4. *If* l *is a label and* $t_1, \ldots, t_n$ *are* $n \geq 1$ *database terms, then* $l[t_1, \ldots, t_n]$ *and* $l\{t_1, \ldots, t_n\}$ *are database terms.*

Condition 1: *For a given identifier id an* identifier definition *id*: t_0 *occurs at most once in a term.*
Condition 2: *For every* identifier reference $\uparrow id$ *occurring in a term* t *an identifier definition id*: t_0 *occurs in* t*.*

Example 2. The following Xcerpt database terms describe the book offers of two online book stores `bn.com` and `amazon.com` (This example is inspired from the W3C XQuery Use-Cases [7]). Note that both bookstores rely on different data formats.
`bn.com`:

```
bib {
  a1: author { last{ "Stevens" }, first { "W." } },
  a2: author { last{ "Abiteboul" }, first { "Serge" } },
  a3: author { last{ "Buneman" }, first { "Peter" } },
  a4: author { last{ "Suciu" }, first { "Dan" } },

  book {
    title { "TCP/IP Illustrated" },
    authors [ ↑ a1 ],
    publisher { "Addison-Wesley" },
    price { "65.95" }
  },
  book {
    title { "Advanced Programming in the Unix environment" },
    authors [ ↑ a1 ],
    publisher { "Addison-Wesley" },
    price { "65.95" }
  },
  book {
    title { "Data on the Web" },
    authors [ ↑ a2, ↑ a3, ↑ a4 ],
    publisher { "Morgan Kaufmann Publishers" },
    price { "39.95" }
  }
}
```

`amazon.com`:

```
reviews {
  entry {
    title { "Data on the Web" },
    price { "34.95" },
    review { "A good discussion of semi-structured database systems and
                                                                    XML." },
  },
```

```
  entry {
    title { "Advanced Programming in the Unix environment" },
    price { "65.95" },
    review { "A clear and detailed discussion of UNIX programming." },
  },
  entry {
    title { "TCP/IP Illustrated" },
    price { "65.95" },
    review { "One of the best books on TCP/IP." }
  }
}
```

Note that in both examples the element order is of no importance. This is expressed in the Xcerpt syntax using the single curly brackets { }. However, in the author list of the first example, order might be of relevance and is thus expressed using the square brackets [ ]. Using the $\uparrow$ *id* and *id* : *t* constructs in the first example make it possible to avoid redundant specifications of the authors. □

2.2 Query Terms

A query term is a pattern that specifies a selection of database terms very much like logical atoms and SQL selections do. The evaluation of query terms (cf. below Definition 12 for a formalisation) differs from the evaluation of logical atoms and SQL selections as follows:

1. Answers might have additional subterms to those mentioned in the query term.
2. Answers might have another subterm ordering than the query.
3. A query term might specify subterms at an unspecified depth.

In query terms, the single square and curly brackets, [ ] and { }, denote "exact subterm patterns", i.e. single (square or curly) brackets are used in a query term to be answered by database terms with no more subterms than those given in the query term. Double square and curly brackets, [[ ]] and {{ }}, on the other hand, denote "partial subterm patterns".

[ ] and [[ ]] are used if the subterm order in the answers is to be that of the query term, { } and {{ }} are used otherwise. Thus, possible answers to the query term $t_1 = a[b, c\{\{d, e\}\}, f]$ are the database terms $a[b, c\{d, e, g\}, f]$ and $a[b, c\{d, e, g\}, f\{g, h\}]$ and $a[b, c\{d, e\{g, h\}, g\}, f\{g, h\}]$ and $a[b, c[d, e], f]$. In contrast, $a[b, c\{d, e\}, f, g]$ and $a\{b, c\{d, e\}, f\}$ are no answers to t_1. The only answers to $f\{\ \}$ are f-labelled database terms with no children.

In a query term, a term variable X can be constrained to some query terms using the construct $\rightsquigarrow$, read "as". Thus, the query term $t_2 = a[X_1 \rightsquigarrow b[[c, d]], X_2, e]$ constrains the term variable X_1 to such database terms that are possible answers to the query term $b[[c, d]]$. Note that the term variable X_2 is unconstrained in t_2. Possible answers to t_2 are $a[b[c, d], f, e]$ which binds X_1 to $b[c, d]$ and X_2 to

f, $a[b[c,d], f[g,h], e]$ which binds X_1 to $b[c,d]$ and X_2 to $f[g,h]$, $a[b[c,d,e], f, e]$ which binds X_1 to $b[c,d,e]$ and X_2 to f, and $a[b[c,e,d], f, e]$ which binds X_1 to $b[c,e,d]$ and X_2 to f. In query terms, the construct *desc*, read "descendant", specifies a subterm at an unspecified depth. Thus, possible answers to the query term $t_3 = a[X \rightsquigarrow desc\ f[c,d], b]$ are $a[f[c,d], b]$ and $a[g[f[c,d]], b]$ and $a[g[f[c,d], h], b]$ and $a[g[g[f[c,d]]], b]$ and $a[g[g[f[c,d], h], i], b]$.

Definition 2 (Query Terms). Xcerpt Query terms *are expressions inductively defined as follows and satisfying Conditions 1 and 2 of Definition 1:*

1. *If l is a label and L is a label variable, then l, L, $l\{\{\}\}$, and $L\{\{\}\}$ are (atomic) query terms.*
2. *A term variable is a query term.*
3. *If id is an identifier and t is a query term neither of the form $id_0\colon t_0$ nor of the form $\uparrow id_0$, then $id\colon t$ is a query term.*
4. *If id is an identifier, then $\uparrow id$ is a query term.*
5. *If X is a variable and t a query term, then $X \rightsquigarrow t$ is a query term.*
6. *If X is a variable and t is a query term, then $X \rightsquigarrow desc\ t$ is a query term.*
7. *If l is a label, L a label variable and $t_1, \ldots, t_n$ are $n \geq 1$ query terms, then $l[t_1, \ldots, t_n]$, $L[t_1, \ldots, t_n]$, $l\{t_1, \ldots, t_n\}$, $L\{t_1, \ldots, t_n\}$, $l[[t_1, \ldots, t_n]]$, $L[[t_1, \ldots, t_n]]$, $l\{\{t_1, \ldots, t_n\}\}$, and $L\{\{t_1, \ldots, t_n\}\}$ are query terms.*

Query terms in which no variables occur are ground. *Query terms that are not of the form $\uparrow id$, are* strict. *The* leftmost label *of strict and ground query terms of the form l, $l\{\{\}\}$, $l\{t_1, \ldots, t_n\}$, and $l[t_1, \ldots, t_n]$ is l; the* leftmost label *of a strict and ground query term of the form $id : t$ is the leftmost label of t.*

Note that *desc* never occurs in a ground query term, for it is by Definition 2 always coupled with a variable.

Example 3. Consider the bookstore databases of Example 2. The following simple query term could query the first database for titles and authors and bind the variables `TITLE` and `AUTHOR` to the corresponding values in `bn.com`'s database:

```
bib {{
  book {{
    title { TITLE },
    author { AUTHOR }
  }}
}}
```

□

The evaluation strategy of Xcerpt is based on so-called *Simulation Unification*, which is explained in more detail in [9]. The two variables `TITLE` and `AUTHOR` will have several possible bindings as a result of the simulation unification, representing all valid combinations of a title with an author that can be found in the database, e.g. `AUTHOR="Dan Suciu" and TITLE="Data on the Web"` or `AUTHOR="Serge Abiteboul" and TITLE="Data on the Web"`.

Example 3 would bind the variables to the "leafs" of the database terms. Xcerpt also allows variables at a "higher position" in a query term, as illustrated in the next example.

Example 4. The following Xcerpt query binds the variable `TITLE` to the compound element `title { ... }` (thus retrieving not the "leaf" but the parent element `title`):

```
bib {{
  book {{
    TITLE ⇝ title,
    author { AUTHOR }
  }}
}}
```

□

Thanks to Simulation Unification (cf. below Section 3), the "leaf" of a `title` element does not have to be explicitly mentioned in the query of Example 4 for being included in the answers.

Finally, the descendant construct serves to express indefiniteness.

Example 5. The following Xcerpt query retrieves the titles of books with an author "Stevens" at any depth:

```
bib {{
  book {{
    TITLE ⇝ title,
    author {{ X ⇝ desc "Stevens" }}
  }}
}}
```

□

Definition 2 requires a *desc* expression to be preceded by $X \rightsquigarrow$ for some variable X. This is convenient for simplifying the formalisation of Xcerpt's declarative semantics (cf. below Definition 10). However, this is dispensable in practice (at the cost of a more complicated counterpart in Definition 10).

Example 6. Example 5 can be expressed using the following query term (although not conforming to Definition 2):

```
bib {{
  book {{
    TITLE ⇝ title,
    author {{ desc "Stevens" }}
  }}
}}
```

□

In a query term, multiple occurrences of a same term variable are not precluded. E.g. a possible answer to $a\{\{X \rightsquigarrow b\{\{c\}\}, X \rightsquigarrow b\{\{d\}\}\}$ is $a\{b\{c, d\}\}$. However, $a[[X \rightsquigarrow b\{c\}, X \rightsquigarrow f\{d\}]]$ has no answers, for labels b and f are distinct.

Child subterms and *subterms* of query terms are defined such that if $t = f[a, g\{Y \rightsquigarrow desc\ b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}]$, then a and $g\{Y \rightsquigarrow$

desc $b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}$ are the only child subterms of t and e.g. a and X and $Y \rightsquigarrow$ *desc* $b\{X\}$ and $h\{a, X \rightsquigarrow k\{c\}\}$ and $X \rightsquigarrow k\{c\}$ and t itself are subterms of t. Note that f is not a subterm of t.

The $\rightsquigarrow$ construct makes it possible to express (undesirable) "cyclic" query terms. Definition 3 avoids such "cyclic" query terms.

Definition 3 (Variable Well-Formed Query Terms). *A term variable X* depends *on a term variable Y in a query term t if $X \rightsquigarrow t_1$ is a subterm of t and Y is a subterm of t_1. A query term t is* variable well-formed *if t contains no term variables $X_0, \ldots, X_n$ $(n \geq 1)$ such that 1. $X_0 = X_n$ and 2. for all $i = 1, \ldots, n$, X_i depends on X_{i-1} in t.*

E.g. $f\{X \rightsquigarrow g\{X\}\}$ and $f\{X \rightsquigarrow g\{Y\}, Y \rightsquigarrow h\{X\}\}$ are not variable well-formed. Variable well-formedness precludes queries specifying infinite answers. In the following, query terms are assumed to be variable well-formed.

2.3 Construct Terms

Xcerpt Construct terms serve to re-assemble variables, the "values" of which are specified in query terms, so as to form new database terms. Thus, like in database terms both constructs [ ] and { } can occur in construct terms. Variables as references to subterms specified in a query can also occur in construct terms. However, the construct $\rightsquigarrow$ is not allowed in construct terms. The rationale for forbidding $\rightsquigarrow$ in construct terms is that variables should be constrained where they are defined, i.e. in query terms, not in construct terms where they are used to specify new terms.

Since querying a database may yield multiple alternative bindings for the same variables, it might be desirable to collect all such bindings in the construction of a result. The construct *all* serves this purpose. *all* t denotes the collection of all instances of t (binding the variables free in term t in all possible ways and recursively evaluating nested *all* constructs). A variable X is *free* in t, if X is not already contained within the argument of an *all* construct.

Definition 4 (Construct Terms). *Xcerpt Construct terms are expressions inductively defined as follows satisfying Conditions 1 and 2 of Definition 1:*

1. *Labels and label variables are (atomic) construct term.*
2. *If id is an identifier and t is a construct term, then id: t is a construct term.*
3. *If id is an identifier, then $\uparrow$id is a construct term.*
4. *A term variable is a construct term.*
5. *If t is a construct term, then all t is a construct term.*
6. *If l is a label, L is a label variable and $t_1, \ldots, t_n$ are $n \geq 1$ construct terms, then $l[t_1, \ldots, t_n]$, $L[t_1, \ldots, t_n]$, $l\{t_1, \ldots, t_n\}$, and $L\{t_1, \ldots, t_n\}$ are construct terms.*

Note that construct terms that are not of the form *all* t are (simple kinds of) query terms and database terms are (simple kinds of) construct terms.

Example 7. Consider the database `bn.com` of Example 2. Assume that some query term (e.g. that of Example 3) binds the variables `TITLE` and `AUTHOR`. The construct term

```
results {
  result { TITLE, AUTHOR }
}
```

assembles the bindings of `TITLE` and `AUTHOR` into new database terms like e.g.

```
results {
  result {
    title { "TCP/IP Illustrated" },
    author { last { "Stevens" }, first { "W." } }
  }
}
```

In the general case there are several alternative bindings for the variables `TITLE` and `AUTHOR`, e.g. several values for `TITLE`. The *all* construct may be used to collect all such alternatives:

```
results {
  all result { TITLE, AUTHOR }
}
```

This yields as a result an unordered collection of `result` elements, collecting all possible combinations for `TITLE` and `AUTHOR`, e.g. like in

```
results {
  result {
    title { "TCP/IP Illustrated" },
    author { last { "Stevens" }, first { "W." } }
  },
  result {
    title { "Advanced Programming in the Unix environment" },
    author { "W. Stevens" }
  },
  ...
}
```

□

Example 8. Using the *all* construct, the XQuery expression of Example 1 returning for each author a list of all his titles can be expressed in Xcerpt as follows:

```
results {
  all result { AUTHOR, all TITLE }
}
```

The symmetric query listing for each title all its authors is expressed in Xcerpt as follows:

```
results {
  all result { all AUTHOR, TITLE }
}
```

Note that the only change from the first to the second Xcerpt construct term is the position of the *all* construct. The same query from Example 3 can be used in both cases, as opposed to XQuery which requires two completely different queries (cf. queries Q3 and Q4 of use case "XMP" in [7]). □

2.4 Construct-Query Rules

Xcerpt Construct-query rules relate queries, consisting of a conjunction of query terms, and construct terms. It is assumed (cf. below Point 3 of Definition 5) that each term variable occurring (left or right of $\rightsquigarrow$ or elsewhere) in the construct term of a construct-query rule also occurs in at least one of the query terms of the rule, i.e. variables in construct-query rules are assumed to be "range-restricted" or "allowed".

Definition 5 (Construct-Query Rule). *A construct-query rule is an expression of the form* $t^c \leftarrow t^q_1 \wedge \ldots \wedge t^q_n$ *such that:*

1. $n \geq 1$ *and for all* $i = 1, \ldots n$, t^q_i *is a query term,*
2. t^c *is a construct term, and*
3. *every variable occurring in* t^c *also occurs in at least one of the* t^q_i.

The left hand-side, i.e. the construct term, of a (construct-query) rule will be referred to as the rule "head". The right hand-side of a (construct-query) rule will be referred to as the rule "body". Note that, in contrast to the body of a Prolog clause, the body of a (construct-query) rule cannot be empty, for empty rule bodies do not seem to be needed for the applications considered.

Example 9. The following construct-query rule combines the query and construct terms used in the previous Examples 3 and 8:

```
rule { cons {
        results {
          all result { TITLE, all AUTHOR }
        }
      },
      query {
        in { "bn.com" } ,
        bib {{
          book {{ TITLE ⇝ title, AUTHOR ⇝ author }}
        }}
      }
    }
```

□

The advantage of the clear separation between construct and query parts in Xcerpt is obvious, if you recall Example 1. The rule in Example 9 also demonstrates the circularity of Xcerpt: a rule is itself a term.

Note that the *eval* part contains an *in* construct. This construct allows to specify a different resource for each query term. The rationale behind this is illustrated on the following, more complex example.

Example 10. The following rule creates a list of books with their prices in both stores `bn.com` and `amazon.com`:

```
rule { cons {
    books {
      all book { title { TITLE }, price-a { PRICEA }, price-b { PRICEB } }
    }
  },
  and {
    query {
      in { "bn.com" },
      bib {{
        book {{ title { TITLE }, price { PRICEA } }}
      }} },
    query {
      in { "amazon.com" },
      reviews {{
        entry {{ title { TITLE }, price { PRICEB } }}
      }} }
  }
}
```

□

Note that the conjunction of query terms in the rule's body in example 10 expresses an equijoin on the book title.

2.5 Xcerpt Programs

An Xcerpt program consists of one or several construct-query rules and of a "main query". A notion of modules makes it possible to combine and re-use parts of Xcerpt programs in different manners.

2.6 Rule Chaining

Xcerpt allows to "chain" rules, i.e. to evaluate one rule against the result of another rule. This allows for very complex queries and transformations, encapsulating subqueries and calculations in separate rules.

Example 11. Consider the rule of Example 10. Assume the data constructed is to be further transformed into two different formats, HTML [10] and WML [11], the one suitable for a PC screen, the other suitable for the small screen of a PDA

(personal digital assistant). In Xcerpt, this could be expressed using additional rules that query the "result" of the first rule. A transformation into an HTML table and WML card could look like this:

```
rule { cons {
    table {
      tr { td { "Booktitle" }, td { "Price at A" }, td { "Price at B" } },
      all tr { td { TITLE }, td { PRICEA }, td { PRICEB } }
    }
  },
  query {
    books {{
      book { title { TITLE }, price-a { PRICEA }, price-b { PRICEB } }
    }}
  }
}

rule { cons {
    all card {
      "Title: ", TITLE, br{},
      "Price at A", PRICEA, br{},
      "Price at B", PRICEB, br{}
    }
  },
  query {
    books {{
      book { title { TITLE }, price-a { PRICEA }, price-b { PRICEB } }
    }}
  }
}
```

Both a forward chaining (as in deductive databases) and a backward chaining (as in Prolog) are possible and reasonable for processing Xcerpt rules. Backward chaining can be very efficient but requires a "unification" of query and construct terms. Xcerpt relies on a non-standard unification called *Simulation Unification*. Simulation Unification is introduced below in Section 3.

3 Query Semantics

Xcerpt's query semantics is based on graph simulation. Informally, a simulation of a graph G_1 in a graph G_2 is a mapping of the nodes of G_1 in the nodes of G_2 preserving the edges. The graphs considered are directed, ordered and rooted and their nodes are labelled. If $G = (V, E)$ is a graph, then V is it s set of vertices and E is its set of edges:

Definition 6 (Graph Simulation). *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs and let $\sim$ be an equivalence relation on $V_1 \cup V_2$. A relation $\mathcal{S} \subseteq V_1 \times V_2$ is a simulation with respect to $\sim$ of G_1 in G_2 if:*

1. *If* $v_1 \mathcal{S} v_2$*, then* $v_1 \sim v_2$*.*
2. *If* $v_1 \mathcal{S} v_2$ *and* $(v_1, v_1') \in E_1$*, then there exists* $v_2' \in V_2$ *such that* $v_1' \mathcal{S} v_2'$ *and* $(v_2, v_2') \in E_2$*.*

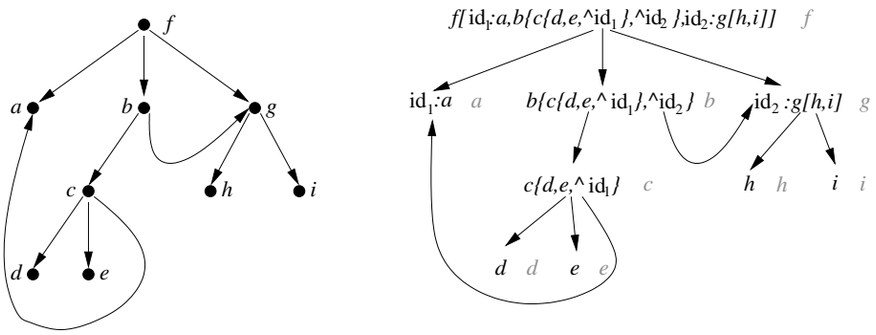
A simulation $\mathcal{S}$ *of a tree* T_1 *with root* r_1 *in a tree* T_2 *with root* r_2 *is a rooted simulation of* T_1 *in* T_2 *if* $r_1 \mathcal{S} r_2$*.*

Definition 7 (Graphs Induced by Strict and Ground Query Terms). *Let* t *be a strict and ground query term. The* graph $G_t = (N_t, V_t)$ induced by t *is defined by:*

1. N_t *is the set of strict subterms (cf. Definition 2) of* t *and each* $t' \in N_t$ *is labelled with the leftmost label (cf. Definition 2) of* t'*.*
2. V_t *is the set of pairs* (t_1, t_2) *such that either* t_2 *is a child subterm of* t_1*, or* $\uparrow id$ *is a child subterm of* t_1 *and the identifier definition* $id\colon t_2$ *occurs in* t*.*
3. *The children of a node are ordered in* G_t *like in* t*.*

Note that t is the root of G_t.

Figure 1 illustrates Definition 7. Note that the graph induced by a ground query term as defined in Definition 7 does not fully convey the term structure: Missing are representations of the various nestings [ ], { }, [[ ]] and {{ }}.



(a) Abstract node representation

(b) Full node representation (node labels in gray)

Fig. 1. Graph induced by $t = f[\mathrm{id}_1 : a, b\{c\{d, e, \uparrow \mathrm{id}_1\}, \uparrow \mathrm{id}_2\}, \mathrm{id}_2 : g[h, i]]$.

Below, a database term is identified with the graph it induces.

Definition 8 (Ground Query Term Simulation). *Let* t_1 *and* t_2 *be ground query terms. Let* S_i *denote the set of subtrees of* t_i *(*$i \in \{1, 2\}$*). A relation* $\mathcal{S} \subseteq S_1 \times S_2$ *is a ground query term simulation of* t_1 *in* t_2 *if:*

1. $t_1 \mathcal{S} t_2$*.*

2. *If* $l_1 \mathcal{S} l_2$, *then* $l_1 = l_2$.
3. *If* $l_1\{\{t_1^1, \ldots, t_n^1\}\} \mathcal{S} l_2\{\{t_1^2, \ldots, t_m^2\}\}$, *then* $l_1 = l_2$ *and for all* $i \in \{1, \ldots, n\}$ *there exists* $j \in \{1, \ldots, m\}$ *such that* $t_i^1 \mathcal{S} t_j^2$.
4. *If* $l_1\{\{t_1^1, \ldots, t_n^1\}\} \mathcal{S} l_2\{t_1^2, \ldots, t_m^2\}$, *then* $l_1 = l_2$ *and for all* $i \in \{1, \ldots, n\}$ *there exists* $j \in \{1, \ldots, m\}$ *such that* $t_i^1 \mathcal{S} t_j^2$.
5. *If* $l_1\{t_1^1, \ldots, t_n^1\} \mathcal{S} l_2\{\{t_1^2, \ldots, t_m^2\}\}$, *then* $l_1 = l_2$ *and for all* $i \in \{1, \ldots, n\}$ *there exists* $j \in \{1, \ldots, m\}$ *such that* $t_i^1 \mathcal{S} t_j^2$, *and for all* $j \in \{1, \ldots, m\}$ *there exists* $i\{1, \ldots, n\}$ *such that* $t_i^1 \mathcal{S} t_j^2$.
6. *If* $l_1\{t_1^1, \ldots, t_n^1\} \mathcal{S} l_2\{t_1^2, \ldots, t_m^2\}$, *then* $l_1 = l_2$ *and for all* $i \in \{1, \ldots, n\}$ *there exists* $j \in \{1, \ldots, m\}$ *such that* $t_i^1 \mathcal{S} t_j^2$, *and for all* $j \in \{1, \ldots, m\}$ *there exists* $i\{1, \ldots, n\}$ *such that* $t_i^1 \mathcal{S} t_j^2$.

Definition 9 (Simulation Preorder). $\preceq$ *is the preorder on the set of ground query terms defined by* $t_1 \preceq t_2$ *if there exists a ground query term simulation of* t_1 *in* t_2.

Figure 2 illustrates Definition 8. The simulation of Figure 2 is minimal for $\subseteq$ in the sense that no strict subset of this simulation relation is a simulation of t^q in t^{db}.

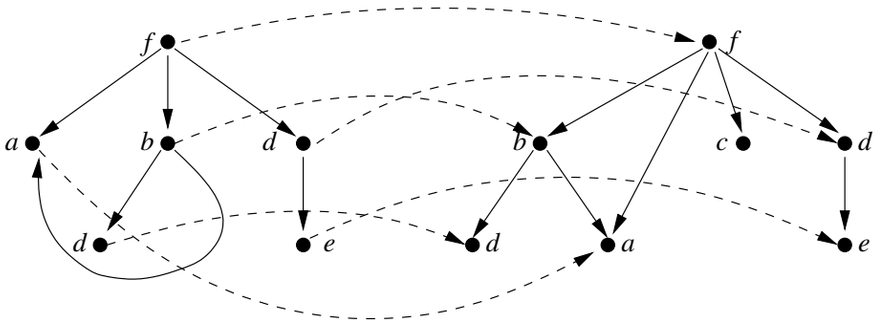


Fig. 2. A simulation of the (graph induced by the) ground query term $t^q = f\{\{\mathrm{id}_1 : a, b[d\{\}, \uparrow \mathrm{id}_1], \mathit{desc}\ e\}\}$ in the (graph induced by the) database term $t^{db} = f[b[d, \mathrm{id}_2 : a], \uparrow \mathrm{id}_2, c, d\{e\}]$.

By Definition 8, label identity is a rooted simulation of every ground query term in itself. By Definition 8, if $\mathcal{S}_1$ is a ground query term simulation of t_1 in t_2 and if $\mathcal{S}_2$ is a ground query term simulation of t_2 in t_3, then $\mathcal{S} = \{(l_1, l_3) \mid \exists l_2\ (l_1, l_2) \in \mathcal{S}_1 \wedge (l_2, l_3) \in \mathcal{S}_2\}$ is a ground query term simulation of t_1 in t_3. In other word, $\preceq$ is reflexive and transitive, i.e. it is a preorder on the set of database terms.

However, $\preceq$ is not a partial order, for although $t_1 = f\{a\} \preceq t_2 = f\{a, a\}$ and $t_2 = f\{a, a\} \preceq t_1 = f\{a\}$ (both a of t_2 can be simulated by the same a of t_1), $t_1 = f\{a\} \neq t_2 = f\{a, a\}$.

Rooted simulation with respect to label equality is a first notion towards a formalisation of answers to query terms: If there exists a ground query term simulation of a ground query term t_1, in a database term t_2, then t_2 is an answer to t_1.

An answer in a database D to a query term t^q is characterised by bindings for the variables in t^q such that the database term t resulting from applying these bindings to t^q is simulated in an element of D. Consider e.g. the query $t^q = f\{\{X \rightsquigarrow g\{\{b\}\}, X \rightsquigarrow g\{\{c\}\}\ \}\}$ against the database $D = \{f\{g\{a,b,c\}, g\{a,b,c\}, h\},\ f\{g\{b\}, g\{c\}\}\}$. The $\rightsquigarrow$ constructs in t^q yields the constraint $g\{\{b\}\} \preceq X \wedge g\{\{c\}\} \preceq X$. Matching t^q with the first database term in D yields the constraint $X \preceq g\{a,b,c\}$. Matching t^q with the second database term in D yields the constraint $X \preceq g\{b\} \wedge X \preceq g\{c\}$. $g\{b\} \preceq X \wedge g\{c\} \preceq X$ is not compatible with $X \preceq g\{b\} \wedge X \preceq g\{c\}$. Thus, the only possible value for X is $g\{a,b,c\}$, i.e. the only possible answer to t^q in D is $f\{g\{a,b,c\}, g\{a,b,c\}, h\}$.

Definition 10 (Ground Instances of Query Terms). *A* grounding substitution *is a function which assigns a label to each label variable and a database term to each term variable of a finite set of (label or term) variables. Let t^q be a query term, $V_1, \ldots, V_n$ be the (label or term) variables occurring in t^q and σ be a grounding substitution assigning v_i to V_i. The* ground instance *$t^q\sigma$ of t^q with respect to σ is the ground query term that can be constructed from t^q as follows:*

1. *Replace each subterm $X \rightsquigarrow t$ by X.*
2. *Replace each occurrence of V_i by v_i $(1 \leq i \leq n)$.*

Requiring in Definition 2 *desc* to occur to the right of $\rightsquigarrow$ makes it possible to characterise a ground instance of a query term by a grounding substitution. This is helpful for formalising answers but not necessary for language implementions. Not all ground instances of a query term are acceptable answers, for some instances might violate the conditions expressed by the $\rightsquigarrow$ and *desc* constructs.

Definition 11 (Allowed Instances). *The* constraint induced by a query term *t^q and a substitution σ is the conjunction of all inequalities $t\sigma \preceq X\sigma$ such that $X \rightsquigarrow t$ is a subterm of t^q not of the form desc t_0, and of all expressions $X\sigma \lhd t\sigma$ (read "$t\sigma$ subterm of $X\sigma$") such that $X \rightsquigarrow$ desc t is a subterm of t^q, if t^q has such subterms. If t^q has no such subterms, the constraint induced t^q and σ is the formula true. Let σ be a grounding substitution and $t^q\sigma$ a ground instance of t^q. $t^q\sigma$ is* allowed *if:*

1. *Each inequality $t_1 \preceq t_2$ in the constraint induced by t^q and σ is satisfied.*
2. *For each $t_1 \lhd t_2$ in the constraint induced by t^q and σ, t_2 is simulated in a subterm of t_1.*

Definition 12 (Answers). *Let t^q be a query term and D a database. An answer to t^q in D is a database term $t^{db} \in D$ such that there exists an allowed ground instance t of t^q satisfying $t \preceq t^{db}$.*

4 Conclusion

This article introduces the rule-based XML query and transformation language Xcerpt. While the World Wide Web Consortium [6] has proposed XQuery as a generic XML query language, rule-based querying may be advantageous in cases involving more complex queries. Rule-based querying arguably allows for programs that are easier to grasp because of a clear separation of construction and query parts. In [9], a more detailed presentation of simulation unification is given and a prototype is currently being worked on. Further examples can be found in the introductory article [12].

References

1. W3C http://www.w3.org/TR/xquery/: XQuery: A Query Language for XML. (2001)
2. Fernandez, M., Siméon, J., Wadler, P.: XML Query Languages: Experiences and Examplars. Communication to the XML Query W3C Working Group (1999)
3. W3C http://www.w3.org/Style/XSL/: Extensible Stylesheet Language (XSL). (2000)
4. Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. VLDB Journal **9** (2000) 76–110
5. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing Simulations on Finite and Infinite Graphs. Technical report, Computer Science Department, Cornell University (1996)
6. World Wide Web Consortium (W3C) http://www.w3.org/. (2002)
7. Chamberlin, D., Fankhauser, P., Marchiori, M., Robie, J.: XML query use cases. W3C Working Draft 20 (2001)
8. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: Looking Forward. In: Proceedings of Workshop on XML Data Management (XMLDM). Volume 2490 of LNCS, Springer-Verlag (2002) http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-4.
9. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proceedings of the Int. Conf. on Logic Programming (ICLP). Volume 2401 of LNCS., Copenhagen, Springer-Verlag (2002) 255–270 http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-2.
10. W3C http://www.w3.org/TR/xhtml1/: XHTML 1.0: The Extensible HyperText Markup Language. (2000)
11. WAP Forum http://www.wapforum.org: Wireless Markup Language (WML). (2000)
12. Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (invited article), Sardinia/Italy (2002) http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-11.

Author Index