Robert Meersman
Zahir Tari et al. (Eds.)

LNCS 3761

On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE

OTM Confederated International Conferences
CoopIS, DOA, and ODBASE 2005
Agia Napa, Cyprus, October/November 2005
Proceedings, Part II

DOA

Springer

Lecture Notes in Computer Science 3761

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

Editorial Board

Robert Meersman Zahir Tari
Mohand-Saïd Hacid John Mylopoulos
Barbara Pernici Ozalp Babaoglu
H.-Arno Jacobsen Joseph Loyall
Michael Kifer Stefano Spaccapietra (Eds.)

On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE

OTM Confederated International Conferences
CoopIS, DOA, and ODBASE 2005
Agia Napa, Cyprus, October 31 – November 4, 2005
Proceedings, Part II

Springer

Volume Editors

Robert Meersman
Vrije Universiteit Brussel , STAR Lab
Pleinlaan 2, Bldg G/10, 1050 Brussels, Belgium
E-mail: meersman@vub.ac.be

Zahir Tari
RMIT University, School of Computer Science and Information Technology
City Campus, GPO Box 2476 V, Melbourne, Victoria 3001, Australia
E-mail: zahirt@cs.rmit.edu.au

Library of Congress Control Number: 2005934471

CR Subject Classification (1998): H.2, H.3, H.4, C.2, H.5, I.2, D.2.12, K.4

ISSN 0302-9743
ISBN-10 3-540-29738-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-29738-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11575801 06/3142 5 4 3 2 1 0

CoopIS

Mohand-Saïd Hacid
John Mylopoulos
Barbara Pernici

DOA

Ozalp Babaoglu
H.-Arno Jacobsen
Joseph Loyall

ODBASE

Michael Kifer
Stefano Spaccapietra

OTM 2005 General Co-chairs' Message

The General Chairs of OnTheMove 2005, Agia Napa, Cyprus, are happy to observe that the conference series that was started in Irvine, California in 2002, and continued in Catania, Sicily, in 2003, and in the same location in Cyprus last year, clearly supports a scientific concept that attracts a representative selection of today's worldwide research in distributed, heterogeneous and autonomous yet meaningfully collaborative computing, of which the Internet and the WWW are its prime epitomes.

Indeed, as such large, complex and networked intelligent information systems become the focus and norm for computing, it is clear that there is an acute need to address and discuss in an integrated forum the implied software and system issues as well as methodological, theoretical and application issues. As we all know, email, the Internet, and even video conferences are not sufficient for effective and efficient scientific exchange. This is why the OnTheMove (OTM) Federated Conferences series has been created to cover the increasingly wide yet closely connected range of fundamental technologies such as data and Web Semantics, distributed objects, Web services, databases, information systems, workflow, cooperation, ubiquity, interoperability, and mobility. OTM aspires to be a primary scientific meeting place where all aspects for the development of internet- and intranet-based systems in organizations and for e-business are discussed in a scientifically motivated way. This fourth 2005 edition of the OTM Federated Conferences therefore again provides an opportunity for researchers and practitioners to understand and publish these developments within their individual as well as within their broader contexts.

The backbone of OTM is formed by the co-location of three related, complementary and successful main conference series: DOA (Distributed Objects and Applications, since 1999), covering the relevant infrastructure-enabling technologies; ODBASE (Ontologies, DataBases and Applications of SEmantics, since 2002), covering Web semantics, XML databases and ontologies; and CoopIS (Cooperative Information Systems, since 1993), covering the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. Each of these three conferences encourages researchers to treat their respective topics within a framework that incorporates jointly (a) theory , (b) conceptual design and development, and (c) applications, in particular case studies and industrial solutions.

Following and expanding the model created in 2003, we again solicited and selected quality workshop proposals to complement the more "archival" nature of the main conferences with research results in a number of selected and more "avant garde" areas related to the general topic of distributed computing. For instance, the so-called Semantic Web has given rise to several novel research areas combining linguistics, information systems technology, and artificial intelligence,

such as the modeling of (legal) regulatory systems and the ubiquitous nature of their usage. We were glad to see that in 2005 under the inspired leadership of Dr. Pilar Herrero, several of earlier successful workshops re-emerged with a second or even third edition (notably WOSE, MIOS-INTEROP and GADA), and that 5 new workshops could be hosted and successfully organized by their respective proposers: AWeSOMe, SWWS, CAMS, ORM and SeBGIS. We know that as before, their audiences will mutually productively mingle with those of the main conferences, as is already visible from the overlap in authors!

A special mention for 2005 is again due for the second and enlarged edition of the highly successful Doctoral Symposium Workshop. Its 2005 Chairs, Dr. Antonia Albani, Dr. Peter Spyns, and Dr. Johannes Maria Zaha, three young and active post-doc researchers defined an original set-up and interactive formula to bring PhD students together: they call them to submit their research proposals for selection; the resulting submissions and their approaches are presented by the students in front of a wider audience at the conference, where they are then independently analyzed and discussed by a panel of senior professors (this year they were Domenico Beneventano, Jaime Delgado, Jan Dietz, and Werner Nutt). These successful students also get free access to "all" other parts of the OTM program, and only pay a minimal fee for the Doctoral Symposium itself (in fact their attendance is largely sponsored by the other participants!). The OTM organizers expect to further expand this model in future editions of the conferences and so draw an audience of young researchers into the OTM forum.

All three main conferences and the associated workshops share the distributed aspects of modern computing systems, and the resulting application-pull created by the Internet and the so-called Semantic Web. For DOA 2005, the primary emphasis stayed on the distributed object infrastructure; for ODBASE 2005, it became the knowledge bases and methods required for enabling the use of formal semantics, and for CoopIS 2005, the topic was the interaction of such technologies and methods with management issues, such as occur in networked organizations. These subject areas naturally overlap and many submissions in fact also treat an envisaged mutual impact among them. As for the earlier editions, the organizers wanted to stimulate this cross-pollination by a "shared" program of famous keynote speakers: this year we got no less than Erich Neuhold (Emeritus, Fraunhofer/IPSI), Stefano Ceri (Politecnico di Milano), Doug Schmidt (Vanderbilt University), and V.S. Subrahmanian (University of Maryland)! We also encouraged multiple event attendance by providing "all" authors, also those of workshop papers, with free access or discounts to one other conference or workshop of their choice.

We received a total of 360 submissions for the three main conferences and a whopping 268 (compared to the 170 in 2004!) in total for the workshops. Not only can we therefore again claim success in attracting an increasingly representative volume of scientific papers, but such a harvest of course allows the program committees to compose a higher quality cross-section of current research in the areas covered by OTM. In fact, in spite of the larger number of submissions, the Program Chairs of each of the three main conferences decided to accept only

approximately the same number of papers for presentation and publication as in 2003 and 2004 (i.e, average 1 paper out of 4 submitted, not counting posters). For the workshops, the acceptance rate varies but was much stricter than before, about 1 in 2-3, to almost 1 in 4 for GADA and MIOS. Also for this reason, we continue to separate the proceedings in two books with their own titles, with the main proceedings in two volumes, and we are grateful to Springer for their suggestions and collaboration in producing these books and CD-ROMs. The reviewing process by the respective program committees as usual was performed very professionally and each paper in the main conferences was reviewed by at least three referees, with email discussions in the case of strongly diverging evaluations. It may be worthwhile to emphasize that it is an explicit OTM policy that all conference program committees and chairs make their selections completely autonomously from the OTM organization itself. Continuing a costly but nice tradition, the OTM Federated Event organizers decided again to make all proceedings available as books and/or CD-ROMs to all participants of conferences and workshops, independently of one's registration.

The General Chairs are once more especially grateful to all the many people directly or indirectly involved in the set-up of these federated conferences and in doing so made this a success. Few people realize what a large number of individuals have to be involved, and what a huge amount of work, and sometimes risk, the organization of an event like OTM entails. Apart from the persons in their roles mentioned above, we therefore in particular wish to thank our 8 main conference PC Co-chairs (DOA 2005: Ozalp Babaoglu, Arno Jacobsen, Joe Loyall; ODBASE 2005: Michael Kifer, Stefano Spaccapietra; CoopIS 2005: Mohand-Said Hacid, John Mylopoulos, Barbara Pernici), and our 26 workshop PC Co-chairs (Antonia Albani, Lora Aroyo, George Buchanan, Lawrence Cavedon, Jan Dietz, Tharam Dillon, Erik Duval, Ling Feng, Aldo Gangemi, Annika Hinze, Mustafa Jarrar, Terry Halpin, Pilar Herrero, Jan Humble, David Martin, Gonzalo Médez, Aldo de Moor, Hervé Panetto, María S. Pérez, Víctor Robles, Monica Scannapieco, Peter Spyns, Emmanuel Stefanakis, Klaus Turowski, Esteban Zimányi). All, together with their many PCs, members did a superb and professional job in selecting the best papers from the large harvest of submissions. We also thank Laura Bright, our excellent Publicity Chair for the second year in a row, our Conference Secretariat staff and Technical Support Daniel Meersman and Jan Demey, and last but not least our hyperactive Publications Chair and loyal collaborator of many years, Kwong Yuen Lai.

The General Chairs gratefully acknowledge the logistic support and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB) and RMIT University, Melbourne.

We do hope that the results of this federated scientific enterprise contribute to your research and your place in the scientific network... We look forward to seeing you again at next year's edition!

August 2005 Robert Meersman, Vrije Universiteit Brussel, Belgium
Zahir Tari, RMIT University, Australia
(General Co-chairs, OnTheMove 2005)

Organization Committee

The OTM (On The Move) 2005 Federated Conferences, which involve CoopIS (Cooperative Information Systems), DOA (Distributed Objects and Applications) and ODBASE (Ontologies, Databases and Applications of Semantics), are proudly supported by RMIT University (School of Computer Science, Information Technology) and Vrije Universiteit Brussel (Department of Computer Science) and Interop.

Executive Committee

OTM 2005 General Co-chairs	Robert Meersman (Vrije Universiteit Brussel, Belgium) and Zahir Tari (RMIT University, Australia)
CoopIS 2005 PC Co-chairs	Mohand-Said Hacid (Université Claude Bernard Lyon I), John Mylopoulos (University of Toronto), and Barbara Pernici (Politecnico di Milano)
DOA 2005 PC Co-chairs	Ozalp Babaoglu (University of Bologna), Arno Jacobsen (University of Toronto), and Joe Loyall (BBN Technologies)
ODBASE 2005 PC Co-chairs	Michael Kifer (Stony Brook University) and Stefano Spaccapietra (Swiss Federal Institute of Technology at Lausanne)
Publication Co-chairs	Kwong Yuen Lai (RMIT University, Australia) and Peter Dimopoulos (RMIT University, Australia)
Organizing Chair	Skevos Evripidou (University of Cyprus, Cyprus)
Publicity Chair	Laura Bright (Oregon Graduate Institute, Oregon, USA)

CoopIS 2005 Program Committee

Wil van der Aalst
Bernd Amann
Lefteris Angelis
Naveen Ashish
Alistair Barros
Zohra Bellahsene
Boualem Benatallah
Salima Benbernou
Djamal Benslimane
Elisa Bertino
Athman Bouguettaya
Mokrane Bouzeghoub
Christoph Bussler
Barbara Carminati

Fabio Casati
Malu Castellanos
Barbara Catania
Henning Christiansen
Bin Cui
Umesh Dayal
Alex Delis
Drew Devereux
Susanna Donatelli
Marlon Dumas
Schahram Dustdar
Johann Eder
Rik Eshuis
Opher Etzion
Elena Ferrari
Avigdor Gal
Paul Grefen
Manfred Hauswirth
Geert-Jan Houben
Michael Huhns
Paul Johannesson
Latifur Khan
Manolis Koubarakis
Akhil Kumar
Winfried Lamersdorf
Steven Laufmann
Qing Li
Maristella Matera
Massimo Mecella
Michael zur Muehlen
Werner Nutt
Andreas Oberweis
Jean-Marc Petit
Evaggelia Pitoura
Alessandro Provetti
Zbigniew W. Ras
Manfred Reichert
Tore Risch
Marie-Christine Rousset
Kai-Uwe Sattler
Monica Scannapieco
Ralf Schenkel
Antonio Si
Farouk Toumani
Susan Urban
Athena Vakali
Mathias Weske
Kyu-Young Whang
Jian Yang
Ming Yung
Arkady Zaslavsky
Leon Zhao
Roger Zimmermann

CoopIS 2005 Additional Reviewers

Rong Liu
Jianrui Wang
Agnieszka Dardzinska
Samuil Angelov
Yigal Hoffner
Sven Till
Jochem Vonk
Stratos Idreos
Christos Tryfonopoulos
Harald Meyer
Hagen Overdick
Hilmar Schuschel
Guido Laures
Frank Puhlmann
Camelia Constantin
Florian Rosenberg
Benjamin Schmit
Wil van der Aalst
Ana Karla Alves de Medeiros
Christian Guenther
Eric Verbeek
Aviv Segev
Mati Golani
Ami Eyal
Daniela Berardi
Fabio De Rosa
Woong-Kee Loh
Jae-Gil Lee

Horst Pichler
Marek Lehmann
Renate Motschnig
Diego Milano
Xumin Liu
Qi Yu
Zaki Malik
Xu Yang
George Zheng
Florian Daniel
Federico Michele Facca
Jialie Shen
Min Qin
Hong Zhu
Wei-Shinn Ku
Leslie S. Liu
Bart Orriens
James Pruyne
George Pallis
Vasiliki Koutsonikola
Konstantina Stoupa
Theodosios Theodosiiou
Sarita Bassil
Fabien DeMarchi
Etienne Canaud
Dickson Chiu
Xiang Li
Zhe Shan
Elvis Leung
Jing Chen
Jingshan Huang
Armin Haller
Kostas Stefanidis
Nikos Nikolaidis
Mick Kerrigan
Massimo Marchi
Brahmananda Sapkota
Hamid Reza Motahari
Julien Ponge
Halvard Skogsrud
Aixin Sun
Quan Zheng Sheng
Guy Gouardéres
Mar Roantree
Pierre Pompidor
Ela Hunt
Anna Cinzia Squicciarini

ODBASE 2005 Program Committee

Juergen Angele
Alessandro Artale
Mira Balaban
Denilson Barbosa
Daniela Berardi
Sonia Bergamaschi
Abraham Bernstein
Leo Bertossi
Harold Boley
Alex Borgida
Christoph Bussler
Jos de Bruijn
Gilberto Câmara
Marco Antonio Casanova
Kajal Claypool
Mariano Consens
Isabel Cruz
Rainer Eckstein
Johann Eder
Tim Finin
Enrico Franconi
Fausto Giunchiglia
Mohand Said Hacid
Jeff Heflin
Ian Horrocks
Arantza Illarramendi
Mustafa Jarrar
Christopher Jones
Vipul Kashyap
Larry Kerschberg
Roger (Buzz) King
Werner Kuhn
Georg Lausen
Bertram Ludaescher
Sanjay Madria
Murali Mani

Leo Mark
Wolfgang May
Michele Missikoff
Boris Motik
Saikat Mukherjee
Moira Norrie
Maria Orlowska
Yue Pan
Christine Parent
Torben Bach Pedersen
Axel Polleres
Louiqa Raschid
Monica Scannapieco
Amit Sheth
Michael Sintek
Naveen Srinivasan
Steffen Staab
Jianwen Su
York Sure
David Toman
Christophides Vassilis
Holger Wache
Gerd Wagner
Guizhen Yang
Esteban Zimanyi

ODBASE 2005 Additional Reviewers

Alex Binun
David Boaz
Lior Limonad
Azzam Marii
Steffen Lamparter
Johanna Voelker
Peter Haase
Denny Vrandecic
Carsten Saathoff
Kalyan Ayloo
Huiyong Xiao
Jesús Bermúdez
Alfredo Goñi
Sergio Ilarri
Birte Glimm
Lei Li
Jeff Pan
Evgeny Zolin
Francesco Taglino
Antonio De Nicola
Federica Schiappelli
Fulvio D'Antonio
Karl Wiggisser
Christian Koncilia
Diego Milano
Dimitris Kotzinos
Ansgar Bernardi
Malte Kiesel
Ludger van Elst
Hans Trost
Adrian Giurca
Sergey Lukichev
Michael Lutz
Fabio Machado Porto
Aida Boukottaya
Matthew Moran
Roberta Benassi
Domenico Beneventano
Stefania Bruschi
Francesco Guerra
Mirko Orsini
James Scicluna
Cristina Feier

DOA 2005 Program Committee

Cristiana Amza
Matthias Anlauff
Mark Baker
Guruduth Banavar
Alberto Bartoli
Judith Bishop

Gordon Blair
Alex Buchmann
Harold Carr
Michel Chaudron
Shing-Chi Cheung
Geoff Coulson
Francisco "Paco" Curbera
Wolfgang Emmerich
Patrick Eugster
Pascal Felber
Kurt Geihs
Jeff Gray
Mohand-Said Hacid
Rebecca Isaacs
Mehdi Jazayeri
Bettina Kemme
Fabio Kon
Doug Lea
Peter Loehr
Frank Manola
Philip McKinley
Keith Moore
Francois Pacull
Simon Patarin
Joao Pereira
Rajendra Raj
Andry Rakotonirainy
Luis Rodrigues
Isabelle Rouvellou
Rick Schantz
Heinz-W. Schmidt
Douglas Schmidt
Richard Soley
Michael Stal
Jean-Bernard Stefani
Stefan Tai
Hong Va Leong
Maarten van Steen
Steve Vinoski
Norbert Voelker
Andrew Watson
Doug Wells

DOA 2005 Additional Reviewers

Jochen Fromm
Steffen Bleul
Roland Reichle
Thomas Weise
An Chunyan
Glenn Ammons
Norman Cohen
Thomas Mikalsen
Paul Grace
António Casimiro
Hugo Miranda
Yuehua Lin
Shih-hsi Liu
Jing Zhang
Marc Schiely
Jaksa Vuckovic
Partha Pal
Paul Rubel
Franklin Webber
Jianming Ye
John Zinky
Vinod Muthusamy
Arlindo Flávio da Conceião
Raphael Camargo
David Cyrluk
Klaus Havelund
Arnaud Venet
Asuman Suenbuel
S. Masoud Sadjadi
Eric Kasten
Zhinan Zhou
Farshad Samimi
David Knoester
Chunyang Ye
Chang Xu
Thomas Mikalsen
Norman Cohen
Glenn Ammons
Chi-yin Chow
Kei Shiu Ho
Hans P. Reiser

Table of Contents – Part II

Distributed Objects and Applications (DOA) 2005 International Conference (continued)

Security and Data Persistence

Component Middleware

Java Environments

Peer-to-Peer Computing Architectures

Aspect Oriented Middleware

Ontologies, Databases and Applications of Semantics (ODBASE) 2005 International Conference

Information Integration and Modeling

Query Processing

Ontology Construction

Metadata

Information Retrieval and Classification

System Verification and Evaluation

Active Rules and Web Services

ODBASE 2005 Short Papers

Table of Contents – Part I

OTM 2005 Keynotes

Cooperative Information Systems (CoopIS) 2005 International Conference

Workflow

Workflow and Business Processes

Mining and Filtering

Petri Nets and Processs Management

Information Access and Integrity

Heterogeneity

Semantics

Querying and Content Delivery

Web Services, Agents

Security, Integrity and Consistency

Chain and Collaboration Mangement

Distributed Objects and Applications (DOA)2005 International Conference

Web Services and Service-Oriented Architectures

Multicast and Fault Tolerance

Communication Services (Was Messaging and Publish/Subscribe)

Techniques for Application Hosting

Mobility

Evaluation of Three Approaches for CORBA Firewall/NAT Traversal*

Antonio Theophilo Costa, Markus Endler, and Renato Cerqueira

PUC-Rio - Catholic University of Rio de Janeiro
{theophilo, endler, rcerq}@inf.puc-rio.br

Abstract. Applications that use CORBA as communication layer often have some restrictions for multi-domain deployment. This is particularly true when they have to face firewall/NAT traversal. Furthermore, nowadays there isn't a well-accepted unique or standardized solution adopted by all ORBs, compelling applications using this middleware to use proprietary solutions that sometimes do not address the environment restrictions in which they are deployed (e.g. impossibility to open firewall ports). This work presents and compares three solutions for firewall/NAT traversal by CORBA-based distributed applications, each one suitable for a specific situation and exploring its advantages. Examples of such situations are the possibility of opening firewall ports or the possibility of starting a TCP connection to the outside network.

1 Introduction

From its beginning, the CORBA specification [Group, 2004a] [Henning and Vinoski, 1999], [Bolton and Walshe, 2001] was designed aiming to offer to developers a reduction of the complexity of developing distributed object-oriented applications. However, in the meantime the Internet has seen a vertiginous growth in the number of hosts and users, and unfortunately, also a growth of misuse and attacks to networks and the need to protect them. One of these countermeasures has been the extensive use of firewalls [Tanenbaum, 2003] to control in-bound and out-bound traffic to/from a protected network.

So far, firewalls and CORBA applications have coexisted quite well since the latter are usually deployed either in a single administrative domain, or only among domains of partner institutions/companies. However, problems occur when they are required to cross network barriers. The reason is that firewall/NAT crossing conflicts with following two CORBA features: location transparency and peer-based communication model [Group, 2004b].

The first feature allows clients to be unaware of the exact localization of a server object when making a remote invocation to it. These server objects can change their location without breaking existing references to them. Although this feature can be seen as a simplification from the viewpoint of the application developer, for firewall traversal this is a great problem since firewalls usually

* This work is supported by a grant from CNPq, Brazil, proc. # 550094/2005-9.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 923–940, 2005.

control in-bound traffic through a set of rules controlling which address-port pairs are allowed to be reached. To enable a change of server object location, the firewalls rules would have to be updated, so that the permission to cross the network boundaries would work.

The peer-communication model also conflicts with deployment of firewalls because it incurs into a high number of servers that consequently would require a great number of firewall rules. However, this turns out to be an unacceptable administrative burden, and would also significantly reduce the communication performance across the firewall. With CORBA this problem arises when there are a high number of ORBs deployed in the internal network (objects inside a single ORB often share a pair address/port).

Yet another problem related to inter-domain CORBA applications happens when the internal network uses NAT [1] [Tanenbaum, 2003]. This service creates an isolated IP address space inside the network and prohibits these addresses to be used in the external network. When an internal host has to send messages to the outside network, an element (usually the firewall) maps their local addresses to a few public addresses belonging to the administrative domain of the organization. The problem arises when a CORBA object that resides in a NAT network exports its IOR [Henning and Vinoski, 1999]. In the IOR the NAT address specified at IIOP profile doesn't make sense in the external network since it is not a public address, and cannot be used for routing.

This paper describes our work attempting to address the problem of firewall/NAT traversal by CORBA applications, which was driven by the following implicit requirements of acceptable solutions:

- it must not burden the firewall administration (if possible even do not require any configuration);
- as much as possible, it must be easy to configure and be transparent to the application developer;
- it must not have significant negative impact on the application performance.

The work is inserted in the scope of InteGrade Project [Goldchleger et al., 2003] which is a middleware for grid computing. It uses CORBA as middleware and needs the firewall/NAT traversal capability.

In our work we proposed, implemented and evaluated three possible solutions to this problem, where each of them assumes specific situation (or degree) of firewall configurability. The remainder of the paper is organized as follows: Section 2 presents the three proposals and Section 3 discusses implementation details of each proposal. Section 4 contains our evaluation of the three approaches and presents some results of our tests analyzed. In Section 5 we survey related work in this topic, and compare them with the proposals presented here. Finally, in Section 6 we draw some concluding remarks and future work.

[1] Network Address Translation.

2 Firewall/NAT Traversal Proposals

In networks protected by firewalls/NAT, distributed applications can face different scenarios of access permissions to the outside network. These can range from the plain possibility of opening TCP connections to the outside network to the total prohibition of establishing any kind of connection except HTTP connections through a proxy.

Similarly, network security policies (and the negotiation willingness of network administrators) vary a lot. In some cases, it is relatively easy to get a single port at the firewall opened, but in many cases such concession is extremely difficult.

In the following we describe three approaches for CORBA firewall/NAT traversal, each of them being suited to a different level of network access permission. The idea is that the application deployer should be able to choose the most appropriate alternative and configure her application accordingly using an XML configuration file (described in Section 3.1).

2.1 OMG Approach

The first approach is based on the OMG specification for CORBA firewall traversal [Group, 2004b] presented at Section 5.1. It addresses the situations where the server object is not allowed to receive connections from the outside network, or stated the other way round, the client ORB is disallowed to create connections to any host in a *foreign* protected network.

This approach requires a single port to be opened at the firewall, which is used to drain all the IIOP traffic crossing the network boundaries. Within the protected network, all this in-bound traffic is directed to an application proxy [Group, 2004b], which listens at the opened firewall port and is responsible for forwarding these messages to the intended host. Similarly, all out-bound IIOP traffic is first routed to the application proxy which will use the opened IIOP port to send the messages to its final destination. Hence, a single port needs to be opened at the firewalls, which can be shared by several applications (Figure 1).

When a server object is to be deployed, it must somehow advertise the existence of its application proxy. According to the specification, this is done through a tagged component which is to be added to the IIOP profile of the object's IOR and which contains references to all intermediaries (i.e. proxies) between the external network (e.g. Internet) and the object ORB, including itself. Thus, in the simplest and more common case, it would include only a reference to one application proxy and the ORB. When the client ORB receives this IOR it has to identify this tagged component and create a GIOP NegotiateSession message mentioning all the elements between itself and the server object ORB, including the server object ORB and any other intermediary, excluding itself. This message is then sent sequentially to all of the intermediaries mentioned in the IIOP profile, from the first to the last before the server object (i.e. the server application proxy). If the message succeeds in reaching this last element, it will send a reply to the client ORB announcing that the client is allowed to send normal GIOP messages (Request, Reply, ...).

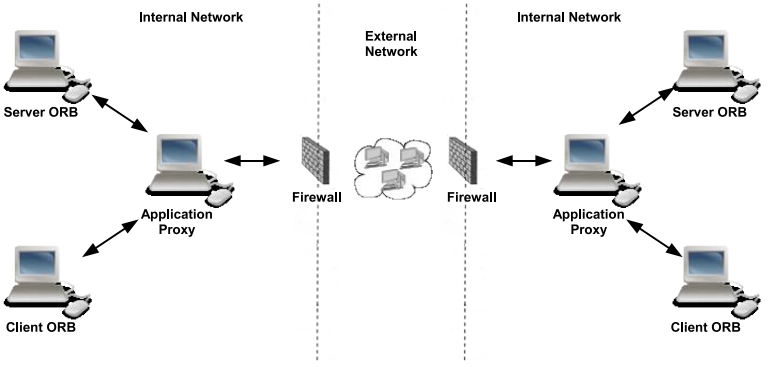


Fig. 1. OMG Approach

In addition, in this approach both the server and client must inform their ORBs of the existence of an eventual application proxy. This is done through a XML configuration file (see Section 3.1) which contains all information needed to build the tagged component and the GIOP NegotiateSession message, by the object ORB and the client ORB, respectively.

This solution has the advantage enabling the interoperability among different ORBs that follow the OMG standard. However, the major drawbacks are the need of a firewall configuration and that the client ORB must be able to properly recognize and handle these new IIOP elements added by the specification.

2.2 TCP Proxy Approach

This approach is intended for server objects that cannot receive connections from elements located outside network, but which may create TCP connections with them.

The main idea is to deploy a proxy (from now called *TCP proxy*, due to the specific transport protocol used) at the outside network and make the object ORB within the protected network connect to it at startup and keep this connection as long as it wishes to remain reachable from the outside network. The object ORB sends a registration message to the TCP proxy for each CORBA object exported, and receives an IOR to be published. This IOR contains a proxy's endpoint, and every client using this IOR contacts the proxy as if it were the object ORB. The proxy then forwards the request to the destination ORB through the TCP connection opened at ORB startup. Through this connection it also receives the replies for the requests, and forwards them to the intended client ORB (see Figure 2). In this figure, the arrows labeled with 1 show the connections opened by object ORBs at startup, while the arrow labeled with 2 represent the connection made by a client ORB after it has received the object's IOR.

Two kinds of connections to the proxy are made by the object ORB. The first one is a short-lived connection created when the ORB needs to register an

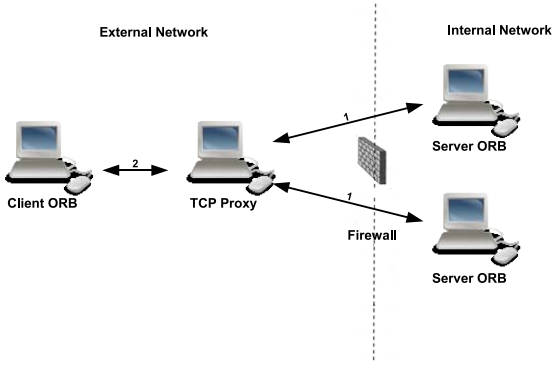


Fig. 2. TCP Proxy Approach

object at the proxy. It is used only for sending the registration message and receiving of the reply containing the IOR to be published.

Hence, during the ORB lifetime many such connections may be made. The second type is a long-lived connection that is made after the first successful object registration. It will be used to receive/send the requests/replies for the objects that have already been or (will be) registered. After the initial handshaking, normal GIOP messages are sent through this single connection shared by all objects. In what follows, we will call them *registration connection* and *data connection*, respectively.

Apart from the fact that a registration must be done at each CORBA object creation, the remaining processing is transparent to the object ORBs: after the the data connection is opened, all request processing is done as if this connection were a normal ORB listen connection with normal GIOP messages passing through it. From the client ORB perspective this is also totally transparent, and no modification whatsoever is required.

An important issue concerns the request IDs used in GIOP Request and Reply messages. According to the CORBA specification [Group, 2004a] this field is used to match requests and replies messages sent over the same connection, and it is the client's responsibility to assign ids correctly in order to avoid ambiguities, e.g. ids for requests which have been canceled, or which have pending replies, must not be reused. Since in this approach, the proxy plays the role of a client of the object ORB and forwards requests, request ids sent by the original clients cannot be considered. This is because these ids will be sent over the same data connection (between the proxy and the object ORB) and may cause ambiguities both at the server ORB and the proxy. Hence, the proxy must generate new request IDs and keep a mapping between the original and the new IDs to replace the new ID with the original ID in reply messages forwarded to the corresponding client ORB.

The main advantage of this approach is that it does not require any firewall/NAT configuration, as long as a connection to the outside network can be made and sustained. Other positive points are the transparency to clients and the ease to modify the object ORB to support this approach (i.e. once the data

connection is open, it is treated as a trivial GIOP listen connection). The main drawback is its lack of scalability at the proxy, as a connection must be kept with each object ORB, but the number of connections are usually limited by the underling operational system.

2.3 HTTP Proxy Approach

The third approach serves server objects in protected networks that can neither get connections from elements at the outside network, nor create TCP connections with them, but where the only allowed connectivity to the external network is by using HTTP protocol.

Similarly to the TCP Proxy (cf. Section 2.2) in this case a proxy (HTTP Proxy[2]) is also deployed on the external network which will be contacted by the client ORBs as if it were the CORBA object owner. Because the object ORBs have the above mentioned access restriction, all its communication with the proxy will use polling to the HTTP Proxy. Thus, in this approach object ORBs send messages encapsulated in a HTTP request and receive the corresponding HTTP reply containing the data requested by the encapsulated messages.

The first thing an object ORB must do is to register one or more CORBA objects at the proxy by sending a HTTP request containing these registration messages and receiving a HTTP reply with the object's IORs to be exported. The IOR created by the proxy will contain its (proxy) endpoint so that the requests can be directed to it. When the proxy receives requests, it will store them and wait for another HTTP request from the object ORB that registered the CORBA object. Once this HTTP request arrives, the stored GIOP Requests will be piggybacked on the HTTP reply sent to the ORB. In addition to object registration request, the object ORB can also send three more types of message: Remove, Reply and Polling. The first is a request to remove an object registry at the proxy; the second is intended to carry a GIOP Reply of a previously received GIOP Request. The third kind of message is used to ask the proxy whether it has some GIOP Requests messages stored for the object. The Polling message thus aims at implementing a periodic and continuous inspection of newly arrived GIOP Requests. The polling periodicity (i.e. time interval) used by the ORB must be set in the configuration file (cf. Section 3.1).

The proxy can send the following two types of messages to the object ORB: RegisterReply and Request, where the former one acknowledges a previously received register message, while the latter contains a GIOP Request received by the proxy from a client ORB.

The HTTP messages can hold any number of encapsulated messages: a HTTP request can have in its body any combination of the four previously described message types, and the HTTP reply may have any combination of RegisterReply and Request messages. This has the advantage of reducing the number of HTTP messages exchanged between the proxy and the object ORB and of reducing the latency of the GIOP request handling.

[2] The name HTTP Proxy is due to the use of HTTP as transport protocol.

As with the TCP Proxy approach, the HTTP Proxy also cannot directly use the request IDs sent by the client ORBs and will generate new request IDs and map them to the original IDs in other to properly forward the replies.

The main advantages of this approach are that it neither requires firewall/NAT configuration (only the "usual" HTTP connectivity), nor any specific client ORB configuration. Of course, its main disadvantage is the HTTP polling. Depending on its periodicity polling either causes waste of network bandwidth, or decreases the applications' responsiveness. Yet another disadvantage is that it requires more adaptations than the TCP Proxy Approach at the object ORB level (see Section 3.4).

3 Implementation

In this section we discuss some issues related to the implementation of the approaches described in Section 2. In our implementation we used a lightweight ORB called OiL [Cerqueira et al., 2005] that is written in the scripting language Lua [Ierusalimschy, 2003] [LUA, 2005]. Like OiL, all the proxies are written in Lua. We start describing the XML configuration file, which is common to all approaches, and then explain the implementation of each approach in some detail.

3.1 Configuration File

In order to deploy a CORBA application that should be able to traverse firewall/NAT, the application developer must write a configuration file in XML. The ORB is informed about the configuration file path through the global variable `FIREWALL_TRAVERSAL_CONF_FILE`.

The XML file has a root element called `firewall-traversal` with a single mandatory attribute called `choice`. This attribute indicates which approach is to be used, and can assume the values `omg` or `proxy`, where the latter represents both the TCP and the HTTP proxy approaches. If the OMG approach is chosen, child elements `inbound-path` and `outbound-path` will indicate if an in-bound or out-bound (or both) path need to be traversed, and their child elements will describe each host of the corresponding path. If `proxy` has been selected, a mandatory child element named `proxy` defines the kind of proxy (TCP or HTTP) and other parameters, such as host address, port and pooling interval (in case of a HTTP proxy), as shown in the following example of a HTTP proxy configuration file.

```
<?xml version='1.0'?>
<!DOCTYPE firewall-traversal PUBLIC "Firewall Traversal
- LAC PUC-Rio" "http://www.lac.inf.puc-rio.br/~theophilo/
firewallTraversal/firewallTraversal.dtd">
<firewall-traversal choice="proxy">
<proxy type="http" address="145.72.24.240"
       port="10012" polling-interval="1"/>
</firewall-traversal>
```

3.2 OMG Approach

To implement the OMG Approach, (cf. Section 2.1), both the server and client ORB required some modifications so that they were able to handle the new data structures defined by OMG. Moreover, an application proxy had to be developed in order to enable access to CORBA objects from the external network.

ORB. As explained in Section 2.1, the traversal information is carried in each IOR through a tagged component in the IIOP profile. Since the tagged component sequence (IOR field where the tagged components are stored) was defined in IIOP version 1.1, the ORB must support it and also be able to handle the elements defined in [Group, 2004b]. Since OiL didn't have this support from the beginning, we had to incorporate this feature in this ORB.

In the server-side ORB the only required modification was the creation of the tagged component in the IIOP profile of the IOR. At each CORBA object creation the ORB checks for any firewall traversal option and whether the OMG Approach has been chosen. If this is the case, it reads some information from the configuration file and uses it to create the tagged component. After that, all the further processing is done as usual, and the server ORB will not be able to distinguish if a request comes directly from a client or through the application proxy.

The client-side ORB has to be modified as follows: before sending a remote invocation message, it first has to check if the object's IOR contains a tagged component for firewall traversal, and if this is the case, a NegotiateSession message has to be sent to the first element between the two. Only after a successful reply, the normal request message can be sent. To avoid the building of the path between the client and the server at each remote invocation, the client-side ORB maintains a cache, saving the time of the IIOP profile and XML file configuration analysis.

Application Proxy. The application proxy was developed with the purpose of supporting multiple concurrent clients and not blocking on any I/O operation, i.e. avoiding the blocking at any accept or receive operation [Stevens, 1998]. The concurrent handling of clients was implemented using Lua coroutines [de Moura et al., 2004] [Ierusalimschy, 2003]. Figure 3 depicts the coroutines within the proxy and their creation order: a main coroutine called *dispatcher* is responsible for choosing the next coroutine to be executed, which also creates and starts a second coroutine called *listenClient* which is responsible for listening to new connections from client ORBs. Whenever a new connection is accepted the *listenClient* coroutine creates and starts a new *treatClient* coroutine, which will be responsible for handling all the communication between two ORBs, after which it will be destroyed.

Both the *listenClient* and the *treatClient* coroutines return control to the *dispatcher* whenever they block at a network I/O operation. It is the *dispatcher*'s responsibility to resume each such coroutine as soon as data arrives at the corresponding connection.

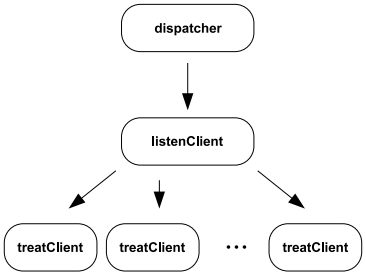


Fig. 3. Application Proxy Coroutines

Problems Encountered. When developing the OMG Approach we encountered some problems due to the omission of some points in the OMG standard. The first problem is related to the GIOP NegotiateSession message. Although the firewall traversal specification mentions its existence, its definition is a reference to a yet unpublished section at [Group, 2004a]. Since the IDL definition of this message could not be found, we created our own simple one, shown below:

```
struct NegotiateSession_1_3 {
  IOP::ServiceContextList service_context;
}
```

The other omission problem relates to the existence or not of the service context entry `FIREWALL_PATH` in the GIOP messages Request and Reply. The existence of this entry (created and used at NegotiateSession message) in these messages would permit the reutilization of already opened connections among proxies.

3.3 TCP Proxy Approach

To implement TCP Proxy Approach (cf. Section 2.2) we had to modify the server ORB and to develop a TCP proxy. As this approach is transparent to the client, no changes in its ORB were necessary.

ORB. The only modification required on the server-side ORB was to make it register newly created CORBA objects at the proxy. At the first time (i.e. first object registration) a data connection is also initialized and inserted in the list of connections used for GIOP message listening. Thereafter, all processing is done as usual. In order to give the application developer access to the object's IOR exported and used at external network, a new method called `_get_ior_exported` was included to the servant returned by the ORB.

Proxy. Like the OMG Application Proxy (Section 3.2), the TCP Proxy was also implemented to handle concurrent requests using Lua coroutines. Figure 4 shows the different types of coroutines and their creation order. Here there is also a main coroutine called *dispatcher* which is responsible to schedule the execution of the other coroutines. Its first action is to create two auxiliary coroutines

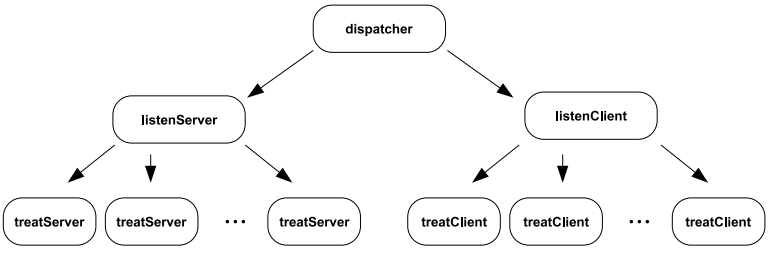


Fig. 4. TCP Proxy Coroutines

called *listenServer* and *listenClient*, responsible for listening to new server and client ORBs connections requests, respectively. Whenever a new connection is established, it creates a new coroutine (*treatServer* and *treatClient* coroutines), responsible for handling the communication with a server or a client ORB, respectively.

The *treatServer* coroutine is in charge of handling either a server ORB request to register an object at the proxy, or to open a data connection through which GIOP messages will pass. In the first case, the coroutine is destroyed after the registration is done. In the latter case, the coroutine remains during the lifetime of the data connection, listening to new GIOP Reply messages and forwarding them to the correct client.

On the other hand, the *treatClient* coroutine just waits for GIOP Request messages, manipulates and forwards them to the correct server ORB. It is destroyed when either the client closes the connection or sends a GIOP CloseConnection message.

3.4 HTTP Proxy Approach

As the TCP Proxy Approach, in this one we have needed to modify the server side ORB and to build the proxy to validate the solution.

ORB. The server side ORB modifications required in this approach are more complex than the ones demanded in the TCP Proxy approach. At each CORBA object creation, a registration must be done using HTTP protocol and the one defined in the Section 2.3. The complexity comes out at the inspection for new requests. In the TCP Proxy approach it suffices to add the data connection opened to a list of connections that would go on a select operation [Stevens, 1998] due to the fact that only GIOP messages are exchanged. Now a different approach must be done because requests may arrive encapsulated in HTTP reply messages.

The original request listening process of OiL is very simple: the server application calls a function called `lo_handleRequest` that will listen for one GIOP Request and reply it. If the application want to treat a infinite number of requests it can put this call on an infinite loop. At ORB level, this function calls a select operation on a list of connections to get a single request. This is the point that must be modified: now the select operation on normal listening connections

must be interleaved with HTTP polling to the HTTP Proxy. The polling interval specified at configuration file (Section 3.1) must be honored and because of this the select operation must have a timeout based on it. Other important point to be noticed is that this HTTP polling can bring more than one request, which compels the ORB to store the additional requests in order to be used in subsequent invocations of the `lo_handleRequest` function.

Proxy. The HTTP Proxy built to validate the solution is very similar to the one developed in the Section 3.3. The architecture presented in Figure 4 is the same of the HTTP Proxy: there is a coroutine responsible for the scheduling of the others (*dispatcher*); two coroutines responsible for server and client listening (*listenServer* and *listenClient*); and others coroutines responsible for the treatment of a single server or client (*treatServer* and *treatClient*).

4 Validation

Once implemented, we evaluated and compared the three approaches by making some performance and scalability tests. For those tests, we run the ORB clients on PCs (Intel Pentium IV 2,80 GHz with 1 GB RAM and running Linux) hosted in two of our Labs (i.e. university sub-networks), and where each client made invocations to a different server object. The server objects were executed on 13 machines (Intel Pentium IV 1,70 GHz with 256 MB RAM and running Linux) of our graduate student lab (LabPos), which is protected by a firewall blocking in-bound connections. The two sets of machines are interconnected by 100 Mbps network with three routers between them.

4.1 Delay

In the first test, we measured the round-trip-delay incurred by each approach when invoking firewall-protected objects. We created five firewall traversal configurations, one for testing both the OMG Approach and the TCP Proxy Approach, two for the HTTP Approach with polling intervals O and 1 second, and finally, one with without a proxy, i.e. by configuring the ORB port manually at the firewall. This fifth configuration, called Direct, was used as a reference measure, to evaluate the delay introduced by each proxy approach. For each configuration, we made approximately 5.000 invocations, and obtained the results shown in Table 1, where columns *Lower*, *Higher* and *Mean* show the lowest, highest and the mean values (in seconds) of the invocations.

As can be seen from the data, the fastest approach was the TCP Proxy Approach, whose delay on average has only 0.0022 seconds (59%) higher, and in some cases even lower delay than the Direct configuration. Compared to the OMG Approach, the main advantage of the TCP Proxy is the smaller number of TCP connections required, and the fewer number of messages exchanged at each invocation. Regarding the number of connections, in the OMG Approach, a connection between the server object ORB and the proxy has to be established

Table 1. Invocation Delay Tests Results

	Measures			
Approach	Lower	Higher	Mean	Std. Deviation
Direct	0.0023	0.0697	0.0037	0.0017
OMG	0.0083	0.3707	0.0351	0.0518
TCP Proxy	0.0036	0.0605	0.0059	0.0011
HTTP Proxy (0s)	0.0073	0.0384	0.0122	0.0024
HTTP Proxy (1s)	0.8762	1.0235	1.0127	0.0139

while in the TCP Proxy Approach such connection has already been opened before at object registration (see Section 2.2). One can say that the TCP Proxy Approach trades scalability (i.e. less number of simultaneous object handled by the proxy due to a limited number of connections allowed) for a better performance. Concerning the number of messages the OMG Approach is also less efficient than the TCP Approach, since the GIOP protocol requires NegotiateSession messages to be sent and received between the client and the application proxies. One should also notice the larger variation of the delay in the OMG Approach, expressed by the standard deviation.

Even the HTTP Proxy Approach, using the 0s polling interval, has slightly superior performance than OMG Approach, and presents less variation of delay. However, one should not forget that the HTTP Approach has a greater impact on network bandwidth due to the HTTP polling process.

In spite of the TCP Proxy Approach presenting up to 65% higher invocation times, when compared with the Direct configuration, the absolute value is still quite low. Moreover, for larger GIOP messages this overhead tends to become even smaller, for example, if the remote method had more than just an integer parameter and an integer result, as with our tests. Actually, we believe that more tests would be necessary to evaluate how this overhead varies with the kind of method being invoked.

4.2 Scalability

This test suite aimed at measuring the scalability of the approaches, i.e. how the delay caused by the proxies varies with the number of simultaneous pairs client/server interactions. The test scenario consisted of following configuration: the server objects were deployed on the 13 machines (one object for each machine) of our graduate student lab sub-network (LabPos), which is protected by a firewall and connected via one CISCO 7204 router to the network of our other research lab, where the clients were deployed on a 54-node cluster (Intel Pentium IV 1,70 GHz with 256 MB RAM and running Linux). Each of the clients executed on one cluster machine and made remote invocations to one of the servers. We measured the mean round-trip delay for up to 54 simultaneous client invocations (1 to 54 clients accessing 13 servers uniformly distributed) and the results of this test are depicted in the Figure 5.

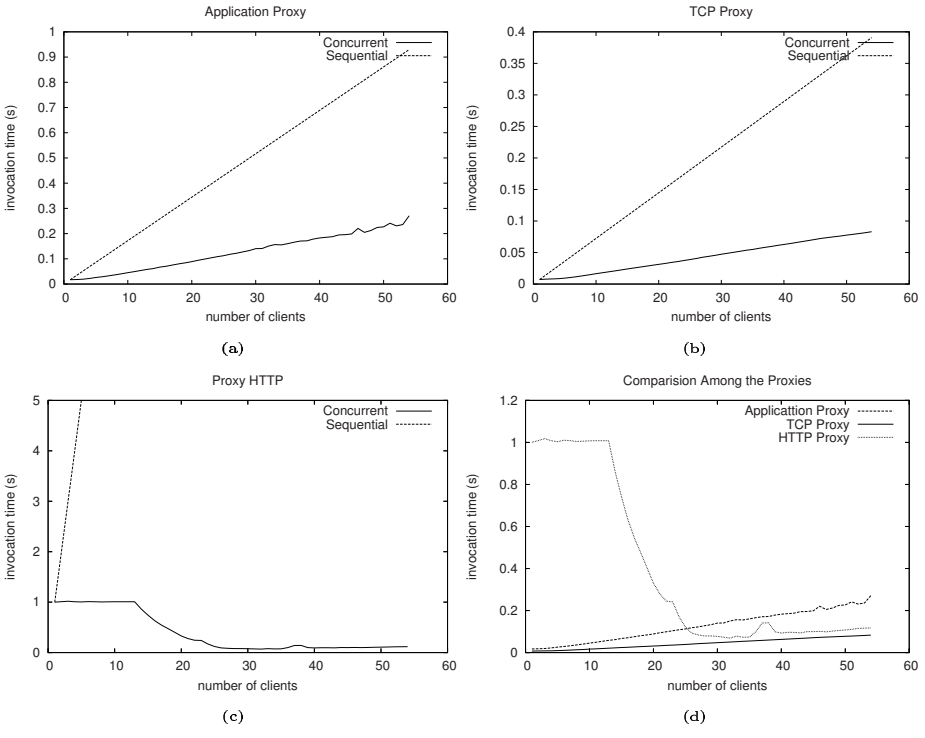


Fig. 5. Scalability Tests Results

The graphics of Figures 5.a, 5.b and 5.c show the delay measurements for the proxies of the OMG, TCP Proxy and HTTP Proxy Approach, respectively (the latter using a polling interval of 1 seconds). Each graphic compares the results obtained with a hypothetical sequential proxy, i.e. a proxy that handles client requests one at a time. As expected, in all approaches the concurrent version shows a better performance than the sequential processing. But it is interesting to notice also the reduction of the invocation delay of the HTTP Proxy with the increase of the number of clients (Figure 5.c). This starts when the client number is 14, and can be explained by the piggyback feature used in HTTP: once there is more than one client per server (at 14 clients) the client requests starts to arrive at the servers not by a polling reply, but as piggybacked on a GIOP Reply sent over HTTP (see Section 2.3). This seems to be the main cause of the delay reduction. The small delay remains until the proxy starts to become saturated with requests, which in our tests happened with approximately 32. Since in our tests we had 13 servers and a variable number of clients, the curve seems to indicate that the lowest delay is obtained when the client/server ratio is between 2 and 2.5. However, also here we believe that more tests should be made in order to obtain a better understanding of the HTTP proxy saturation behavior.

The Figure 5.d compares the delays with the three proxies, showing that the TCP proxy offers smallest increase of delay, albeit the usual server-side connectivity limitations. However, the small difference between the delay increase of the HTTP and the TCP approaches for large number of clients suggests that the HTTP approach is a reasonable alternative when the server ORBs are not allowed to open out-bound TCP connections. Moreover, the results suggest that the HTTP Proxy is better already than the OMG Approach when the client/server ratio is 1.9 or more.

5 Related Work

This section briefly describes related work on firewall traversal for CORBA applications, some of which inspired the present work.

5.1 OMG CORBA Firewall/NAT Traversal Specification

In 2004 the OMG published the CORBA Firewall Traversal Specification [Group, 2004b], which was used as the basis for our *OMG Approach* (Section 2.1).

The standard's basic idea is to extend the GIOP/IIOP protocol with data structures that enable server objects to provide information to clients and proxies on how to open connections to reach them. According to the spec, server objects that want to be reachable by external clients have to put a firewall traversal component in their IOR's tagged components sequence [Group, 2004a]. This is a data structure that contains information about the endpoints of all hosts between the server ORB (inclusive) and the external network (e.g. Internet) which are be addressed in order to make the traversal. When a client gets such an IOR it has to identify the firewall traversal component and build a GIOP NegotiateSession message, which has a service context entry also holding the information about all the hosts on the path between the client and server ORB. The client then sends this message to the first element in the path, which then gets forwarded by the hosts to the next ones on the path, until it reaches the last proxy before the server ORB. If it arrives there, this last proxy positively replies the NegotiateSession message to the previous proxy, which is then forwarded back and all the way along reverse path. When this reply arrives at the client the GIOP Requests/Reply messages can be exchanged normally.

This specification also provides others features, such as the support for secure transport protocols, which so far we have not handled in our work.

5.2 JXTA

The JXTA Project [Brookshier et al., 2002] [Gradecki, 2002] [JXTA, 2005] is a open source worldwide project created by Sun Microsystems intended to offer an infrastructure for peer-to-peer application development. It consists of a set of XML-based protocols that provide system and programming language

independence. The JXTA peers form an ad-hoc network and information is exchanged using some peers (Rendezvous and Relay peers [Brookshier et al., 2002]) as providers and routers of the network.

JXTA claims to offer firewall traversal to applications that use it as the communication layer. The solution offered is a set of public peers (Relay Peers) accessible through HTTP protocol. After the protected peer registers itself at one Relay Peer, the JXTA advertisement and routing engine will route the messages addressed to the protected peer to the Relay Peer at which it registered. This Relay Peer stores the messages until the protected peer contacts it through HTTP, using the fact that out-bound connections using this protocol are generally allowed by firewalls. The protected peers have to periodically make such inquiry (called HTTP polling) which is used both to check for messages at the Relay Peer and to send messages to the JXTA network.

At first, we took into consideration the idea of replacing the ORB communication layer by JXTA in order to traverse firewalls. However, our preliminary tests showed that the delay and high frequency of failures caused by the use of JXTA network were unacceptable. In face of this, we decided to borrow its idea of HTTP polling and provide it as an alternative for the scenarios in which no firewall configuration is possible at all, and TCP connections cannot be established without using the HTTP protocol (see Section 2.3).

5.3 JacORB

JacORB [JacORB, 2005] [Brose, 1997] is a full-featured Java implementation ORB. It provides firewall traversal through a service called *Appligator*, which is a GIOP proxy supposed to be deployed inside the protected network and to have a firewall port opened to it. It is similar to the OMG Approach described in our work, but the JacORB online manual doesn't make clear if the external client has to be configured in order to become aware of Appligator's existence, or if the latter modifies the server object's IOR in order not to require the client configuration. If the client has to be configured, then the OMG Approach is a better option regarding to interoperability. Moreover, JacORB's Appligator does not provide a solution for the situation where the firewall configuration is not possible.

NAT support is offered through an application called `fixior` that patches the Appligator's IOR, inserting the firewall endpoint.

5.4 ICE

The ICE - Internet Communication Engine [Henning, 2004] is a communication middleware similar in concept with CORBA, but not CORBA-compliant. It provides a solution for firewall/NAT traversal through a service called Glacier [Henning and Spruiell, 2005] that may also be used as the firewall of a network. Similarly to our work, ICE doesn't require any application code modification, but only a configuration file. The main idea is to configure both the client and the server to use this service, which will work as a broker between both, acting

as a client to the server, and vice-versa. It also requires firewall configuration, and the client has to be aware of the Glacier's existence.

5.5 Xtradyne

Xtradyne is a company that has a commercial product called I-DBC (IIOP Boundary Controller) [Technologies, 2005], that offers firewall/NAT traversal to applications. In fact, it is a firewall, and its solution is to patch the IOR with an endpoint opened for IIOP traffic so that client invocations are re-directed to the firewall, which in turn contacts the server. Since there is few technical information available, it is not clear how its protocol between the protected application objects and the firewall works, in order to map the outside requests to the intended recipients. An interesting feature of this firewall is its ability to identify IORs sent as CORBA parameters and to modify them accordingly in both directions.

This solution also requires firewall configuration, and does not require client awareness of the firewall's presence.

6 Conclusions and Future Work

This work has presented the design, implementation and evaluation of three solutions for CORBA-based application firewall/NAT traversal. Through several tests we have demonstrated their viability and also discussed the suitability of each approach for specific degrees of firewall/NAT permeability.

In spite of being less efficient than the TCP Proxy Approach, and in some scenarios also worse than the HTTP Approach, the solution based on the OMG standard demonstrated to be a practicable approach, specially when the client is not allowed to make an out-bound connection. An interesting future work on this approach is to support secure transport protocols, as already defined by the OMG.

The TCP Approach turned out to be the most efficient solution developed, and also the easiest to implement since it required the smallest number of server ORB modifications. Its premise - that out-bound connections are allowed - is quite common, and as no firewall configuration is required, this approach represents a very good alternative. For the cases where high performance is a strong requirement, this solution is definitely the best. However, its main drawback is a possibly limited number of simultaneous connections at the proxy. A future work on this approach could be to modify the protocol between the server ORB and the proxy, allowing the latter to terminate unused data connections and enabling the server ORB to recover from this action when desired.

The HTTP Approach gave the most interesting result. It was developed aiming to work in the most restrictive scenario, i.e. where just out-bound connections through HTTP are allowed. Its good performance, due to the extensive use of the HTTP piggybacking feature, has proven this solution to be more efficient than expected, and in some situations even better than the approach based in the

OMG specification. An interesting future work in this direction would be to make it compatible with the presence of Web proxy caches (e.g. Squid [Squid, 2005]) and enable clients in a protected network to use the proxy to send/receive GIOP Request/Reply messages.

As a final remark, it should be clear that more tests need to be done with different message sizes, method parameter types and network configurations, so that we are able to pinpoint the specific implications of each approach on the invocation delay, the network overhead, and the scalability. However, we believe that in any case a customizable, multi-solution, rather a *one-fits-all* approach should be pursued.

References

[Bolton and Walshe, 2001] Bolton, F. and Walshe, E. (2001). *Pure CORBA: A Code-Intensive Premium Reference*. Sams.

[Brookshier et al., 2002] Brookshier, D., Govoni, D., Krishnan, N., and Soto, J. C. (2002). *JXTA: Java P2P Programming*. Sams.

[Brose, 1997] Brose, G. (1997). Jacorb: Implementation and design of a java orb. In *Procs. of DAIS'97, IFIP WG 6.1 International Working Conference on Distributed Aplications and Interoperable Systems*, Cottbus, Germany.

[Cerqueira et al., 2005] Cerqueira, R., Maia, R., Nogara, L., and Mello, R. (2005). Oil - (orb in lua). http://luaforge.net/projects/o-two/ (Last Visited in 06/06/2005).

[de Moura et al., 2004] de Moura, A. L., Rodriguez, N., and Ierusalimschy, R. (2004). Coroutines in lua. *Journal of Universal Computer Science*, 10(7):910–925.

[Goldchleger et al., 2003] Goldchleger, A., Kon, F., Goldman, A., and Finger, M. (2003). Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. In *ACM/IFIP/USENIX Middleware'2003 Workshop on Middleware for the Grid*, Rio de Janeiro, Brazil.

[Gradecki, 2002] Gradecki, J. D. (2002). *Mastering Jxta: Building Java Peer-to-Peer Applications*. John Wiley & Sons, Inc.

[Group, 2004a] Group, O. M. (2004a). Common object request broker architecture: Core specification. http://www.omg.org/docs/formal/04-03-01.pdf - Version 3.0.3.

[Group, 2004b] Group, O. M. (2004b). Corba firewall traversal specification. http://www.omg.org/docs/ptc/04-03-01.pdf - Final Adopted Specification.

[Henning, 2004] Henning, M. (2004). A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75.

[Henning and Spruiell, 2005] Henning, M. and Spruiell, M. (2005). Distributed programming with ice. http://www.zeroc.com/download/Ice/2.1/Ice-2.1.1.pdf (Last Visited in 06/06/2005).

[Henning and Vinoski, 1999] Henning, M. and Vinoski, S. (1999). *Advanced CORBA programming with C++*. Addison-Wesley Longman Publishing Co., Inc.

[Ierusalimschy, 2003] Ierusalimschy, R. (2003). *Programming in Lua*. Ingram (US) and Bertram Books (UK).

[JacORB, 2005] JacORB (2005). Jacorb: The free java implementation of the omg's corba standard. http://www.jacorb.org (Last Visited in 06/06/2005).

[JXTA, 2005] JXTA (2005). jxta.org. http://www.jxta.org (Last Visited in 06/06/2005).

[LUA, 2005] LUA (2005). The programming language lua. http://www.lua.org (Last Visited in 06/06/2005).

[Squid, 2005] Squid (2005). Squid web proxy cache. http://www.squid-cache.org/ (Last Visited in 06/06/2005).

[Stevens, 1998] Stevens, W. R. (1998). *Unix Networking Programming*, volume 1. Prentice-Hall, 2 edition.

[Tanenbaum, 2003] Tanenbaum, A. S. (2003). *Computer Networks*. Prentice Hall, 4th edition.

[Technologies, 2005] Technologies, X. S. I. (2005). Iiop domain boundary controller - the corba an ejb firewall. http://www.xtradyne.com/documents/whitepapers/Xtradyne-I-DBC-WhitePaper.pdf (Last Visited in 06/06/2005).

On the Design of Access Control to Prevent Sensitive Information Leakage in Distributed Object Systems: A Colored Petri Net Based Model

Panagiotis Katsaros

Department of Informatics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
katsaros@csd.auth.gr
http://delab.csd.auth.gr/~katsaros/index.html

Abstract. We introduce a Colored Petri Net model for simulating and verifying information flow in distributed object systems. Access control is specified as prescribed by the OMG CORBA security specification. An insecure flow arises when information is transferred from one object to another in violation of the applied security policy. We provide precise definitions, which determine how discretionary access control is related to the secure or insecure transfer of information between objects. The model can be queried regarding the detected information flow paths and their dependencies. This is a valuable mean for the design of multilevel mandatory access control that addresses the problem of enforcing object classification constraints to prevent undesirable leakage and inference of sensitive information.

1 Introduction

For a secure application it is not enough to control access to objects, without taking into account the information flow paths implied by a given, outstanding collection of access rights. The problem is faced by the use of *multilevel mandatory policies*, where users have no control and therefore they cannot be bypassed. Access to objects is granted on the basis of *classifications* (taken from a partially ordered set) assigned to objects and subjects requesting access to them.

The rigidity of these policies implies the need for a systematic design approach. In the design of mandatory access control we aim to enforce *object classification constraints* to prevent undesirable leakage and inference of sensitive information, while at the same time guaranteeing that objects will not be overclassified (to maximize information visibility). The set of constraints is decided based on the restrictions of the system's enterprise environment and on a view of the potential information flow paths. Our work aims to provide information regarding the system's structure, to be used in the design of appropriate mandatory access control.

In distributed object systems, objects interact by synchronous, asynchronous or deferred synchronous messages and detection of information flow is complicated by the presence of bi-directional information transfer between the senders and the receivers. Information transfer from the sender to the receiver is accomplished through message parameters, while the opposite, through message reply. An information flow does not

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 941 – 959, 2005.

require direct message exchange between objects (*indirect information flow paths*). An object acquires information only when writing to its attributes and this operation results in one or more direct or indirect information flows.

We assume that access control is specified as prescribed by the OMG CORBA security specification. An *insecure flow* arises when information is transferred from one object to another in violation of the applied security policy.

We introduce a Colored Petri Net model to detect information flow paths based on system's object method dependencies and the applied access control. The model was implemented in CPN Tools ([7]), an advanced ML-based tool for editing, simulating and analyzing Colored Petri Nets.

To detect the insecure flow paths we implement the definitions we provide to determine how the applied discretionary access control is related to the secure or insecure transfer of information between objects. If there is an insecure flow to an object, we are often interested in enforcing an appropriate object classification constraint over all (and not only the insecure) sources of information flow to that object. In that case, we use the CPN Tools state space analysis functions to make queries regarding the detected flow paths.

Thus, the proposed model is a static analysis tool that provides data for the systematic design of multilevel mandatory access control. We preferred to use Colored Petri Net model checking and not to use Finite State Machine model checking because: (i) Colored Petri Nets possess the expressiveness and the formal analysis capacity of a Petri Net modeling language, (ii) they provide an explicit representation of both states and actions and at the same time retain the modeling flexibility provided in a programming language environment and (iii) Colored Petri Nets is a widespread modeling formalism with an easily accessible tool support that allows interactive simulation in an advanced graphical environment.

Section 2 provides a detailed description of the problem under consideration. Section 3 introduces the basic components of the information flow security model and provides the implemented definitions that determine when an information flow path is secure and when it is not secure. Section 4 focuses on the use of the CPN Tools state space analysis functions to make queries regarding the detected flows. Section 5 summarizes the latest related work reported in the bibliography and the paper concludes with a discussion on the potential impact of our work.

2 Basic Definitions and Problem Statement

Distributed object systems are composed of a set of objects $o_1, o_2, \ldots, o_n$, which interact to accomplish common goals. An object's methods $op_l^{o_i}$, $1 \le l \le$ #(methods of o_i), $1 \le i \le n$, are invoked by synchronous, asynchronous or deferred synchronous method requests.

A *message msg* is defined as a pair (*method*, *type*), with

$$\begin{aligned} type \in \; & \{s \mid \text{synchronous invocation request}\} \\ & \cup \{drq \mid \text{deferred synchronous invocation request}\} \\ & \cup \{a \mid \text{asynchronous invocation request}\} \\ & \cup \{drp \mid \text{deferred synchronous invocation reply}\} \end{aligned}$$

and

$$method \in \bigcup_i \left\{ op_l^{o_i} \mid 1 \le l \le \#(\text{methods of } o_i) \right\} \cup \{read, write\}$$

where *read*, *write* correspond to primitive synchronous messages (*type* = *s*) that are sent by an object to itself to read or respectively update its state.

A *message sequence specification* $MsgSeq(op_l^{o_i})$ for method $op_l^{o_i}$, $1 \le l \le$ #(methods of o_i) is a total order relation $\Rightarrow$ over the set

$$\{msg_s \mid 1 \le s \le \#(\text{messages in } MsgSeq(op_l^{o_i}))\}$$

of nested method invocations, as well as read and/or write messages generated by $op_l^{o_i}$. For every two msg_s, $msg_t \in MsgSeq(op_l^{o_i})$, $msg_s \Rightarrow msg_t$ if and only if msg_s *is sent/received before* msg_t. We note that $\Rightarrow$ defines an *invocation order* that does not necessarily coincide with the order in which method executions are completed.

To make it clear, if in $MsgSeq(op_l^{o_i})$ holds that $(op_k^{o_j}, drq) \Rightarrow (write, s)$, this does not mean that $op_k^{o_j}$ *is executed before write*. However, if

$$(op_k^{o_j}, drq) \Rightarrow (op_k^{o_j}, drp) \Rightarrow (write, s)$$

i.e. if the reply of a deferred synchronous request to o_j is received before *write*, then $op_k^{o_j}$ is executed before *write*. In that case, if $read \in MsgSeq(op_k^{o_j})$ and the used credentials include the read access right for o_j, then there is an information flow from o_j to o_i. Moreover, o_j may also be the source of additional indirect flows to other objects, if for example $op_l^{o_i}$ has been synchronously invoked by another object method.

An information flow is *insecure*, if the derived information is transferred to the target, in violation of the applied security policy. An information flow takes place *even if the information written to the target is not the same as the information read*, but is derived from it by executing computations.

We are primarily interested in detecting insecure flow paths and enforcing appropriate object classification constraints to prevent undesirable leakage of information. However, this is not enough. It is also necessary to ensure compliance with potential business constraints regarding undesirable inference of sensitive information. In a service-outsourcing environment (e.g. web services) these are key concerns that have to be satisfied. Thus, we also need to query the model about the detected "secure" information flow paths to an object.

The design problem under consideration is addressed by the implementation of an appropriate multilevel mandatory policy. This policy is based on the assignment of *access classes* to objects and subjects and is used in conjunction with the applied discretionary access control. Access classes in a set L are related by a partial order, called *dominance relation* and denoted by $\geq$. The dominance relation governs the visibility of information: a subject has read access only to the objects classified at the subject's level or below and is possible to perform write operations only to the objects classified above the subject's level. The expression $x \geq y$ is read as "*x dominates y*". The partially ordered set $(L, \geq)$ is assumed to be a *lattice*. We refer to the maximum and minimum elements of a lattice as $\top$ (top) and $\bot$ (bottom). Figure 1 depicts an example classification lattice.

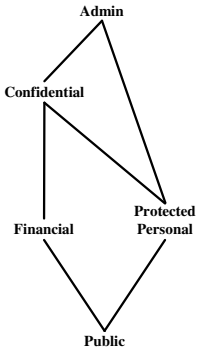


Fig. 1. An example security lattice

In general, the security level to be assigned to an object depends on the sensitivity of the data in its state. If an object's state is related to publicly accessible customer data this object might be labeled at a level such as `Public`. If an object's state is related to customer income data, this object might be labeled at a higher level, such as `Financial`.

However, the design of multilevel mandatory policies can be done in a systematic manner, based on a set of classification constraints. These constraints specify the requirements that the security levels assigned to objects must satisfy. They are constraints on a mapping λ: $\{o_i \mid 1 \leq i \leq n\} \rightarrow L$ that assigns to each object a security level $l \in L$, where $(L, \geq)$ is a classification lattice. Object classification constraints, like for example when $\lambda(o_i) \geq$ `Financial`, are used to ensure that objects are assigned security levels high enough to protect their state.

If our model detects an "insecure" information flow from o_j to o_m, the flow will take place only when the subjects *clearance level* dominates the classification of o_j and is dominated by the classification of o_m. In that case, the constraint $\lambda(o_m) \geq \lambda(o_j)$ prevents sensitive information leakage in a low-classified object (the opposite prevents the occurrence of the detected information flow in all cases). If there is also an additional flow from o_s to o_m, both of them take place only when the subjects clearance level dominates the least upper bound (`lub`) of the classifications of o_s and o_j. In that case, the constraint

$$\texttt{lub}\{\lambda(o_s), \lambda(o_j)\} \geq \texttt{Financial}$$

will ensure that both flows will take place only when the subjects clearance level dominates a given ground level. In this way it is possible to prevent potential inference of high-classified data from low-classified objects.

The proposed model is queried regarding the "insecure" and the "secure" information flow paths, in order to direct the specification of constraints, wherever there is a need *to prevent undesirable information leakage or information inference*. This allows to avoid overclassification and thus to maximize information visibility. It is then possible to implement an appropriate mandatory policy after having solved the derived set of constraints. A recently published algorithm is the one described in [3], but it is not the only one published.

3 The Colored Petri Net Based Model

Colored Petri Nets (CP-nets) provide us the primitives for the definition of diverse data types (such as privilege attributes, method names, object ids and others) and the manipulation of their data values, while retaining the expressiveness and the formal analysis capacity of a Petri Net modeling language.

The formal semantics of CP-nets is outlined in Appendix. Model states are represented by means of *places* (which are drawn as ellipses). Each place has an associated *data type* determining the kind of data, which the place may contain (by convention the type information is written in italics, next to the place). The type declarations implicitly specify the operations that can be performed on the values of the types. A state of a CP-net is called a *marking* and consists of a number of *tokens* positioned on the individual places. Each token carries a data value, which belongs to the type of the corresponding place.

A marking of a CP-net is a function, which maps each place into a *multi-set of tokens* (see the Appendix) of the correct type. We refer to the token values as *token colors* and to their data types as *color sets*. The types can be arbitrarily complex, e.g., a record where one field is a real, another field is a text string and a third field is a list of integers.

CP-net actions are represented by means of *transitions*, which are drawn as rectangles. An incoming arc indicates that the transition may remove tokens from the corresponding place while an outgoing arc indicates that the transition may add tokens. The exact number of tokens and their data values are determined by *arc expressions*, which are positioned next to the arcs. Arc expressions may contain variables as well as constants. To talk about the *occurrence of a transition*, we need to bind incoming expressions to values from their corresponding types. Let us assume that we bind the incoming variable v of some transition T to the value d. The pair $(T, <v = d>)$ is called *binding element* and this binding element is *enabled* in a marking M, when there are enough tokens in its input places. In a marking M, it is possible to have enabled more than one binding elements of T. If the binding element $(T, <v = d>)$ occurs, it removes tokens from its input places and adds tokens to its output places. In addition to the arc expressions, it is possible to attach a boolean expression with variables to each transition. This expression is called *guard* and specifies that we only accept binding elements for which the expression evaluates to true.

The behavior of a CP-net is characterized by a set of dynamic properties:

- *Bounds-related properties* characterize the model in terms of the number of tokens we may have at the places of interest.
- *Home properties* provide information about markings or sets of markings to which it is always possible to return.
- *Liveness properties* examine whether a set of binding elements X remains active: "For each reachable marking M', is it possible to find a finite sequence of markings starting in M' that contain an element of X?"
- *Fairness properties* provide information about how often the different binding elements occur.

CP-nets are analyzed, either by

- simulation,
- formal analysis methods such as the construction of *occurrence graphs*, which represent all reachable markings,
- calculation and interpretation of system *invariants* (called place and transition invariants),
- performance of *reductions* which shrink the net without changing a certain selected set of properties and
- the check of structural properties, which guarantee certain behavioral properties.

In CPN Tools, CP-nets are developed in a modern GUI-based environment that provides interactive feedback for the model's behavior through simulation. Colors, variables, function declarations and net inscriptions are written in CPN ML, which is an extension of Standard ML and for this reason employs a functional programming style. In CPN Tools we employ simple as well as compound color sets such as product, record, list and union color sets.

The toolset provides the necessary functionality for the analysis of simple and *timed CP-nets* specified in a number of *hierarchically related pages*. Typical models consist of 10-100 pages with varying complexity and programming requirements. The companion *state space tool* allows the generation of the entire or a portion of the model's state space (occurrence graph) and the performance of standard as well as non-standard analysis queries.

In our model, *the CP-net structure depends on the system's object method dependencies*. These dependencies may be derived from the system's source code with a code-slicing tool ([11]). Taking into account that in CPN Tools the net is stored in an XML-based format, we believe that models can be automatically generated using an appropriate XML text generator.

3.1 The CORBA Security Model

Distributed object systems typically support a large number of objects. CORBA Security ([13]) provides abstractions to reduce the size of access control information and at the same time to allow fine-grained access to individual operations rather than to the object as a whole. Access policies are defined based on privilege and control attributes and access decisions are made via the standard access decision interface that is modeled by the CP-net we present here.

Principals are users or processes accountable for the actions associated with some user. In a given security policy, each principal possesses certain *privilege attributes* that are used in access control: such attributes may be access identities, roles, groups, security clearance and so on. At any time, a principal may choose to use only a subset of the privilege attributes it is permitted to use, in order to establish its rights to access objects.

Access control is defined at the level of individual object invocations. The access decision function bases its result on the current privilege attributes of the principal, the operation to be performed and the *access control attributes* of the target object.

A set of objects where we apply common security policies is called *security policy domain*. Security domains provide leverage for dealing with the problem of scale in

policy management. The CORBA Security specification allows objects to be members of multiple domains: the policies that apply to an object are those of all its enclosing domains. CORBA Security does not prescribe specific *policy composition rules*. Such rules are the subject of the system's security design and this allows for potentially unlimited flexibility in combining complementary access control policies.

A *domain access policy* grants a set of subjects the specified set of rights to perform operations on all objects in the domain. In Table 1 we provide a sample domain access policy. As subject entries we use the privilege attributes possessed by the principals. Thus, user identities can be considered to be a special case of privilege attributes. In CORBA Security, rights are qualified into sets of "access control types", known as *rights families*. There is only one predefined rights family that is called `corba` and contains the three rights `g` (for get or read), `s` (for set or write) and `m` (for manage).

Table 1. Domain access policy (granted rights)

Privilege Attribute	Domain	Granted Rights
access_id: a1	1	corba: gs-
access_id: a2	2	corba: g--
group: g1	1	corba: g--
group: g1	2	corba: gs-
group: g2	1	corba: gs-

Rights to privilege attributes are granted by an `AccessPolicy` object. An operation of a secure object can be invoked only when the principal possesses the set of rights prescribed by the `RequiredRights` object. Table 2 shows an example `RequiredRights` object that defines the rights required to gain access to each specific method of an object. There is also a mechanism to specify whether a user needs all the rights - in a method's required rights entry - to execute that method (AND semantics) or whether it is sufficient to match any right within the entry (OR semantics).

Table 2. Required rights

Required Rights	Rights Combinator	Operation	Interface (class)
corba: g--	all	M1	c_1
corba: g--	all	M3	
corba: gs-	all	M4	c_2
corba: -s-	all	M0	
corba: -s-	all	M2	
corba: gs-	any	M5	c_3

Table 3. Domain memberships and object classes

Object	Domain
o_1 , o_2, o_5, o_{12}	d_1
o_8, o_9	d_2

Objects	Class
o_1 , o_8	c_1
o_2 , o_5, o_9	c_2
o_{12}	c_3

Table 3 specifies the security domain memberships and the object classes, for the case access control introduced in Tables 1 and 2.

The `AccessDecision` object determines the validity of invocation requests based on the privilege and control attributes provided by the `AccessPolicy` and `RequiredRights` objects. There are no explicit rules on how to calculate the access decision: CORBA Security does not prescribe how an `AccessPolicy` object combines rights granted by different privilege attribute entries (when a subject has more than one privilege attribute to which the `AccessPolicy` grants rights). Taking into account the absence of policy composition rules for domain hierarchies, all these make feasible the implementation of different access decision functions and thus allow for potentially unlimited flexibility in security policy specification.

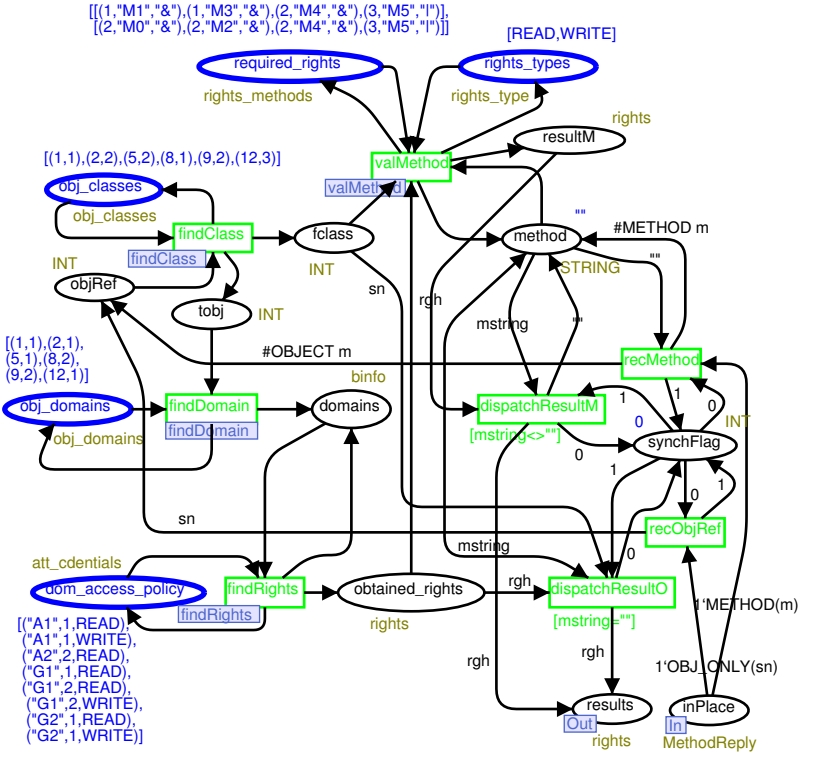


Fig. 2. The access decision CP-net submodel

Figure 2 presents the top-layer of the CP-net submodel implementing the following access decision function: "A method *m* can be executed if the requester's rights match the rights specified in the method's entry in the `RequiredRights` table". The shown CP-net is used in the following ways:

- To obtain privilege attributes and access rights (output place `results`) to proceed to the execution of the method specified in the union typed place `inPlace` (if any).

- To derive the *access control list* (list of privilege attribute and access right pairs in output place `results`) for the object specified in place `inPlace` (if any).

The domain access policy (bold place `dom_access_policy`) is specified as a list of triads, which respectively represent privilege attribute, domain number and right. The required rights table is given as lists of triads (bold place `required_rights`), which respectively represent class number, method name and rights combinator and each ML list refers to the corresponding right of the ML list shown in the `rights_types` bold place. The data shown in Table 3 determine the initial markings of the bold places `obj_domains` and `obj_classes`.

Due to space limitations we omit the description of the low-level CP-nets implementing the hierarchically related substitution transitions `valMethod`, `findClass`, `findDomain` and `findRights`.

3.2 The Information Flow Security Model

The CP-net of Figure 2 corresponds to the `protSys` substitution transition of Figure 3 that mimics a method execution: an object sends the messages of method's sequence specification (given in the bold input/output place `obj`) to itself or to other objects. Access to the object's state is accomplished by dispatching primitive read and write messages to itself: each of them is supposed to be executed synchronously.

In synchronous and deferred synchronous communication (hierarchically related substitution transitions `doSynchSend` and `doDSynchSend`) a reply is eventually returned, together with a list of all object identifiers (color `binfo` for the place `AOsL`) where read operations were allowed by the used access control. This list is termed as *Accessed Objects List* (AOsL). AOsL list is transmitted forward (requests) and backward (replies) as prescribed by methods' message sequence specifications, in order to record the performed read operations in all accessed objects.

An information flow to an object takes place only when information is written to it (substitution transition `doWrite`). In that case, there is an information flow from each one of the objects contained in the transmitted AOsL list. However, *not all of them violate the applied access control*:

Definition 3.1
An information flow from an object o_i (source) to an object o_j (target) *is not secure* (may cause undesirable information leakage), if the privilege attributes that grant read access to the target are not a subset of the set of attributes, which grant read access to the source.

Definition 3.2
An information flow to an object o_j *is secure*, if the privilege attributes that grant read access to it are also contained in all sets of privilege attributes, which grant read access to the objects contained in the transmitted AOsL list.

Figure 4 summarizes the color, variable and function declarations used for the transition and arc inscriptions of the CP-nets of Figures 3 and 5.

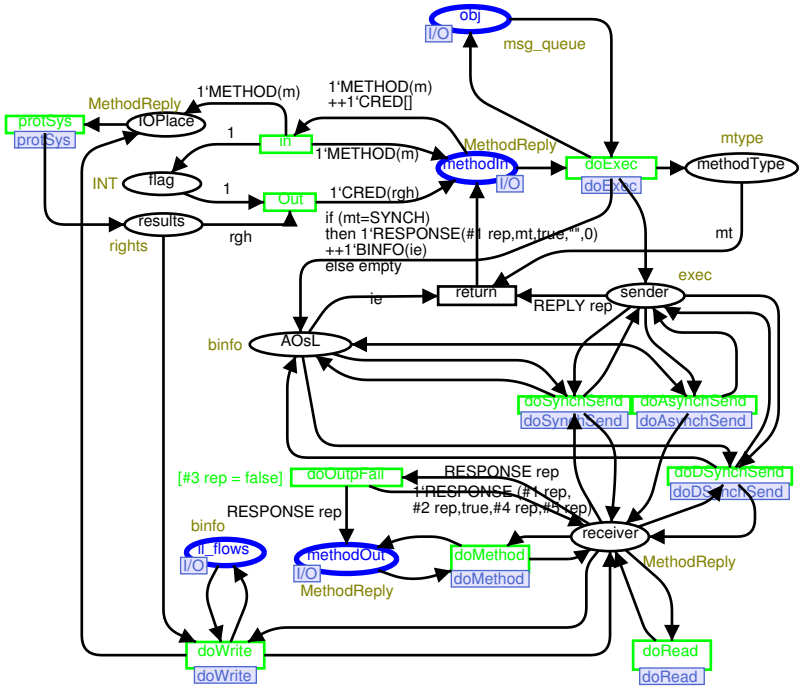


Fig. 3. The top layer of the method execution CP-net

```
color mtype                 = with SYNCH | ASYNCH | DSYNCH | DREP;
color crtype                = with READ | WRITE;
color rights_type           = list crtype;
color INT                   = int;
color BOOL                  = bool;
color STRING                = string;
color MSG_SNxSTATUS         = product INT * STRING;
color msg_rec               = record NUMBER:INT * METHOD:STRING
                              * TYPE:mtype * OBJECT:INT;
color right                 = product STRING*crtype;
color rights                = list right;
color binfo                 = list INT;
color msg_queue             = list msg_rec;
color reply                 = product INT * mtype * BOOL * STRING * INT;
color strLst                = list STRING;
color MethodReply           = union   RESPONSE:reply+
                              METHOD:msg_rec+
                              CRED:rights+
                              ACL:strLst+
                              BINFO:binfo+
                              OBJ_ONLY:INT;
color exec                  = union REPLY:reply +
                              MSG_QUEUE:msg_queue +
                              CREDENT:rights;

var q,p, messages           : msg_queue;
var m                       : msg_rec;
var sn,sm,k                 : INT;
var mt                      : mtype;
var rep                     : reply;
var ie,ic                   : binfo;
var rgh                     : rights;

fun aux2 k l                = if cf(k,l)>0 then nil else [k];
fun unio (x::xl,yl)         = (aux2 x yl)++unio(xl,yl)
                            | unio (_,yl) = yl;
```

Fig. 4. Colors, variables and functions used in CP-net incriptions

Figure 5 reveals the details of the `doSynchSend` substitution transition of Figure 3. Method execution is blocked (place `RmvMessage`) up to the reception of the expected reply - signaled by the token `(sn,"REPLIED")` in place `NextSend`. AOsL list is returned (arc inscription `1'BINFO(ic)`) together with the method reply. Then, AOsL list is updated as appropriate (function inscription `unio(ie,ic)` to calculate the union of the existing list `ie` and the received list `ic`).

We omit the details of the `doAsynchSend` and `doDSynchSend` transitions shown in Figure 3, but we stress the fact that AOsL list is never changed as a result of an asynchronous method execution.

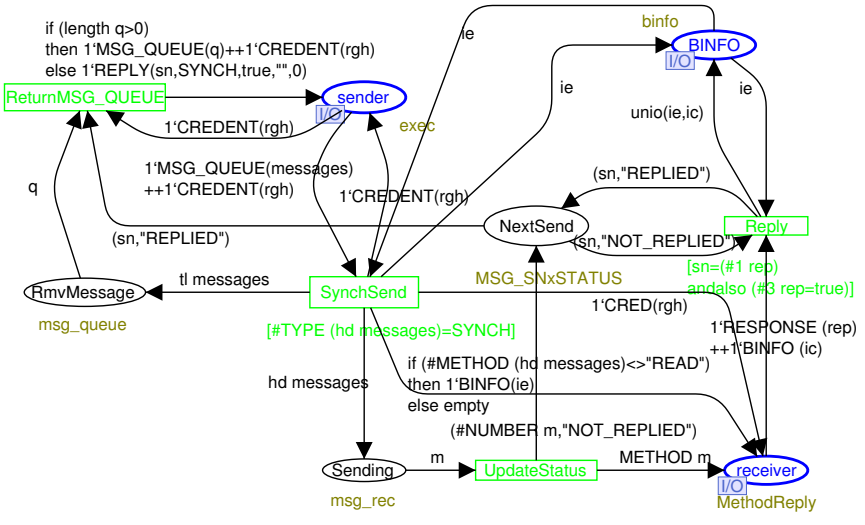


Fig. 5. The CP-net for the `doSynchSend` substitution transition shown in Figure 3

A system model is composed of a number of interacting instances of the CP-net shown in Figure 3 and each instance represents a particular method execution. For all method executions their instance input places (`obj`, `methodIn`) are initialized as imposed by the system's object method dependencies. Insecure information flow paths are detected at the `doWrite` substitution transition and for each object are separately recorded at the `il_flows` output places.

Figure 6 reports the simulation results given for the access control of Figure 2 and the case system model:

$$MsgSeq(M0^{o_2}) = \{(M1^{o_1}, s) \Rightarrow (M2^{o_9}, s) \Rightarrow (M3^{o_8}, s) \Rightarrow (write, s)\}$$
$$MsgSeq(M1^{o_1}) = \{(read, s)\}$$
$$MsgSeq(M2^{o_9}) = \{(M4^{o_5}, s) \Rightarrow (write, s)\}$$
$$MsgSeq(M3^{o_8}) = \{(read, s)\}$$
$$MsgSeq(M4^{o_5}) = \{(read, s) \Rightarrow (write, s)\}$$

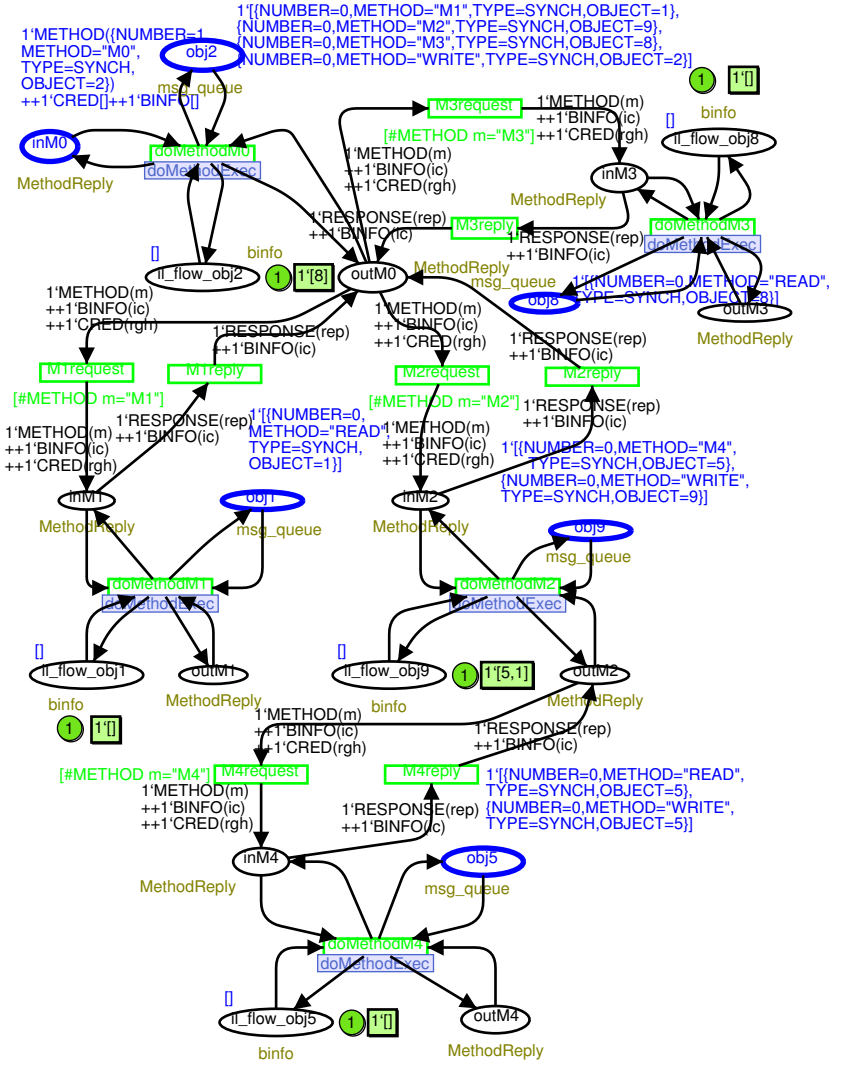


Fig. 6. A case system model and the detected insecure information flow paths

Insecure information flow paths are detected and recorded in `il_flow_obj2` and `il_flow_obj9` places. We observe the existence of flows from o_1 and o_5 to o_9 and from o_8 to o_2. We note that method execution control flow, like for example conditional execution, is not taken into account. We are interested for the detection of all potential information flow paths and we want to take them into account, in order to ensure compliance with business constraints regarding undesirable leakage or inference of sensitive information.

The simulated model reports the verification results regarding the detection of insecure flow paths. However, this is not enough if we want to use it as a mean to guide the specification of object classification constraints.

4 State Space Analysis

Analysis of the occurred information flow paths is performed after having generated all possible states that the system can reach. This can be done, by exploiting the CPN Tools state space analysis facilities ([7]).

Given the full state space, also known as *occurrence* or *reachability graph*, we can check the standard properties mentioned in section 3, as well as the existence of an *occurrence sequence* (reachability) to a particular marking (state). Figure 7 summarizes the results for the standard checks mentioned in section 3. The full state space is generated in 13 secs and consists of 2029 nodes (states) and 3282 arcs. There are no live transition instances and infinite occurrence sequences and the single dead marking that reflects the completed execution of method "M0" corresponds to the node number 2029.

```
Statistics
------------------------------------------------
  Occurrence Graph
    Nodes:  2029
    Arcs:   3282
    Secs:   13
    Status: Full

 Boundedness Properties
------------------------------------------------
  Best Integers Bounds    Upper      Lower
  NewPage'il_flow_obj1 1  1          1
  NewPage'il_flow_obj2 1  1          1
  NewPage'il_flow_obj5 1  1          1
  NewPage'il_flow_obj8 1  1          1
  NewPage'il_flow_obj9 1  1          1
  . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . . . . . . . . . . . .

 Home Properties
------------------------------------------------
  Home Markings:  [2029]

 Liveness Properties
------------------------------------------------
  Dead Markings:  [2029]
  Live Transitions Instances: None

 Fairness Properties
------------------------------------------------
No infinite occurrence sequences.
```

Fig. 7. The state space analysis standard report for the CP-net of Figure 6

Model querying regarding all detected information flow paths is performed through evaluation of simple ML functions. The set of predefined ML functions used to explore the generated state space is summarized in Table 4.

Table 5 shows (in the result column) the information flow data derived for the case system model.

Table 4. State space querying functions

function description	use
`Mark.<PageName>'<PlaceName> N M`	Returns the set of tokens positioned on place <PlaceName> on the Nth instance of page <PageName> in the marking M
`ListDeadMarkings ()`	Returns a list with all those nodes that have no enabled binding elements.
`hd l`	Returns the head of l.
`SearchNodes ( <search area>, <predicate function>, <search limit>, <evaluation function>, <start value>, <combination function>)`	Traverses the nodes of the part of the occurrence graph specified in <search area>. At each node the calculation specified by <evaluation function> is performed and the results of these calculations are combined as specified by <combination function> to form the final result. The <predicate function> maps each node into a boolean value and selects only those nodes, which evaluate to true. We use the value EntireGraph for <search area> to denote the set of all nodes in the occurrence graph and the value 1 for <start value> to continue the search until the first node, for which the predicate function evaluates to true.

Table 5. Information flow security queries

1. Insecure information flow sources:		
object id	function	result
o_2	Mark.NewPage'il_flow_obj2 1 (hd (ListDeadMarkings( )))	val it = [[8]]: binfo ms
o_1	Mark.NewPage'il_flow_obj1 1 (hd (ListDeadMarkings( )))	val it = [[]]: binfo ms
o_8	Mark.NewPage'il_flow_obj8 1 (hd (ListDeadMarkings( )))	val it = [[]]: binfo ms
o_9	Mark.NewPage'il_flow_obj9 1 (hd (ListDeadMarkings( )))	val it = [[5,1]]: binfo ms
o_5	Mark.NewPage'il_flow_obj5 1 (hd (ListDeadMarkings( )))	val it = [[]]: binfo ms
2. All information flow paths to o_2 (including the "secure" ones):		
function		result
Mark.doWrite'recBINFO 1 (hd (SearchNodes (EntireGraph, fn n => (Mark.doWrite'recBINFO 1 n <> empty andalso (Mark.doWrite'rstrrights 1 n <> empty andalso Mark.doWrite'torrights 1 n <> empty)), 1, fn n => n, [], op::)))		val it = [[1,5,8]]: binfo ms
3. Privilege attributes for read access to o_2:		
Mark.doWrite'torrights 1 (hd (SearchNodes (EntireGraph, fn n => (Mark.doWrite'recBINFO 1 n <> empty andalso (Mark.doWrite'rstrrights 1 n <> empty andalso Mark.doWrite'torrights 1 n <> empty)), 1, fn n => n, [], op::)))		val it = [["A1", "G1", "G2"]]: strLst ms
4. Privilege attributes for read access to insecure source o_8:		
Mark.doWrite'rstrrights 1 (hd (SearchNodes (EntireGraph, fn n => (Mark.doWrite'recBINFO 1 n <> empty andalso (Mark.doWrite'rstrrights 1 n <> empty andalso Mark.doWrite'torrights 1 n <> empty)), 3, fn n => n, [], op::)))		val it = [["A2", "G1"]]: strLst ms

The results of query 1 verify the simulation results of Figure 6 regarding the insecure flow paths detected at the found dead marking. The query is based on inspection of the marking of `il_flow` places on the first instance of page `NewPage`, for the head of the list of dead markings, which in fact contains the single dead marking with node number 2029.

In query 2 we use the function `SearchNodes` to explore the entire state space for a marking that yields all flows (including the "secure" ones) to o_2. Queries 3 and 4 reveal the details of the insecure flow (definition 3.1) sourced at o_8.

Function `SearchNodes` is used as a tool of potentially unlimited flexibility in querying the model regarding the "insecure" and the "secure" information flow paths, in order to direct the specification of object classification constraints. Alternatively, CPN Tools includes a library for defining queries in a CTL-like temporal logic.

State spaces grow exponentially, with respect to the number of independent processes. In the proposed model, this problem becomes evident, when using asynchronous and/or deferred synchronous method calls. From the alternative published analyses our model fits to the modular state space analysis described in [2]. The behavior of the entire system can be captured by the state spaces of the modules corresponding to individual method executions (Figure 6), combined with an appropriate synchronization graph. Unfortunately, CPN Tools does not currently support the generation of separate state space modules and the required synchronization graph. Thus, the application of the forenamed analysis approach remains an open research prospect.

5 Related Work

Information flow security is an active research problem that was first approached in 1973 ([10]) and that is still attracting the interest of researchers in a number of recently published works ([12], [1], [6], [4], [5]).

Recent research works in the context of distributed object systems ([6], [16] and [14])

- are based on different and often not realistic assumptions on when an information flow occurs,
- do not always take into account that in real systems, methods are invoked in a nested manner,
- are bound to specific role-based or purpose-oriented access control models and none employs the CORBA Security reference model or
- aim in the dynamic control of information flow by the use of an appropriate run-time support that in most systems is not available.

Our work (i) takes into account the bi-directional nature in the direct or indirect information transfer between the senders and the receivers, (ii) allows for modeling nested object invocations, (iii) employs the CORBA Security reference model and for this reason is not bound to a specific access control model and (iv) aims in the static verification of information flow security and for this reason does not assume proprie-

tary run-time support. Moreover, it is based on a widespread modeling formalism with an easily accessible advanced tool support.

Other interesting sources of related work are the introduction into lattice-based access control that was published in [15] and the mandatory access control that is specified in [9], by the use of the CORBA Security reference model.

6 Conclusion

In modern networked business information systems and in service-based business activities, where different customers are concurrently using the provided services, compliance with business constraints regarding undesirable inference of sensitive information is a central design issue.

The problem under consideration is addressed by the implementation of an appropriate multilevel mandatory policy. However, the rigidity of these policies implies the need for a systematic design approach. We introduced a Colored Petri Net model that simulates and detects "insecure" information flow paths according to the given definitions determining when a flow path is not secure The model can be queried regarding the existing ("insecure" and "secure") flows, in order to direct the specification of object classification constraints, wherever there is a need of them. This allows to avoid overclassification and thus to maximize information visibility.

Acknowledgments

We acknowledge the CPN Tools team at Aarhus University, Denmark for kindly providing us the license of use of the valuable CP-net toolset.

References

1. Chou, S.-C.: Information flow control among objects: Taking foreign objects into control, In: Proceedings of the 36th Hawaii International Conference on Systems Sciences (HICSS'03), IEEE Computer Society (2003) 335a-344a
2. Christensen, S., Petrucci, L.: Modular state space analysis of Coloured Petri Nets, In: Proceedings of the 16th International Conference on Application and Theory of Petri Nets, Turin, Italy (1995) 201-217
3. Dawson, S., Vimercati, S., Lincoln, P., Samarati, P.: Maximizing sharing of protected information, Journal of Computer and System Sciences 64 (3), (2002) 496-541
4. Georgiadis, C., Mavridis, I., Pangalos, G.: Healthcare teams over the Internet: Programming a certificate-based approach, International Journal of Medical Informatics 70 (2003) 161-171
5. Halkidis, S. T., Chatzigeorgiou, A., Stephanides, G.: A qualitative evaluation of security patterns, In: Proceedings of ICICS 2004, LNCS 3269, Springer-Verlag (2004) 132-144
6. Izaki, K., Tanaka, K., Takizawa, M.: Information flow control in role-based model for distributed objects, In: Proceedings of the 8th International Conference on Parallel and Distributed Systems (ICPADS'01), Kyongju City, Korea, IEEE Computer Society (2001) 363-370

7. Jensen, K.: An introduction to the practical use of colored Petri Nets, In: Lectures on Petri Nets II: Applications, LNCS, Vol. 1492 (1998) 237-292
8. Jensen, K.: An introduction to the theoretical aspects of colored Petri Nets, In: A Decade of Concurrency, LNCS, Vol. 803 (1994) 230-272
9. Karjoth, G.: Authorization in CORBA security, In: ESORICS'98, LNCS, Vol. 1485 (1998) 143-158
10. Lampson, B. W.: A note on the confinement problem, Communication of the ACM 16 (10), (1973) 613-615
11. Larsen, L., Harrold, M.: Slicing object oriented software, In: Proceedings of the 18th International Conference on Software Engineering (1996) 495–505
12. Masri, W., Podgurski, A., Leon, D.: Detecting and debugging insecure information flows, In: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04), Saint-Malo, Bretagne, France, IEEE Computer Society (2004) 198-209
13. Object Management Group: Security service specification, version 1.7, OMG Document 99-12-02 (1999)
14. Samarati, P., Bertino, E., Ciampichetti, A., Jajodia, S.: Information flow control in object-oriented systems, IEEE Transactions on Knowledge and Data Engineering 9 (4), (1997) 524-538
15. Sandhu, R. S.: Lattice-based access control models, IEEE Computer 26 (11), (1993) 9-19
16. Yasuda, M., Tachikawa, T., Takizawa, M.: Information flow in a purpose-oriented access control model, In: Proceedings of the 1997 International Conference on Parallel and Distributed Systems (ICPADS'97), Seoul, Korea, IEEE Computer Society (1997) 244-249

Appendix

In this section, we outline the formal semantics of CP-nets, as they are defined in [8].

Definition 1. A multi-set m, over a non-empty set S is a function $S \rightarrow \aleph$ represented as a sum

$$\sum_{s \in S} m(s)`s$$

By S_{MS} we denote the set of all multi-sets over S. The non-negative integers $\{m(s) | s \in S\}$ are the coefficients of the multi-set.

Definition 2. A Colored Petri Net (CP-net) is a tuple $CPN=(\Sigma, P, T, A, N, C, G, E, I)$ where:

(i) Σ is a finite set of non-empty types, also called color sets
(ii) P is a finite set of places (drawn as ellipses)
(iii) T is a finite set of transitions (drawn as rectangles)
(iv) A is a finite set of arcs
(v) N is a node function $A \rightarrow P \times T \cup T \times P$
(vi) C is a color function $P \rightarrow \Sigma$

(vii) G is a guard function that maps each transition $t \in T$ into a Boolean expression where all variables have types that belong to Σ:

$$\forall t \in T: \text{Type}(G(t)) = \mathcal{B} \wedge \text{Type}(\text{Var}(G(t))) \subseteq \Sigma$$

(viii) E is an arc expression function that maps each arc $a \in A$ into an expression that is evaluated in multi-sets over the type of the adjacent place p:

$$\forall a \in A: \text{Type}(E(a)) = C(p)_{MS} \wedge \text{Type}(\text{Var}(E(a))) \subseteq \Sigma, \text{ with } p = N(a)$$

(ix) I is an initialization function that maps each place $p \in P$ into a closed expression of type $C(p)_{MS}$:

$$\forall p \in P: \text{Type}(I(p)) = C(p)_{MS}$$

When we draw a CP-net we omit initialization expressions, which evaluate to $\varnothing$.

The set of arcs of transition t is

$$A(t) = \{a \in A \mid N(a) \in P \times \{t\} \cup \{t\} \times P\}$$

and the variables of transition t is

$$\text{Var}(t) = \{v \mid v \in \text{Var}(G(t)) \vee \exists a \in A(t): v \in \text{Var}(E(a))\}$$

Definition 3. A binding of a transition t is a function b defined on Var(t), such that:

(i) $\forall v \in \text{Var}(t): b(v) \in \text{Type}(v)$

(ii) The guard expression G(t) is satisfied in binding b, i.e. the evaluation of the expression G(t) in binding b - denoted as G(t)<b> - results in true.

By B(t) we denote the set of all bindings for t.

Definition 4. A token element is a pair (p, c) where $p \in P$ and $c \in C(p)$. A binding element is a pair (t, b) where $t \in T$ and $b \in B(t)$. The set of all token elements is denoted by TE and the set of all binding elements is denoted by BE.

A marking is a multi-set over TE and a step is a non-empty and finite multi-set over BE. The initial marking M_0 is the marking, which is obtained by evaluating the initialization expressions:

$$\forall (p,c) \in TE: M_0(p,c) = (I(p))(c)$$

The set of all markings and the set of all steps are denoted respectively by M and Y.

For all $t \in T$ and for all pairs of nodes $(x_1, x_2) \in (P \times T \cup T \times P)$ we define

$$A(x_1, x_2) = \{a \in A \mid N(a) = (x_1, x_2)\} \text{ and } E(x_1, x_2) = \sum_{a \in A(x_1, x_2)} E(a)$$

Definition 5. A step Y is enabled in a marking M if and only if

$$\forall p \in P: \sum_{(t,b)\in Y} E(p,t) < b > \leq M(p)$$

We then say that (t,b) is enabled and we also say that t is enabled. The elements of Y are concurrently enabled (if $|Y| \geq 1$).
When a step Y is enabled in a marking M_1 it may occur, changing the marking M_1 to another marking M_2, defined by:

$$\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t) < b >) + \sum_{(t,b)\in Y} E(t,p) < b >$$

M_2 is directly reachable from M_1.

Offline Business Objects: Enabling Data Persistence for Distributed Desktop Applications

Pawel Gruszczynski, Stanislaw Osinski, and Andrzej Swedrzynski

Poznan Supercomputing and Networking Center,
Poznan Noskowskiego 10, 61-704, Poland
{grucha, stachoo, kokosz}@man.poznan.pl

Abstract. Despite the overwhelming popularity of web-based distributed systems, in certain areas the traditional desktop applications still seem to be a better choice. In this paper we examine the challenges involved in developing a secure and fault-tolerant data persistence layer for distributed desktop applications. We analyse the currently existing persistence frameworks and show why they do not meet the requirements set by certain distributed applications. The body of this paper concentrates on the Offline Business Objects framework which aims to fill this gap. The framework introduces the offline operation paradigm, whereby client applications operate without a permanent server connection and only periodically synchronise their data with the central database. We successfully deployed a distributed information system based on this paradigm and the Offline Business Objects for five major cities in Poland, which confirmed the practical value of our approach.

1 Introduction

The increasing availability of broadband networking infrastructure is an attractive incentive for software architects to abandon the traditional desktop application model of software delivery in favour of the thin client Application Service Provision [7]. One reason for this is that a great majority of today's distributed application frameworks, e.g. data persistence frameworks, have been designed with web-based systems in mind and cannot be directly used in desktop software.

In areas, however, where important are such factors as high security standards, fault-tolerance or responsive user interfaces, the desktop application model is still a viable approach [1]. Unfortunately, with the lack of adequate software frameworks, the main challenge involved in developing a secure and fault-tolerant distributed desktop application becomes the implementation of a suitable data access layer.

In this paper we describe the design and implementation of the Offline Business Objects (OBO) framework, which aims to fill the gap in data persistence for distributed desktop applications. In section 2 we briefly describe the High School On-line Admission System called *Nabor*—an application that defined the requirements for OBO. In section 3 we analyse the currently available data persistence frameworks and explain why they cannot be directly integrated with

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 960–977, 2005.

distributed desktop applications. Section 4 presents the design and implementation of Offline Business Objects, emphasising problems that are not addressed in the currently available frameworks, while section 5 briefly evaluates the framework. Finally, section 6 concludes the paper and gives directions for future work.

2 Requirements

The requirements for Offline Business Objects framework were driven to a large extent by the requirements for the High School On-line Admission System called *Nabor*[1], which we developed and deployed for five major Polish cities in 2003 and 2004 [13]. The main objective of *Nabor* was to provide high enough a level of co-ordination between all high schools in a city as to reduce the time and effort involved in the admissions process. The key idea was to gather all necessary data (e.g. candidates' marks, admission limits set by schools) in a central database and design an algorithm that would automatically generate admission lists for all schools in the city.

To comply with legal regulations governing high school admissions, the architecture of *Nabor* had to meet a number of technical requirements:

High security standards. A great majority of data processed by *Nabor* (e.g. candidates' marks) was of a confidential character and had to be carefully protected from unauthorised access and modification. Also, the system had to be designed in such a way as to minimise the possibility of manipulating the admission results and to enable tracing possible manipulation attempts.

Fault-tolerance. High school admissions procedures divide the whole process into several phases and impose strict deadlines on each of them. For example, one of the final phases is entering the candidates' marks to the system, which must be completed by all schools within a time span of two or three days. Therefore, *Nabor* had to be able to operate even in case of a temporary failure of the central database or the network infrastructure.

Varied user group. An important characteristic of *Nabor* was also the fact that it required active involvement of all parties of the admissions process. High school candidates used the system to find out about the offered schools, fill in and print out the application form. School administration used *Nabor* to enter the candidates' data and download the admission lists once the admissions had been closed. Finally, for the local education authorities the system provided a range of analytical reports, which aided global planning.

Having analysed the possible architectural designs [13], we decided that in order to fulfill all the requirements, *Nabor* should be implemented as a client-server system, with client nodes running offline desktop applications. In this setting the client application would normally stay disconnected from the central server and perform all operations on locally cached data. Occasionally, a connection would be established to synchronise the local data cache with the central database.

[1] *Nabor* is the Polish word *admissions*.

We also made a decision to implement the *Nabor* system using Java technology. Employing Java Swing [18] on the client side would make the desktop application platform-independent, while using non-proprietary Java technologies on the server side would reduce the cost of the whole undertaking.

As a result of the above requirements and architectural decisions, the underlying data persistence layer would have the following responsibilities:

Object-relational mapping. Performing mapping between the relational and object-oriented representation for all business objects in the system.
Offline operation. Supporting remote reading and editing of business objects on client nodes in the absence of the server link.
Data synchronisation. Synchronising client's local changes with the database maintained on the server side.
Logging. Automatic logging of all client operations on the server side.
Security. Ensuring high standards of data security.

In section 3 we explain why none of the currently available Java frameworks was suitable as a persistence layer for *Nabor*, and in section 4 we describe how Offline Business Objects implements the desired mechanisms.

3 Existing Persistence Frameworks

3.1 Introduction to Object Relational Mapping

Contrary to earlier expectations and predictions, relational databases are still the primary technology for data persistence. Although the alternative approaches, such as object database systems, are gaining more and more popularity, it is still relational or object-relational databases that prevail.

Implementing a large information system based on simple relational database manipulation interfaces, such as JDBC [19] for Java, can prove a daunting and error-prone task. This is especially true in case of object-oriented technologies, where a significant amount of application code would deal not with implementing business logic, but with object-relational conversions.

This is one of the reasons why Object Relational Mapping (ORM) technologies started to emerge [11]. ORM can be defined as a mechanism for automated persistence of objects (e.g. Java objects) to the tables of a relational database. Essentially, ORM can be thought of as a means of transforming data from the object-oriented to the relational representation and *vice versa* [3, 10]. Most modern ORM tools perform the transformation fully transparently, i.e. any changes made to a persistent business object, such as setting a new value for its instance field, are automatically reflected in the database without a single persistence-related line of code on the business object's side.

The most popular Java ORM tools are currently Enterprise Java Beans (EJB) [16], Java Data Objects (JDO) [17] and Hibernate [2]. In the remaining part of this section we will use Hibernate as a representative example of an ORM tool. The reason for this is that JDO is in essence very similar to Hibernate and the recently released EJB 3.0 draft specification is also largely influenced by the experiences with Hibernate.

3.2 ORM and Desktop Applications

As we pointed out earlier, the majority of currently available data persistence frameworks, including Hibernate, have been designed and implemented with web-based applications in mind. For this reason integrating ORM tools with a desktop application, such as a Java Swing application in case of *Nabor*, is not a straightforward task.

A naive approach could be to use the ORM framework on the client side. This would require that the database manipulation interface (e.g. JDBC in case of Java) be directly exposed so that it can be accessed by client nodes. For obvious reasons this approach introduces high security risks, as the database connection could be used in a trivial way to read, modify or delete data. Although in many scenarios, such as an intranet application with trusted users, the naive scheme would be perfectly sufficient, it was not acceptable for the purposes of *Nabor*.

The alternative approach is to integrate the ORM framework into the server software and create a special layer of communication between the client nodes and the server. In this way, the direct database connection would not need to be exposed. On the other hand, the introduction of an additional communication layer would mean losing or re-implementing a number of useful features the ORM tools can offer. In case of Hibernate these features would be e.g.:

Unambiguous representation. Hibernate guarantees that querying the database for the same object many times within the same session will always return the same instance of a Java class.

Modification detection. Hibernate automatically detects which objects have been modified and sends to the database only the data that needs updating.

Yet another possibility would be to use ORM on the client side with a local off-line database and periodically synchronise the database with the central server. One major difficulty with this approach is the lack of freely available software frameworks for database synchronisation. Moreover, in some applications each client should have access to only a small part of the system's business objects, which can be enforced more flexibly by synchronising and authorising individual operations rather than the whole database. Finally, as client workstations are usually significantly less powerful than server machines, this approach may not be feasible for systems managing large numbers of business object instances.

3.3 ORM and the Offline Operation Model

As we mentioned earlier, the key characteristic of an application implemented in the offline operation model is that it does not maintain a permanent link to the server. Instead, all client-side operations are performed on locally cached data which is then periodically synchronised with the server. The possible advantages of the offline model are better fault-tolerance and lower bandwidth usage. The biggest problem with this model, on the other hand, is that none of today's ORM frameworks fully supports offline operation. Therefore, in order to achieve the offline functionality, additional components must be built including a local data cache and some data synchronisation layer.

The responsibility of the local data cache component would be to ensure read and write access to business objects in the absence of the server link. Ideally, the data cache would have properties similar to those known from relational databases, namely: atomicity, consistency, isolation and durability, together referred to as ACID [12]. One way of achieving these properties is implementing the local cache using a lightweight embedded relational database engine, such as the Java-based HypersonicSQL [15]. An alternative approach would be based around the concept of Prevayler [21], which implements persistence by keeping all data in memory, logging every operation before it is executed, storing periodic snapshots of all data and using these snapshots together with operation logs to recover system state.

The task of the data synchronisation component is to maintain consistency between the contents of the client's local cache and the global database managed by the the server [4]. The main problem here is to properly resolve conflicting modifications of the same business object performed locally by disconnected clients. A general solution would require taking into account the complex semantics of business objects and thus might be quite costly to implement. Alternatively, specific characteristics of a concrete application can be exploited in order to simplify the problem. In case of *Nabor*, for example, for a large number of business objects there would be only one client authorised to modify them. Later in this paper we show how we took advantage of this property. Another important requirement for the data synchronisation component is making the communication efficient in terms of the volume of data transferred and guaranteeing that data consistency will not be violated as a result of e.g. a communication layer failure.

Although Hibernate does not explicitly support the offline operation mode, it offers a mechanism of *detached objects* which can prove helpful in implementing the local cache and data synchronisation components. With this mechanism, every persistent data object (i.e. a Java object stored in a database by Hibernate) can be temporarily detached from Hibernate's data persistence layer, serialised and transmitted between different system layers, and then re-attached back to Hibernate.

In late 2003, when the development of the *Nabor* system was already underway, a specification of a standard called Service Data Objects [6] was announced, which aimed at simplifying and unifying the ways in which applications handled data. Among others, SDO defines a *disconnected programming model*, which seems to be the counterpart of the offline operation model required for *Nabor*. Unfortunately, no implementations of SDO were available at that time, hence the need for the Offline Business Objects we describe in detail in the next section.

4 Offline Business Objects

The purpose of Offline Business Objects is to provide a reliable and secure persistence layer for offline desktop applications. Its main responsibilities are:

- object-relational mapping of business data
- offline access to business data
- client-server synchronisation

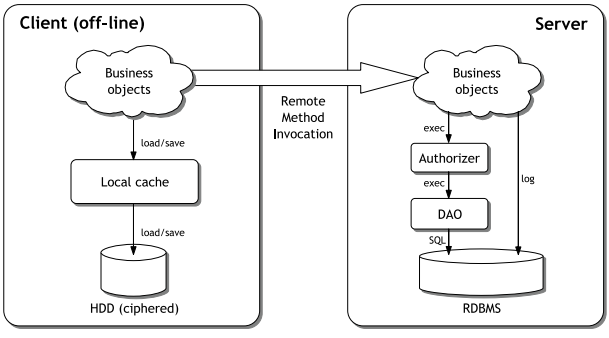


Fig. 1. The architecture of Offline Business Objects

- authorisation of client operations
- logging of client operations
- ensuring high standards of data security

In the following subsections we describe how client- and server-side components of OBO shown in Figure 1 cooperate to implement the required mechanisms.

4.1 Object-Relational Mapping

The task of the object-relational mapping component is to translate business data between the relational and object-oriented models. As this part of OBO would reside on the server side, it's implementation could almost entirely rely on one of the existing ORM tools. We decided to implement the object-relational mapping component using the Torque framework[2] [8].

Torque is based on the idea of so-called Partial Class Generators [14], where a declarative XML specification of object-relational mapping is used as an input for automatic generation of base classes for business objects and data access objects (DAO). Specific behaviour of business objects can be implemented by subclassing the automatically generated classes.

For the following example specification:

```
<table name="PERSON">
  <column name="PERSON_ID" required="true"
          primaryKey="true" type="INTEGER"/>
  <column name="FIRST_NAME" required="true"
          size="50" type="VARCHAR"/>
  <column name="LAST_NAME" required="true"
          size="50" type="VARCHAR"/>
</table>
```

[2] The alternative choice was Hibernate [2], but at the time of designing OBO (mid 2003) Hibernate was not as popular as it is now and we felt we should not yet use it on production scale.

the following artifacts will be generated:

BasePerson. A business object base class with appropriate getter and setter methods. Business-specific behaviour can be implemented by subclassing this class.

BasePersonPeer. A class that handles reading and writing of the business object's data to a relational database.

SQL DDL statements. SQL Data Definition Language statements that can be used to create all database structures (e.g. tables, sequencers) required for persisting the business objects.

4.2 Offline Operation and Client-Server Synchronisation

The offline operation and client-server synchronisation component was by far the most difficult to design and implement part of OBO. It involved both the client-side local cache and the server-side synchronisation and persistence mechanisms.

Local Cache. The purpose of the local cache is to provide read and write access to business objects in the absence of the server connection. To that end, business data is stored on the client's workstation in an encrypted form. The storage mechanism is similar to this used in Prevayler [21]. All data is kept in memory, with each client's operation being logged before execution. Additionally, the system takes periodic snapshots of all data and stores them on the local disk. Local system state recovery is based on the freshest snapshot and the log of operations performed after that snapshot was taken.

An interesting design problem related to the local cache was how and when to assign unique identifiers to business objects. There are two possibilities here:

- a unique identifier is created on the client side at the time of creating or storing of the business object in the local cache
- a unique identifier is assigned on the server side during client-server synchronisation

In a naive implementation of the latter method, the client application could transactionally synchronise its local changes with the server and as a result receive the identifiers of newly created business objects. Such an approach, however, has a subtle flaw connected with fault-tolerance. Suppose a transaction has been successfully committed on the server side, but due to a failure of the client-server link, the client node did not receive the identifiers of newly created objects. In this situation, the client application will have no way of knowing that a number of its local business objects that do not yet have identifiers correspond to some server-side objects for which unique identifiers have already been assigned.

One way of solving this problem would be to extend the communication protocol with additional acknowledgment messages, e.g. a "ready to commit" message sent by the client application after the identifiers have been received

and applied to the locally cached data. However, in order to simplify the communication, we have decided that the identifiers should be generated on the client side. Global uniqueness of locally assigned identifiers is achieved by dividing the global pool of identifiers into extendable subsets used exclusively by one client. If a client reports that it has used a certain number of identifiers from its set (e.g. 80%), the server will extend the identifier set for that client.

Business Object Lifecycle. Designing a data persistence framework requires a decision as to how the lifecycle of business objects should look like, i.e. in what states an individual object can be and how they relate to the client-server communication. A starting point here can be the object lifecycle defined in one of the existing persistence frameworks, such as Hibernate or JDO. With the additions required by the offline processing model, the list of states of an individual business object can be the following (Figure 2):

Transient. An object is Transient right after it has been created, e.g. by a call to a constructor method.
Persistent New. A Transient object becomes Persistent New when it gets stored in the local cache, and has not yet been sent to the server database.
Persistent Clean. A business object becomes Persistent Clean after it has been synchronised with the server.
Persistent Dirty. A Persistent Clean object becomes Persistent Dirty after it has been locally modified.
Deleted. A business object that has been locally deleted.

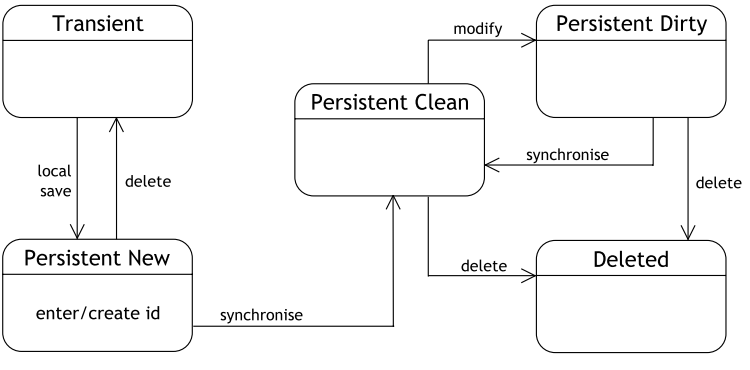


Fig. 2. Offline Business Objects initial life cycle

The state of an individual business object determines the actions required to synchronise this object with the central database. During data synchronisation the client application sends to the server the contents of objects that are Persistent New or Persistent Dirty, and also identifiers of objects that are Deleted. On the server side, the object's state can be used to choose the appropriate SQL

operation to perform: *insert* for Persistent New objects, *update* for Persistent Dirty objects, and *delete* for Deleted objects.

It turns out, however, that for enhanced fault-tolerance we need to introduce one more state to the lifecycle of a business object. To illustrate this, let us assume that the client application has created a new business object and stored it in the local data cache as Persistent New. During the next client-server synchronisation, the new object should be sent to the server and its state should become Persistent Clean. The major problem here is that committing changes on the server side and changing object's state on the client should be one atomic operation. With the initially proposed lifecycle, the client could change the object's state to Persistent Clean either "right before" it sends the changes to the server, or "right after" the server acknowledges that the data has been successfully committed to the database.

Neither of these variants, however, is perfect with respect to fault-tolerance. If a communication layer failure occurs, the client's information about the states of locally cached business objects may become inconsistent with what is stored on the server side. With the "right before" strategy, the client application may mark an object as Persistent Clean even though the server did not commit a corresponding change to the database. With the "right after" variant, on the other hand, if the server acknowledgment is not received on the client side, the state of some objects will not be changed to Persistent Clean even though it should. As we can see, neither of the above strategies guarantees correct information about the objects' state on the client side. Consequently, the server cannot rely on this information received from the client during the subsequent synchronisations either.

A solution to the above problem can be enhancing the business object's lifecycle with an additional intermediate state called Persistent Possibly New shown in Figure 3. In this way, during the client-server synchronisation a Persistent New object is sent to the server and then its local state is changed to Persistent Possibly New. Only after the server acknowledges that the transaction has been successfully committed, a Persistent Possibly New object becomes Persistent Clean. Otherwise, if a communication failure occurs, the object remains Persistent Possibly New and will be sent in that state to the server during the next data synchronisation. On the receipt of a Persistent Possibly New object, the server will need to do an additional *select* operation to check if corresponding data is already present in the database.

A similar type of problem can arise with transitions between the Persistent Dirty and Persistent Clean states. In this case, however, there is no need for intermediate states and the transition can take place right after the server acknowledges a successful completion of the transaction. If the client application does not receive the acknowledgment because of a communication layer failure, it will send the modified object to the server once again during the next synchronisation.

Data Synchronisation. The purpose of data synchronisation is to ensure consistency between the desktop application's local cache and the global database

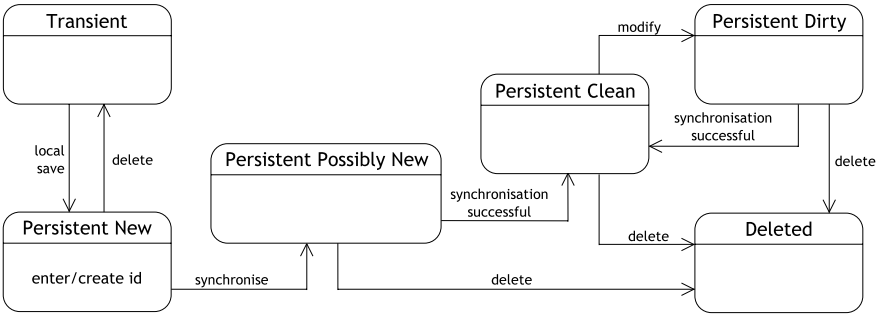


Fig. 3. Offline Business Objects modified life cycle

on the server. While designing OBO we decided that data synchronisation should be performed in two phases. Phase one is applying client's local changes to the global database while phase two is updating client's local cache with modifications made by other clients.

In the first phase the client application sends to the server a log of locally cached operations. Based on the current state of a business object, different actions will be taken:

Create object. For a Persistent New object, the client application will send the contents of that object (values of attributes, identifiers of collection members) to the server. The server will add corresponding data to the database

Delete object. For a Deleted object, the client application will send the identifier of that object to the server. The server will delete the corresponding data from the database.

Update attribute value. For a Persistent Dirty object with modified attributes, the client application will send to the server the identifier of that object and name/new value pairs for all modified attributes. The server will modify the corresponding values in the database.

Update collection. For a Persistent Dirty object containing a modified collection of business objects, the client application will send identifiers of objects added and removed from that collection. The server will make corresponding changes to the database. Noteworthy is the fact that this operation modifies the *relationship* between business objects, but not the objects themselves.

Crucial to the second phase of data synchronisation is versioning of business objects [20]. OBO implements this mechanism by assigning an integer version number to each of the business objects managed by the system. Every time the server processes an attribute update operation, the version number of the involved business object is incremented. For a collection update increased is only the version number of the business object that contains the collection, version numbers of collection members remain unchanged.

The client applications initialises the second phase of data synchronisation by sending to the server the identifier/version number pairs of all locally cached

objects. Then, for each business object the server compares its version number V_S of that object with the corresponding version number V_C sent by the client. Based on that comparison, the server can take the following actions:

When $V_C < V_S$. The client's version of the business object is older than the server's version. The up-to-date object needs to be sent to the client.

When no V_C corresponds to V_S. The client has not yet received any information about the business object. The object needs to be sent to the client.

When no V_S corresponds to V_C. The client has an object that has been deleted some time after the last synchronisation. Information that the object has been deleted needs to be sent to the client.

When $V_C = V_S$. The client's version of the business object is up-to-date. No data needs to be sent. This is the case where versioning can bring a substantial decrease in the volume of data transferred during synchronisation.

An interesting problem related to data synchronisation is how the system should deal with concurrent modifications of business objects. Situations in which the same version of an object is modified by two or more clients are far more probable in an offline processing model. With Offline Business Objects this problem can be solved in two different ways. One method is to accept the first modification of a certain version of an object and reject all subsequent modification attempts related to that version. Users whose modifications have been rejected will be notified of the fact, and will need to explicitly cancel either all local operations performed since the last the synchronisation or only the conflicting ones. In this way, the first accepted modification will effectively overwrite the remaining concurrent modifications[3]. The alternative method is to execute all modifications in the order they were received by the server. This method, on the other hand, means that the last concurrent modification will overwrite the earlier ones. In this case, no special interaction with the user is needed. The choice as to which of these methods OBO should use can be made for each database table separately.

Using the former conflict resolution strategy (*first wins*) in *Nabor* would require the end users to make decisions related to the application's underlying communication protocols, which might be both misleading and frustrating for them. On the other hand, due to the characteristics of *Nabor*, lost updates would occur fairly rarely and would not result in any major inconsistencies. For this reason, in our real-world application, we used the *last wins* strategy for all kinds of business objects.

[3] The exact sequence of operations would be the following: the server accepts the first modification of an object and advances the object's version number from $V_S = i$ to $V_S = i + 1$. On the receipt of another modification related to version $V_S = i$, the server rejects the modification and instructs users to cancel the corresponding operations. Upon the next synchronisation, the client will receive the $V_S = i + 1$ version of that object, which will effectively overwrite the (deleted) local change made to version $V_C = i$.

4.3 Authorisation

One of the requirements for the Offline Business Objects framework was that each modification of data attempted by the client's desktop applications should be authorised. OBO implements a model whereby the client's credentials are verified on the server side during data synchronisation. For maximum flexibility, OBO performs authorisation on the level of Java code before business objects are read or written to the database (see Figure 1).

The details of authorisation can be specified separately for each database table managed by OBO in a declarative way, e.g.:

```
<table name="TEAM" authorizationMethod="hasCoachPrivileges"/>
```

With the above declaration, before a modification of a Team object is committed to the database, the `hasCoachPrivileges` method (available in a globally accessible credentials object) is called to verify if the client is allowed to perform that modification.

More interesting is the problem of authorising operations involving interrelated tables, e.g. master-detail tables:

```
<table name="PLAYER" authorizationMethod="hasCoachPrivileges">
  <references foreignTable="TEAM" authorizationPropagates="true"/>
</table>
```

In this case, before the server applies any modifications to the Player object, it will check whether the client is allowed to modify the Player's Team. Into account will be taken both the current Team of the Player and also the Team to which the Player is to be transferred. Moreover, OBO will group operations involving different Players, so that the data of all required Teams can be fetched with a single database query.

4.4 Data Security

The security model we implemented in Offline Business Objects is based on the Public Key Infrastructure (PKI) and the RSA cryptography [22]. During the design phase we decided that each client operation, such as creating or modifying a business object (see section 4.2), should be transmitted to the server as a separate encrypted packet.

Figure 4 schematically shows all cryptographic operations involved in transmitting a single operation packet from the client's desktop application to the server. In the absence of the server connection, all application data is stored in an encrypted form in the local data cache. For the purposes of local cache protection OBO uses symmetric cryptography with a client-specific secret key distributed together with the desktop application.

During data synchronisation, for each cached client operation a separate packet is created. Each such packet is then encrypted with a dedicated symmetric key generated only for that packet. The symmetric key itself is encrypted with

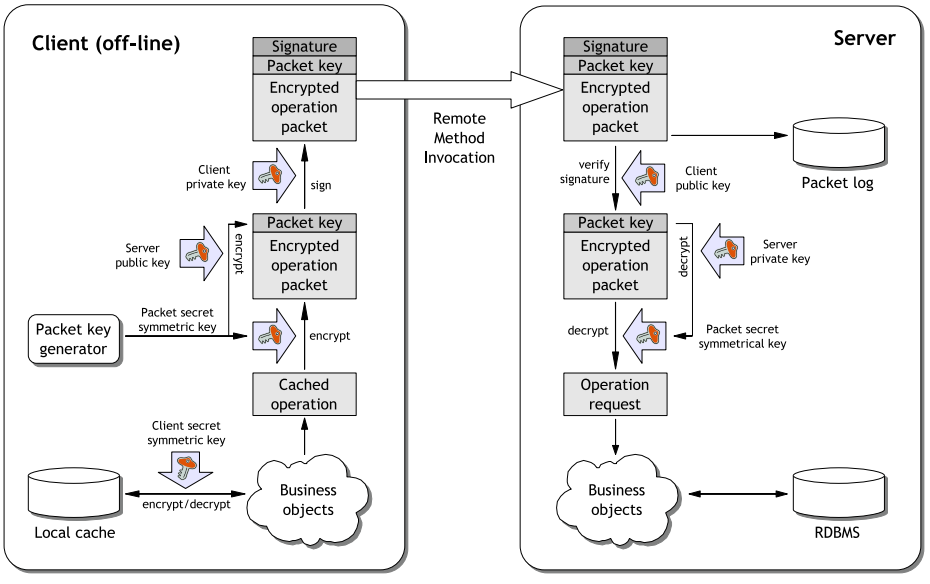


Fig. 4. Offline Business Objects security model

the server's public key and appended at the beginning of the packet. Finally, the packet is signed with the client's private key and the signature along with the client's identifier is appended at the beginning of the packet.

Upon the receipt of a packet, the server verifies its signature. If the signature is invalid, the packet is logged in the "suspicious" packets log and removed from further processing. Otherwise, the server uses its private key to decrypt the packet's symmetric key, which is then used to decrypt the contents of the packet. Finally, the client operation encoded in the packet is applied to the server's database. Communication in the reverse direction is analogous, with private/public keys swapped appropriately.

4.5 Operation Logging

The purpose of operation logging in the OBO framework is twofold: to provide a reliable client operation history and to ensure a data recovery mechanism in case of a database failure.

As we describe in section 4.4, each client operation, such as creating or modifying a business object (see section 4.2), is sent to the server as a separate encrypted packet. Before the server executes the operation, it logs the corresponding packet. Any such packet can be fetched from the log later on and executed once again when needed. The packet log mechanism can therefore be used to recreate a snapshot of the system's business objects from any time in the past.

The primary purpose of packet logging, however, is to implement a reliable client operation history mechanism. To this end, client packets are signed with the client's private key (see section 4.4), so that fraudulent operations can be (probably with diligent manual analysis) detected on the server side.

4.6 Implementation Considerations

It is in the very nature of a distributed offline information system that a significant proportion of a business object's implementation code deals with system-level concerns, such as persistence, communication or security, and not the business-specific behaviour. Moreover, most of the system-level functionality would be very similar across different business objects. To avoid problems caused by manual implementation of this functionality in every single business object[4], Offline Business Object employs a software development technique called active source code generation.

At the heart of source code generation lies a declarative specification of business objects created by the software developer. Based on that specification, the generator can automatically produce different artifacts, such as source code of Java classes or SQL-DDL statements. An important property of active code generation is that the automatically generated code is not intended for the programmer to modify. Instead, all customisations should be implemented with the use of e.g. subclassing or delegation. In this way, should the declarative specification change, none of the customisations will be lost or overwritten.

In case of OBO, the declarative specification of a business object describes such its elements as object-relational mapping information (i.e. which database tables and columns the object maps to), authorisation methods, security levels (e.g. whether the object's data should be encrypted or not) or concurrent modification resolution strategy. Given the following example specification of a business object representing a school unit:

```
<table name="UNIT" concurrentModification="lastWins">
  <references foreignTable="SCHOOL" authorizationPropagates="true"
              primaryKey="false" onDelete="cascade"/>
  <column name="UNIT_ID" primaryKey="true"/>
  <column name="NAME" type="STRING"/>
  <column name="DESCRIPTION" type="STRING"/>
</table>
```

Offline Business Objects will generate the following artifacts:

BaseUnit.java. A base class for implementing the business behaviour of a school unit

BaseUnitPeer.java. A base class that handles the object-relational mapping of the business object

BaseRightChecker.java. A base class for implementing access control

[4] In case of the *Nabor* project, the number of business object classes exceeded 80.

Table 1. Size of the Offline Business Objects source code

Artifact	*Size*
OBO code generator	2 750 LOC (Java)
	938 lines (Velocity templates)
OBO security and utility classes	4 326 LOC (Java)
TOTAL	8 014 lines
Source code generated by OBO	53 059 LOC (Java)
Total number of business object instances	about 500.000

Data synchronisation. Java classes supporting data synchronisation and communication

Local cache. Java classes implementing the offline caching of the business object

SQL-DDL. Table and sequencer definitions for the business object

In Table 1 we show the size of Offline Business Objects expressed as the number of lines of Java source code. Included are also Java code templates in the Velocity [9] format.

4.7 Limitations

Finishing our discussion of the Offline Business Objects framework, we need to emphasise that the offline processing model is not suitable for all kinds of distributed applications. In particular, OBO is not the best choice for systems which perform large numbers of concurrent modifications of the same data. In such cases, with too many concurrent modifications being rejected, the system may become inconvenient or even impossible to use. Choosing the alternative strategy, on the other hand, would increase the risk of losing data coherence.

The assumption of the *Nabor* system, however, was that for a great majority of business objects there would be only one client with the rights to modify them.[5] Consequently, the number of concurrent modifications would be low enough for the offline processing model to be efficient. For the same reason we did not consider implementing more complex conflict resolution mechanisms, such as non-repudiable information sharing [5].

Another problem related to OBO, but also to the ORM frameworks in general, is different operation semantics. To illustrate this, let us consider two concurrent transactions each of which increases the balance of a bank account by 10; the starting balance is 15. In a relational database, both transactions would execute an SQL statement similar to *update account set balance = balance + 10*, which would finally result in the balance being changed to 35. With an ORM framework, however, a similar semantics is very difficult to achieve. Assuming the "last committer wins" strategy, both transactions would have to execute a

[5] For example, only one of the schools a candidate is applying to has the right to enter and further modify the candidates data.

statement similar to *account.setBalance(account.getBalance() + 10)*, which in both transactions would effectively translate to *account.setBalance(25)* and an incorrect final account balance. The only way of dealing with this problem is rejecting concurrent modifications, which may prove inconvenient for the end users.

5 Evaluation

In 2004 the *Nabor* system, which we built based on the Offline Business Objects framework, was deployed in five major cities in Poland on over 200 independent client workstations and handled almost 30.000 high school candidates. Throughout the operation period, the system performed smoothly and reliably. No attempts to manipulate the admission results were reported, no major usability issues were discovered. We therefore feel that this is the best proof that the offline processing model is feasible and can be implemented in practice.

In Table 2 we summarise the size of the *Nabor* system source code. An interesting observation is that the size of code generated by OBO is more than six times bigger that the source code of OBO itself (see Table 1). This might be considered as a sign that our investment in OBO was a profitable one. It must be borne in mind, however, that programming a source code generator is far more complex than writing regular code, which makes the accurate comparison very difficult. The real benefit can come from reusing OBO in similar projects in the future.

Table 2. Size of the *Nabor* system source code

Artifact	*Size*
Hand-written source code	92 051 LOC (Java)
Source code generated by OBO	53 059 LOC (Java)
Source code generated by JAX-B and others	35 957 LOC (Java)
TOTAL	181 107 LOC (Java)

6 Conclusions and Future Work

Creating a secure distributed desktop application supporting offline operation is a non-trivial task. Even more so is designing and implementing a general framework that would facilitate the development of such applications.

In this paper we have presented the Offline Business Objects framework whose aim is to enable data persistence for distributed desktop applications. We identified the requirements for OBO and its main components. We also discussed a number of specific design and implementation problems we faced. Finally, we emphasised the limitations of the offline operation paradigm that must be considered before adopting this approach in real world software.

Our future work on Offline Business Objects will concentrate on looking into the possibilities of extending the existing Java-based persistence frameworks, such as Hibernate, with offline processing capabilities. Service Data Objects implementations can be used as a communication layer for OBO.

Although OBO has been successfully used as a building block for a real-world application, we are aware that detailed performance analyses and benchmarks are required. Another interesting research area is adapting the OBO framework for mobile distributed applications.

References

1. IBM Workplace Client Technology: Delivering the Rich Client Experience. White paper, IBM Corporation, 2004.
2. Christian Bauer and Gavin King. *Hibernate in Action*. Manning Publications, 2004.
3. Luca Cabibbo. Objects meet relations: On the transparent management of persistent objects. In *CAiSE*, pages 429–445, 2004.
4. Paul Castro, Frederique Giraud, Ravi Konuru, Apratim Purakayastha, and Danny Yeh. A programming framework for mobilizing enterprise applications. *WMCSA*, pages 196–205, 2004.
5. Nick Cook, Paul Robinson, and Santosh K. Shrivastava. Component middleware to support non-repudiable service interactions. In *DSN*, pages 605–. IEEE Computer Society, 2004.
6. IBM Corporation and BEA Systems. Service Data Objects Specification v1.0. 2004.
7. Jens Dibbern, Tim Goles, Rudy Hirschheim, and Bandula Jayatilaka. Information systems outsourcing: a survey and analysis of the literature. *SIGMIS Database*, 35(4):6–102, 2004.
8. Apache Software Foundation. Torque Persistence Layer. *http://db.apache.org/torque*, 2004.
9. Apache Software Foundation. Velocity Template Engine. *http://jakarta.apache.org/velocity*, 2004.
10. Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.
11. Mark L. Fussell. Foundations of object-relational mapping. *ChiMu Corporation*, 1997.
12. Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
13. Pawel Gruszczynski, Bernard Lange, Michal Maciejewski, Cezary Mazurek, Krystian Nowak, Stanislaw Osinski, Maciej Stroinski, and Andrzej Swedrzynski. Building a Large-scale Information System for the Education Sector: A Project Experience. In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, volume IV, pages 145–150, Miami, USA, 2005.
14. Jack Herrington. *Code Generation in Action*. Manning Publications, 2003.
15. HypersonicSQL Lightweight Java SQL Database Engine. *http://hsqldb.sourceforge.net*, 2005.
16. Sun Microsystems. Enterprise JavaBeans. *http://java.sun.com/products/ejb*, 2005.
17. Sun Microsystems. Java Data Objects Specification v1.0.1. *http://www.jdocentral.com*, 2005.

18. Sun Microsystems. Java Foundation Classes. *http://java.sun.com/products/jfc*, 2005.
19. Sun Microsystems. JDBC. *http://java.sun.com/products/jdbc*, 2005.
20. Panos A. Patsouris. A formal versioning approach for distributed objectbase. In *ICPADS '97: Proceedings of the 1997 International Conference on Parallel and Distributed Systems*, pages 686–693, Washington, DC, USA, 1997. IEEE Computer Society.
21. Prevayler: Free-software prevalence layer for Java. *http://www.prevayler.org*, 2004.
22. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

Middleware Support for Dynamic Component Updating

Jaiganesh Balasubramanian[1], Balachandran Natarajan[2,⋆], Douglas C. Schmidt[1], Aniruddha Gokhale[1], Jeff Parsons[1], and Gan Deng[1]

[1] Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37203, USA
(jai, schmidt, gokhale, parsons, dengg)@dre.vanderbilt.edu
[2] Veritas Software India Ltd., Pune, India
bala.natarajan@veritas.com

Abstract. Component technologies are increasingly being used to develop and deploy distributed real-time and embedded (DRE) systems. To enhance flexibility and performance, developers of DRE systems need middleware mechanisms that decouple component logic from the binding of a component to an application, i.e., they need support for dynamic updating of component implementations in response to changing modes and operational contexts. This paper presents three contributions to R&D on dynamic component updating. First, it describes an inventory tracking system (ITS) as a representative DRE system case study to motivate the challenges and requirements of updating component implementations dynamically. Second, it describes how our SwapCIAO middleware supports dynamic updating of component implementations via extensions to the server portion of the Lightweight CORBA Component Model. Third, it presents the results of experiments that systematically evaluate the performance of SwapCIAO in the context of our ITS case study. Our results show that SwapCIAO improves the flexibility and performance of DRE systems, without affecting the client programming model or client/server interoperability.

1 Introduction

Component middleware is increasingly being used to develop and deploy next-generation distributed real-time and embedded (DRE) systems, such as shipboard computing environments [1], inventory tracking systems [2], avionics mission computing systems [3], and intelligence, surveillance and reconnaissance systems [4]. These DRE systems must adapt to changing modes, operational contexts, and resource availabilities to sustain the execution of critical missions. However, conventional middleware platforms, such as J2EE, CCM, and .NET, are not yet well-suited for these types of DRE systems since they do not facilitate

⋆ Work performed while author at Vanderbilt University.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 978–996, 2005.

the separation of quality of service (QoS) policies from application functionality [5].

To address limitations of conventional middleware, *QoS-enabled component middleware*, such as CIAO [6], Qedo [7], and PRiSm [8], explicitly separates QoS aspects from application functionality, thereby yielding systems that are less brittle and costly to develop, maintain, and extend [6]. Our earlier work on QoS-enabled component middleware has focused on (1) identifying patterns for composing component-based middleware [9, 10], (2) applying reflective middleware [11] techniques to enable mechanisms within the component-based middleware to support different QoS aspects [12], (3) configuring real-time aspects [6] within component middleware to support DRE systems, and (4) developing domain-specific modeling languages that provide design-time capabilities to deploy and configure component middleware applications [13]. This paper extends our prior work by *evaluating middleware techniques for updating component implementations dynamically and transparently (i.e., without incurring system downtime) to optimize system behavior under diverse operating contexts and mode changes.*

Our dynamic component updating techniques have been integrated into *SwapCIAO*, which is a QoS-enabled component middleware framework that enables application developers to create multiple implementations of a component and update (*i.e.* "swap") them dynamically. SwapCIAO extends CIAO, which is an open-source[1] implementation of the OMG Lightweight CCM [14], Deployment and Configuration (D&C) [15], and Real-time CORBA [16] specifications. Sidebar 1 outlines the features of Lightweight CCM relevant to this paper.

The key capabilities that SwapCIAO adds to CIAO include (1) mechanisms for updating component implementations dynamically without incurring system downtime and (2) mechanisms that transparently redirect clients of an existing component to the new updated component implementation. As discussed in this paper, key technical challenges associated with providing these capabilities involve updating component implementations without incurring significant overhead or losing invocations that are waiting for or being processed by the component.

The remainder of this paper is organized as follows: Section 2 describes the structure and functionality of an inventory tracking system, which is a DRE system case study that motivates the need for dynamic component implementation updating; Section 2.2 describes the key design challenges in provisioning the dynamic component implementation updating capability in QoS-enabled component middleware systems; Section 3 describes the design of SwapCIAO, which provides dynamic component implementation updating capability for Lightweight CCM; Section 4 analyzes the results from experiments that systematically evaluate the performance of SwapCIAO for various types of DRE applications in our ITS case study; Section 5 compares SwapCIAO with related work; and Section 6 presents concluding remarks.

[1] SwapCIAO and CIAO are available from `www.dre.vanderbilt.edu/CIAO`.

Sidebar 1: Overview of Lightweight CCM

The OMG Lightweight CCM [14] specification standardizes the development, configuration, and deployment of component-based applications. Applications developed with Lightweight CCM are not tied to any particular language, platform, or network. *Components* in Lightweight CCM are implemented by *executors* and collaborate with other components via *ports*, including (1) *facets*, which define an interface that accepts point-to-point method invocations from other components, (2) *receptacles*, which indicate a dependency on point-to-point method interface provided by another component, and (3) *event sources/sinks*, which indicate a willingness to exchange typed messages with one or more components.

Assemblies of components in Lightweight CCM are deployed and configured via the OMG D&C [15] specification, which manages the deployment of an application on nodes in a target environment. The information about the component assemblies and the target environment in which the components will be deployed are captured in the form of XML descriptors defined by the D&C specification. A standard deployment framework parses XML assembly descriptors and deployment plans, extracts connection information from them, and establishes the connections between component ports. In the context of this paper, a *connection* refers to the high-level binding between an object reference and its target component, rather than a lower-level transport connection.

2 Case Study to Motivate Dynamic Component Updating Requirements

To examine SwapCIAO's capabilities in the context of a representative DRE system, we developed an *inventory tracking system* (ITS), which is a warehouse management infrastructure that monitors and controls the flow of goods and assets within a storage facility. Users of an ITS include couriers (such as UPS, DHL, and Fedex), airport baggage handling systems, and retailers (such as Walmart and Target). This section describes (1) the structure/functionality of our ITS case study and (2) the key requirements that SwapCIAO dynamic component updating framework had to address. Naturally, SwapCIAO's capabilities can be applied to many DRE systems – we focus on the ITS case study in this paper to make our design discussions and performance experiments concrete.

2.1 Overview of ITS

An ITS provides mechanisms for managing the storage and movement of goods in a timely and reliable manner. For example, an ITS should enable human operators to configure warehouse storage organization criteria, maintain the inventory throughout a highly distributed system (which may span organizational and national boundaries), and track warehouse assets using decentralized operator consoles. In conjunction with colleagues at Siemens [17], we have developed the ITS shown in Figure 1 using SwapCIAO. This figure shows how our ITS consists of the following three subsystems:

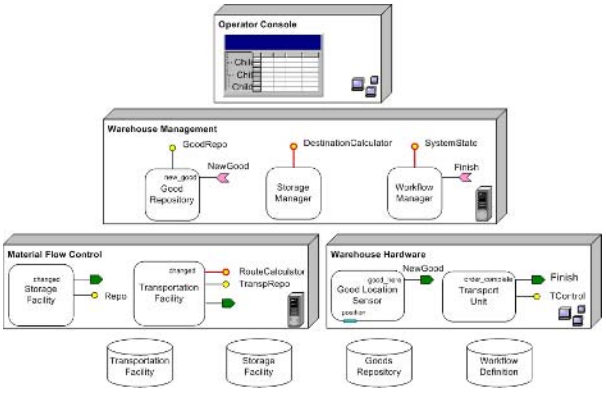


Fig. 1. Key Components in ITS

- **Warehouse management**, whose high-level functionality and decision-making components calculate the destination locations of goods and delegate the remaining details to other ITS subsystems. In particular, the warehouse management subsystem does not provide capabilities like route calculation for transportation or reservation of intermediate storage units.
- **Material flow control**, which handles all the details (such as route calculation, transportation facility reservation, and intermediate storage reservation) needed to transport goods to their destinations. The primary task of this subsystem is to execute the high-level decisions calculated by the warehouse management subsystem.
- **Warehouse hardware**, which deals with physical devices (such as sensors) and transportation units (such as conveyor belts, forklifts, and cranes).

2.2 Requirements for Dynamic Component Updates

Throughout the lifetime of an ITS, new physical devices may be added to support the activities in the warehouse. Likewise, new models of existing physical devices may be added to the warehouse, as shown in Figure 2. This figure shows the addition of a new conveyor belt that handles heavier goods in a warehouse. The ITS contains many software controllers, which collectively manage the entire system. For example, a software controller component manages each physical device controlled by the warehouse hardware subsystem. When a new device is introduced, a new component implementation must be loaded dynamically into the ITS. Likewise, when a new version of a physical device arrives, the component that controls this device should be updated so the software can manage the new version. ITS vendors are responsible for providing these new implementations.

As shown in Figure 2, a workflow manager component is connected to a conveyor belt component using a facet/receptacle pair and an event source/sink pair. To support this scenario, the ITS needs middleware that can satisfy the following three requirements:

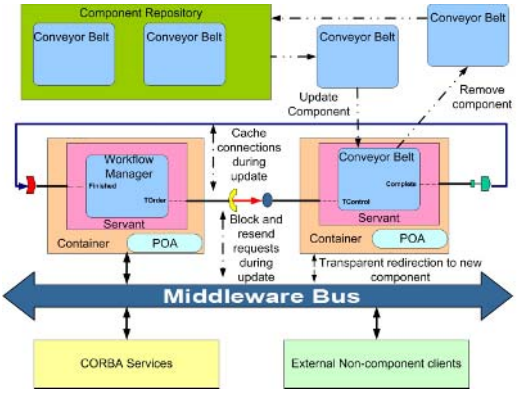


Fig. 2. Component Updating Scenario in ITS

1. Consistent and uninterrupted updates to clients. As part of the dynamic update process, a component's implementation is deactivated, removed, and updated. To ensure that the ITS remains consistent and uninterrupted during this process, the middleware must ensure that (1) ongoing invocations between a component and a client are completed and (2) new invocations from clients to a component are blocked until its implementation has been updated. Figure 2 shows that when a conveyor belt's component implementation is updated, pending requests from the workflow manager to the conveyor belt component to move a new good to a storage system should be available for processing after the implementation is updated. Section 3.1 explains how SwapCIAO supports this requirement.

2. Efficient client-transparent dynamic component updates. After a component is updated, the blocked invocations from clients should be redirected to the new component implementation. This reconfiguration should be transparent to clients, *i.e.*, they should not need to know when the change occurred, nor should they incur any programming effort or runtime overhead to communicate with the new component implementation. Figure 2 shows how a client accessing an ITS component should be redirected to the updated component transparently when dynamic reconfiguration occurs. Section 3.2 explains how SwapCIAO supports this requirement.

3. Efficient (re)connections of components. Components being updated may have connections to other components through the ports they expose. The connected components and the component being updated share a requires/provides relationship by exchanging invocations through the ports. In Lightweight CCM, these connections are established at deployment time using data provided to the deployment framework in the form of XML descriptors. During dynamic reconfiguration, therefore, it is necessary to cache these connections so they can be restored immediately after reconfiguration. Figure 2 shows how, during the update of a conveyor belt component, its connections to the workflow manager

component must be restored immediately after the new updated conveyor belt component implementation is started. Section 3.3 explains how SwapCIAO supports this requirement.

3 The SwapCIAO Dynamic Component Updating Framework

This section describes the design of SwapCIAO, which is a C++ framework that extends CIAO to support dynamic component updates. Figure 3 shows the following key elements in the SwapCIAO framework:

- SwapCIAO's *component implementation language definition* (CIDL) compiler supports the *updatable* option, which triggers generation of "glue code" that (1) defines a factory interface to create new component implementations, (2) provides hooks for server application developers to choose which component implementation to deploy, (3) creates, installs, and activates components within a POA chosen by an application, and (4) manages the port connections of an updatable component.
- The *updatable container* provides an execution environment in which component implementations can be instantiated, removed, updated, and (re)executed. An updatable container enhances the standard Lightweight CCM *session container* [18] to support additional mechanisms through which component creation and activation can be controlled by server application developers.
- The *updatable component factory* creates components and implements a wrapper facade [10] that provides a portable interface used to implement the Component Configurator pattern [10], which SwapCIAO uses to open and load dynamic link libraries (DLLs) on heterogeneous run-time platforms.

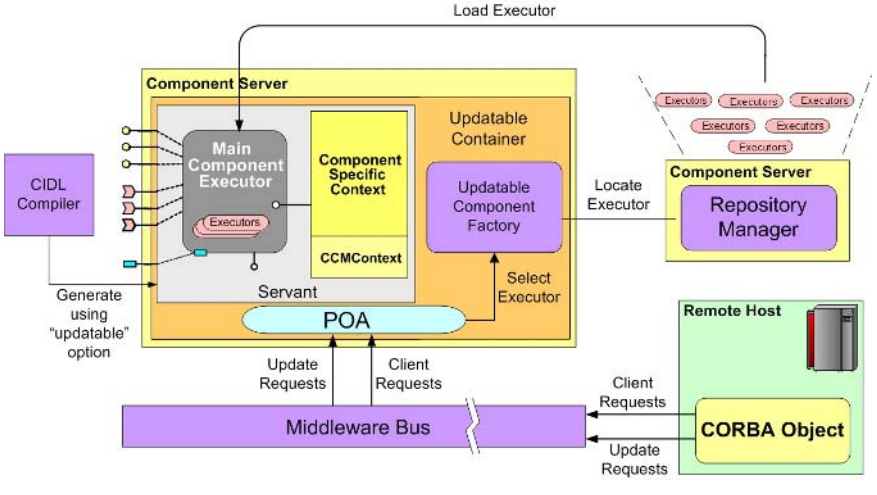


Fig. 3. Dynamic Interactions in the SwapCIAO framework

– The *repository manager* stores component implementations. SwapCIAO's updatable component factory uses the repository manager to search DLLs and locate component implementations that require updating.

The remainder of this section describes how the SwapCIAO components in Figure 3 address the requirements presented in Section 2.2.

3.1 Providing Consistent and Uninterrupted Updates to Clients

Problem. Dynamic updates of component implementations can occur while interactions are ongoing between components and their clients. For example, during the component update process, clients can initiate new invocations on a component – there may also be ongoing interactions between components. If these scenarios are not handled properly by the middleware some computations can be lost, yielding state inconsistencies.

Solution → Reference counting operation invocations. In SwapCIAO, all operation invocations on a component are dispatched by the standard Lightweight CCM portable object adapter (POA), which maintains a *dispatching table* that tracks how many requests are being processed by each component in a thread. SwapCIAO uses standard POA reference counting and deactivation mechanisms [19] to keep track of the number of clients making invocations on a component. After a server thread finishes processing the invocation, it decrements the reference count in the dispatching table.

When a component is about to be removed during a dynamic update, the POA does not deactivate the component until its reference count becomes zero, *i.e.*, until the last invocation on the component is processed. To prevent new invocations from arriving at the component while it is being updated, SwapCIAO's updatable container blocks new invocations for this component in the server ORB using standard CORBA portable interceptors [20].

Applying the solution to ITS. In the ITS case study, when the conveyor belt component implementation is being updated, the warehouse hardware system could be issuing requests to the conveyor belt component to move goods. The updatable container (which runs in the same host as the conveyor belt component) instructs the SwapCIAO middleware to block those requests. After the requests are blocked by SwapCIAO, the updatable container's POA deactivates the conveyor belt component only when all requests it is processing are completed, *i.e*, when its reference count drops to zero.

3.2 Ensuring Efficient Client-Transparent Dynamic Component Updates

Problem. As shown in the Figure 3, many clients can access a component whose implementation is undergoing updates during the dynamic reconfiguration process. In Lightweight CCM, a client holds an object reference to a component. After a component implementation is updated, old object references are no longer valid. The dynamic reconfiguration of components needs to be transparent to

clients, however, so that clients using old references to access updated component do not receive "invalid reference" exceptions. Such exceptions would complicate client application programming and increase latency by incurring additional round-trip messages, which could unduly perturb the QoS of component-based DRE systems.

Solution → Use servant activators to redirect clients to update components transparently. Figure 4 shows how SwapCIAO redirects clients transparently to an updated component implementation. During the component updating process,

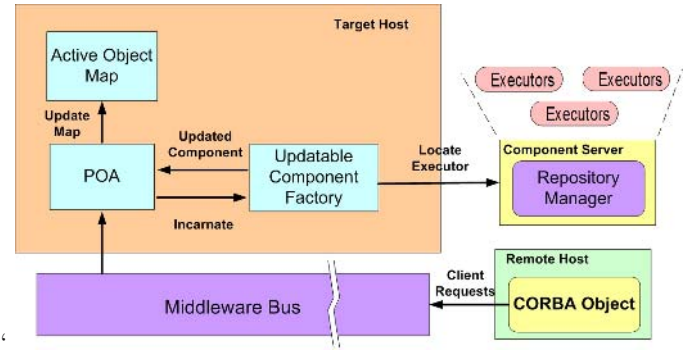


Fig. 4. Transparent Component Object Reference Update in SwapCIAO

the old component implementation is removed. When a client makes a request on the old object reference after a component has been removed, the POA associated with the updatable container intercepts the request via a *servant activator*. This activator is a special type of intercepter that can dynamically create a component implementation if it is not yet available to handle the request. Since the component has been removed, the POA's active object map will have no corresponding entry, so the servant activator will create a new component implementation dynamically.

SwapCIAO stores information in the POA's active object map to handle client requests efficiently. It also uses CORBA-compliant mechanisms to activate servants via unique user id's that circumvent informing clients of the updated implementation. This design prevents extra network round-trips to inform clients about an updated component's implementation.

Applying the solution to ITS. In the ITS case study, when the conveyor belt component implementation is being updated, the warehouse hardware system could be issuing requests to the conveyor belt component to move goods. After the current conveyor belt component is removed, the servant activator in the updatable container's POA intercepts requests from the warehouse hardware subsystem clients to the conveyor belt component. The servant activator then activates a new conveyor belt component implementation and transparently redirects the requests from the warehouse hardware subsystem to this updated implementation. SwapCIAO uses these standard CORBA mechanisms to enable different

component implementations to handle the requests from warehouse hardware subsystem clients transparently, without incurring extra round-trip overhead or programming effort by the clients.

3.3 Enabling (Re)connections of Components

Problem. As discussed in Sidebar 1, Lightweight CCM applications use the standard OMG Deployment and Configuration (D&C) [15] framework to parse XML assembly descriptors and deployment plans, extract connection information from them, and establish connections between component ports. This connection process typically occurs during DRE system initialization. When component implementations are updated, it is therefore necessary to record each component's connections to its peer components since their XML descriptors may not be available to establish the connections again. Even if the XML is available, reestablishing connections can incur extra round-trip message exchanges across the network.

Solution → Caching component connections Figure 5 shows how SwapCIAO handles component connections during the component update process. During the component updating process, SwapCIAO caches component connections to any of its peer component ports. SwapCIAO automatically handles the case where the updated component is a facet and the connected component is a receptacle. Since the receptacle could make requests on the facet while the component implementation is being updated, SwapCIAO uses the mechanisms described in Section 3.1 to deactivate the facets properly, so that no invocations are dispatched to the component. When the new component is activated, the facets are reactivated using the SwapCIAO's POA servant activator mechanism discussed in Section 3.2. For event source and event sinks, if the component being updated is the publisher, SwapCIAO caches the connections of all the connected consumers. When the updated component implementation is reactivated, its connections are restored from the cache. As a result, communication can be started immediately, without requiring extra network overhead.

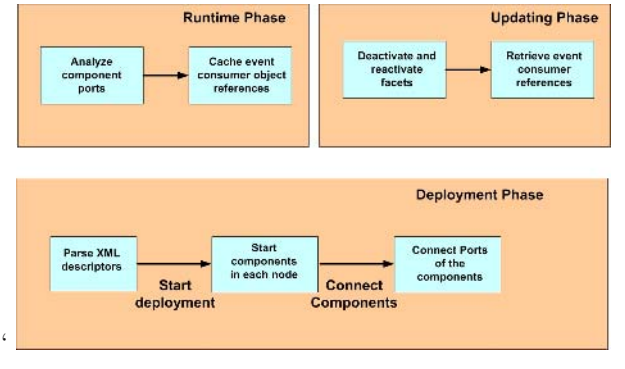


Fig. 5. Enabling (Re)connections of Components in SwapCIAO

Applying the solution to ITS. In the ITS, a conveyor belt component in the warehouse hardware subsystem is connected to many sensors that assist the conveyor belt in tracking goods until they reach a storage system. When a conveyor belt component is updated, its connections to sensor components are cached before deactivation. When the updated conveyor belt component implementation is reactivated, the cached connections are restored and communication with the sensors can start immediately and all requests blocked during the update process will then be handled.

4 Empirical Results

This section presents the design and results of experiments that empirically evaluate how well SwapCIAO's dynamic component updating framework described in Section 3 addresses the requirements discussed in Section 2.2. We focus on the performance and predictability of SwapCIAO's component updating mechanisms provided by version 0.4.6 of SwapCIAO. All experiments used a single 850 MHz CPU Intel Pentium III with 512 MB RAM, running the RedHat Linux 7.1 distribution, which supports kernel-level multi-tasking, multi-threading, and symmetric multiprocessing. The benchmarks ran in the POSIX real-time thread scheduling class [21] to increase the consistency of our results by ensuring the threads created during the experiment were not preempted arbitrarily during their execution.

Figure 6 shows key component interactions in the ITS case study shown in Figure 1 that motivated the design of these benchmarks using SwapCIAO.

As shown in this figure, the workflow manager component of the material flow control subsystem is connected to the conveyor belt and forklift transportation units of the warehouse hardware subsystem. We focus on the scenario where the workflow manager contacts the conveyor belt component using the `move_item()` operation to instruct the conveyor belt component to move an item from a *source* (such as a loading dock) to a *destination* (such as a warehouse storage location). The `move_item()` operation takes source and destination locations as its input arguments. When the item is moved to its destination successfully, the conveyor

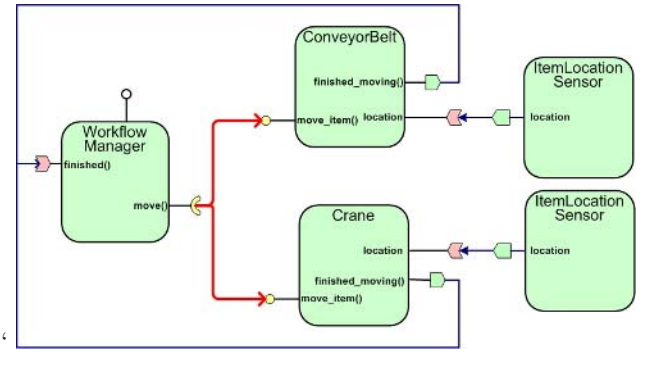


Fig. 6. Component Interaction in the ITS

belt component informs the workflow manager using the `finished_moving()` event operation. The conveyor belt component is also connected to various sensor components, which determine if items fall off the conveyor belt. It is essential that the conveyor belt component not lose connections to these sensor components when component implementation updates occur.

During the component updating process, workflow manager clients experience some delay. Our benchmarks reported below measure the delay and jitter (which is the variation of the delay) that workflow manager clients experience when invoking operations on conveyor belt component during the component update process. They also measure how much of the total delay is incurred by the various activities that SwapCIAO performs when updating a component implementation. In our experiments, all components were deployed on the same machine to alleviate the impact of network overhead in our experimental results.

The core CORBA benchmarking software is based on the single-threaded version of the "`TestSwapCIAO`" performance test distributed with CIAO.[2] This benchmark creates a session for a single client to communicate with a single component by invoking a configurable number of `move_item()` operations. The conveyor belt component is connected to the sensor components using event source/sink ports.

Section 3.3 describes how caching and reestablishing connections to peer components are important steps in the component updating process. We therefore measured the scalability of SwapCIAO when an updated component has upto 16 peer components using event source/sink ports. The tests can be configured to use either the standard Lightweight CCM session containers or SwapCIAO's updatable containers (described in Section 3). `TestSwapCIAO` uses the default configuration of TAO, which uses a reactive concurrency model to collect replies.

4.1 Measuring SwapCIAO's Updatable Container Overhead for Normal Operations

Rationale. Section 3 described how SwapCIAO extends Lightweight CCM and CIAO to support dynamic component updates. DRE systems do not always require dynamic component updating, however. It is therefore useful to compare the overhead of SwapCIAO's updatable container versus the standard Lightweight CCM session container under *normal operations* (*i.e.*, without any updates) to evaluate the tradeoffs associated with this feature.

Methodology. This experiment was run with two variants: one using the SwapCIAO updatable container and the other using the standard CIAO session container. In both experiemnts, we used high-resolution timer probes to measure the latency of `move_item()` operation from the workflow manager component to the conveyor belt component. Since SwapCIAO caches and restores a component's connections to its peer components, we varied the number of sensor components connected to the conveyor belt and then collected latency data with 2, 4, 8, and

[2] The source code for `TestSwapCIAO` is available at `www.dre.vanderbilt.edu/~jai/TAO/CIAO/performance-tests/SwapCIAO`.

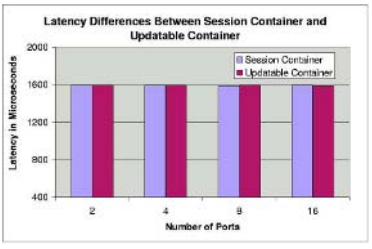


Fig. 7. Overhead of SwapCIAO's Updatable Container

16 ports to determine whether SwapCIAO incurred any overhead with additional ports during normal operating mode. The `TestSwapCIAO` client made 200,000 invocations of `move_item()` operation to collect the data shown in Figure 7.

Analysis of results. Figure 7 shows the comparitive latencies experienced by the workflow manager client when making invocations on conveyor belt component created with the session container versus the updatable container. These results indicate that no appreciable overhead is incurred by SwapCIAO's updatable container for normal operations that do not involve dynamic swapping.

The remainder of this section uses the results in Figure 7 as the *baseline processing delay* to evaluate the delay experienced by workflow manager clients when dynamic updating of a conveyor belt component occurs.

4.2 Measuring SwapCIAO's Updatable Container Overhead for Updating Operations

Rationale. Evaluating the efficiency, scalability, and predictability of SwapCIAO's component updating mechanisms described in Section 3.2 and Section 3.3 is essential to understand the tradeoffs associated with updatable containers. SwapCIAO's *component update time* includes (1) the *removal time*, which is the time SwapCIAO needs to remove the existing component from service, (2) the *creation time*, which is the time SwapCIAO needs to create and install a new component, and (3) the *reconnect time*, which is the time SwapCIAO needs to restore a component's port connections to its peer components.

Methodology. Since the number of port connections a component has affects how quickly it can be removed and installed, we evaluated SwapCIAO's component update time by varying the number of ports and measuring the component's:

- *Removal time*, which was measured by adding timer probes to SwapCIAO's `CCM_Object::remove()` operation, which deactivates the component servant, disassociates the executor from the servant, and calls `ccm_passivate()` on the component.
- *Creation time*, which was measured by adding timer probes to SwapCIAO's `PortableServer::ServantActivator::incarnate()` operation, which creates and installs a new component, as described in Section 3.2.

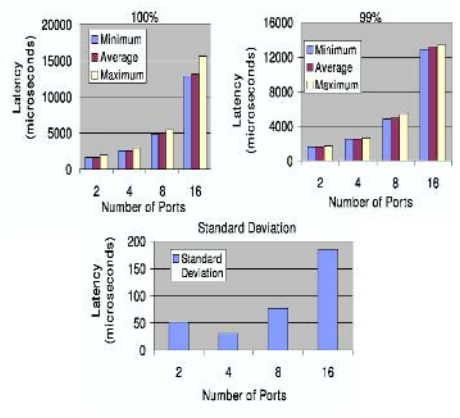


Fig. 8. Latency Measurements for Component Creation

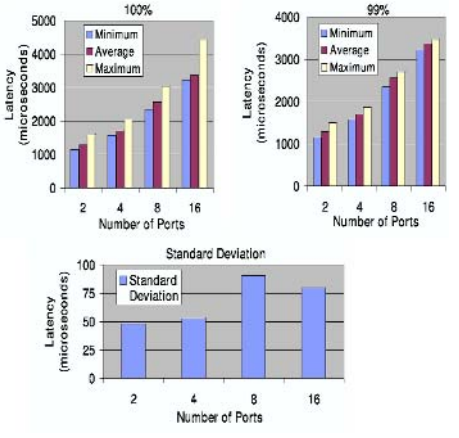


Fig. 9. Latency Measurements for Reconnecting Component Connections

– *Reconnect time*, which was measured by adding timer probes to `CCM_Object::ccm_activate()`, which establishes connections to ports.

We measured the times outlined above whenever a component update occurs during a `move_item()` call for 200,000 iterations and then calculated the results presented below.

Analysis of creation time. Figure 8 shows the minimum, average, and maximum latencies, as well as the 99% latency percentile, incurred by SwapCIAO's servant activator to create a new component, as the number of ports vary from 2, 4, 8, and 16. This figure shows that latency grows linearly as the number of ports initialized by `PortableServer::ServantActivator::incarnate()` increases. It also shows that SwapCIAO's servant activator spends a uniform amount of time creating a component and does not incur significant overhead when this process is repeated 200,000 times. SwapCIAO's creation mechanisms described in Section 3.2 are therefore efficient, predictable, and scalable in *ensuring efficient client-transparent dynamic component updates.*

Analysis of reconnect time. Figure 9 shows the minimum, average, and maximum latencies, as well as 99% latency percentile, incurred by SwapCIAO's reconnect mechanisms to restore a new component's connections, as the number of ports vary from 2, 4, 8, and 16. As shown in the figure, the reconnect time increases linearly with the number of ports per component. These results indicate that SwapCIAO's reconnect mechanisms described in Section 3.3 provide *efficient (re)connection of components* and do not incur any additional roundtrip delays by propagating exceptions or sending GIOP LOCATE_FORWARD messages to restore connections to components.

Analysis of removal time. Figure 10 shows the time used by SwapCIAO's removal mechanisms to cache a component's connections and remove the component from service, as a function of the number of its connected ports. This

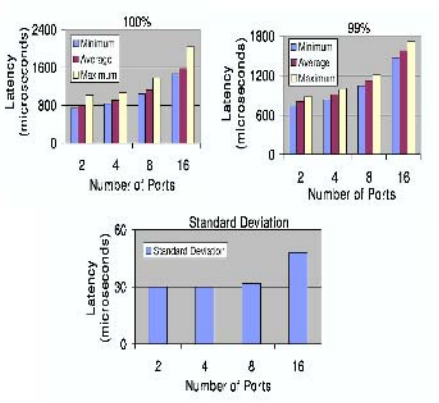


Fig. 10. Latency Measurements for Component Removal

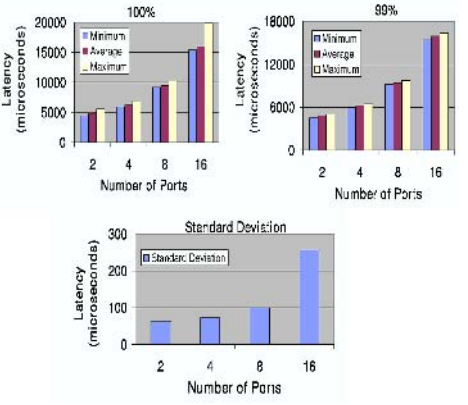


Fig. 11. Client Experienced Incarnation Delays during Transparent Component Updates

removal time increases linearly with the number of ports, which indicates that SwapCIAO performs a constant amount of work to manage the connection information for each port. SwapCIAO's removal mechanisms described in Section 3.1 are therefore able to *provide consistent and uninterrupted updates to clients*.

4.3 Measuring the Update Latency Experienced by Clients

Rationale. Section 3.2 describes how SwapCIAO's component creation mechanisms are transparent to clients, efficient, and predictable in performing client-transparent dynamic component updates. Section 4.2 showed that SwapCIAO's standard POA mechanisms and the servant activator create new component implementations efficiently and predictably. We now determine whether SwapCIAO incurs any overhead – other than the work performed by the SwapCIAO's component creation mechanisms – that significantly affects client latency.

Methodology. The *incarnation delay* is defined as the period of time experienced by a client when (1) its operation request arrives at a server ORB after SwapCIAO has removed the component and (2) it receives the reply after SwapCIAO creates the component, restores the component's connections to peer components, and allows the updated component to process the client's request. The incarnation delay therefore includes the *creation time*, *reconnect time*, and *processing delay* (which is the time a new component needs to process the operation request and send a reply to the client). To measure incarnation delay, we (1) removed a component and (2) started a high-resolution timer when the client invokes a request on the component. We repeated the above experiment for 200,000 invocations and measured the latency experienced by the client for each invocation. We also varied the number of ports between 2, 4, 8, and

16 as described in Section 4.2 to measure the extent to which SwapCIAO's component creation process is affected by the number of ports connected to a component.

Analysis of results. Figure 11 shows the delay experienced by a client as SwapCIAO creates a component with a varying number of connections to process client requests. By adding the delays in Figure 8, Figure 9, and Figure 7 and comparing them with the delays in Figure 11, we show how the incarnation delay is roughly equal to the sum of the creation time, reconnect time, and processing delay, regardless of whether the client invokes an operation on a updating component with ports ranging from 2, 4, 8, to 16.

These results validate our claim in Section 3.2 that SwapCIAO provides component updates that are transparent to clients. In particular, if SwapCIAO's servant activator did not transparently create the component and process the request, the client's delay incurred obtaining a new object reference would be larger than the sum of the creation time, reconnect time, and the processing delay. We therefore conclude that SwapCIAO provides efficient and predictable client transparent updates.

5 Related Work

This section compares our R&D efforts on SwapCIAO with related work ranging from offline updates to hot standby and application-specific techniques.

Offline techniques. Component updating has often been done via offline techniques, where applications are stopped to perform the update and restarted with the new implementation. For example, in [22] when a node is reconfigured, other nodes that require service from the target node are blocked completely, unnecessarily delaying services that are not being reconfigured. To minimize system interruption, [23] uses a centralized configuration manager, that oversees the interactions among components. The centralized configuration manager becomes the single point of failure and also a bottleneck for communication among components. Such techniques can be overly rigid and inefficient for certain types of DRE applications, such as online trading services and inventory tracking systems, where downtime is costly. To address these limitations, SwapCIAO updates component implementations dynamically by (1) queuing requests from other components during the component update and (2) transparently redirecting those requests to the updated implementation, thereby enabling uninterrupted online component updates.

Hot standby techniques. Another component updating technique uses online backup implementations, called "hot standbys." In this approach, when a component needs updating, requests to it will be transferred to the backup, during which the main implementation is updated [24]. Although this solution is common, it can be complex and resource-intensive. In particular, when adding

backup implementations to resource-constrained DRE systems, such as satellite and avionics mission computing systems, it can be unduly expensive and complicated to keep the backup implementation updated and to reroute requests to this standby when the main implementation is being updated. To address these limitations, SwapCIAO does not run a backup implementation and instead updates implementations dynamically. Although requests to the target component are queued during the update, no round-trip overhead is incurred to redirect client requests from one node to another. Moreover, queued requests in SwapCIAO are redirected transparently to the updated implementation once it is activated.

Application-specific techniques. Another technique employs application-specific modifications to handle component updates. For example, [25] introduces a component configurator that performs reconfiguration at the application level. As a result, application developers must implement a configurator for each component. Moreover, handling connections among components is hard since there is no central entity managing information about the overall DRE system structure. To address these issues, SwapCIAO leverages patterns (such as Reference Counting and Dispatching [19], Wrapper Facade [10] and Component Configurator [10]) and frameworks (such as Portable Interceptors [20] and ACE Service Configurator [26]) to implement the dynamic component updating capability in the middleware. Application developers are therefore able to focus on their component implementations, rather than wrestling with complex mechanisms needed to add dynamic component updating capabilities into their applications.

6 Concluding Remarks

This paper describes the design and implementation of SwapCIAO, which is a QoS-enabled component middleware framework based on Lightweight CCM that supports dynamic component updating. SwapCIAO is designed to handle dynamic operating conditions by updating component implementations that are optimized for particular run-time characteristics. The lessons learned while developing SwapCIAO and applying it to the ITS case study include:

- Standard Lightweight CCM interfaces can be extended slightly to develop a scalable and flexible middleware infrastructure that supports dynamic component updating. In particular, SwapCIAO's extensions require minimal changes to the standard Lightweight CCM server programming model. Moreover, its client programming model and client/server interoperability were unaffected by the server extensions. Developers of client applications in our ITS case study were therefore shielded entirely from SwapCIAO's component updating extensions.
- By exporting component implementations as DLLs, SwapCIAO simplifies the task of updating components by enabling their implementations to be linked

into the address space of a server late in its lifecycle, *i.e.*, during the deployment and reconfiguration phases. These capabilities enabled developers in the ITS case study to create multiple component implementations rapidly and update dynamically in response to changing modes and operational contexts.
– SwapCIAO adds insignificant overhead to each dynamic component updating request. It can therefore be used even for normal operations in ITS applications that do not require dynamic component updating. Moreover, due to the predictability and transparency provided by SwapCIAO, it can be used efficiently when operating conditions trigger mode changes.

Our future work will focus on developing selection algorithms [27] that can automatically choose the most suitable component implementation to update in a particular operating condition. We will implement these selection algorithms and validate them in the context of DRE systems, such as our ITS case study. To enhance the autonomic properties of DRE systems, we are developing a monitoring framework within SwapCIAO that (1) observes the performance of different components, (2) identifies when performance is not within the desired QoS bounds, and (3) automatically updates component implementations using our selection algorithms.

References

1. Schmidt, D.C., Schantz, R., Masters, M., Cross, J., Sharp, D., DiPalma, L.: Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems. CrossTalk (2001)
2. Nechypurenko, A., Schmidt, D.C., Lu, T., Deng, G., Gokhale, A., Turkay, E.: Concern-based Composition and Reuse of Distributed Systems. In: Proceedings of the 8th International Conference on Software Reuse, Madrid, Spain, ACM/IEEE (2004)
3. Sharp, D.C., Roll, W.C.: Model-Based Integration of Reusable Component-Based Avionics System. In: Proc. of the Workshop on Model-Driven Embedded Systems in RTAS 2003. (2003)
4. Sharma, P., Loyall, J., Heineman, G., Schantz, R., Shapiro, R., Duzan, G.: Component-Based Dynamic QoS Adaptations in Distributed Real-Time and Embedded Systems. In: Proc. of the Intl. Symp. on Dist. Objects and Applications (DOA'04), Agia Napa, Cyprus (2004)
5. Wang, N., Gill, C.: Improving Real-Time System Configuration via a QoS-aware CORBA Component Model. In: Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2003, Honolulu, HW, HICSS (2003)
6. Wang, N., Gill, C., Schmidt, D.C., Subramonian, V.: Configuring Real-time Aspects in Component Middleware. In: Proc. of the International Symposium on Distributed Objects and Applications (DOA'04), Agia Napa, Cyprus (2004)
7. Ritter, T., Born, M., Unterschütz, T., Weis, T.: A QoS Metamodel and its Realization in a CORBA Component Infrastructure. In: Proceedings of the 36^{th} Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2003, Honolulu, HW, HICSS (2003)

8. Roll, W.: Towards Model-Based and CCM-Based Applications for Real-Time Systems. In: Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC), Hakodate, Hokkaido, Japan, IEEE/IFIP (2003)
9. Balasubramanian, K., Schmidt, D.C., Wang, N., Gill, C.D.: Towards Composable Distributed Real-time and Embedded Software. In: Proc. of the 8^{th} Workshop on Object-oriented Real-time Dependable Systems, Guadalajara, Mexico, IEEE (2003)
10. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2. Wiley & Sons, New York (2000)
11. Wang, N., Schmidt, D.C., Kircher, M., Parameswaran, K.: Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications. IEEE Distributed Systems Online **2** (2001)
12. Wang, N., Schmidt, D.C., Parameswaran, K., Kircher, M.: Applying Reflective Middleware Techniques to Optimize a QoS-enabled CORBA Component Model Implementation. In: 24th Computer Software and Applications Conference, Taipei, Taiwan, IEEE (2000)
13. Balasubramanian, K., Balasubramanian, J., Parsons, J., Gokhale, A., Schmidt, D.C.: A Platform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems. In: Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Sym., San Francisco, CA (2005)
14. Object Management Group: Light Weight CORBA Component Model Revised Submission. OMG Document realtime/03-05-05 edn. (2003)
15. Object Management Group: Deployment and Configuration Adopted Submission. OMG Document ptc/03-07-08 edn. (2003)
16. Object Management Group: Real-time CORBA Specification. OMG Document formal/02-08-02 edn. (2002)
17. Nechypurenko, A., Schmidt, D.C., Lu, T., Deng, G., Gokhale, A.: Applying MDA and Component Middleware to Large-scale Distributed Systems: a Case Study. In: Proceedings of the 1st European Workshop on Model Driven Architecture with Emphasis on Industrial Application, Enschede, Netherlands, IST (2004)
18. Schmidt, D.C., Vinoski, S.: The CORBA Component Model Part 3: The CCM Container Architecture and Component Implementation Framework. The C/C++ Users Journal (2004)
19. Pyarali, I., O'Ryan, C., Schmidt, D.C.: A Pattern Language for Efficient, Predictable, Scalable, and Flexible Dispatching Mechanisms for Distributed Object Computing Middleware. In: Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC), Newport Beach, CA, IEEE/IFIP (2000)
20. Wang, N., Schmidt, D.C., Othman, O., Parameswaran, K.: Evaluating Meta-Programming Mechanisms for ORB Middleware. IEEE Communication Magazine, special issue on Evolving Communications Software: Techniques and Technologies **39** (2001) 102–113
21. Khanna, S., *et al.*: Realtime Scheduling in SunOS 5.0. In: Proceedings of the USENIX Winter Conference, USENIX Association (1992) 375–390
22. Kramer, J., Magee, J.: The Evolving Philosophers Problem: Dynamic Change Management. IEEE Transactions on Software Engineering **SE-16** (1990)
23. Bidan, C., Issarny, V., Saridakis, T., Zarras, A.: A Dynamic Reconfiguration Service for CORBA. In: International Conference on Configurable Distributed Systems (ICCDS '98). (1998)

24. Tewksbury, L., Moser, L., Melliar-Smith, P.: Live upgrades of CORBA applications using object replication. In: International Conf. on Software Maintenance, Florence, Italy (2001) 488–497
25. Kon, K., Campbell, R.: Dependence Management in Component-based Distributed Systems. IEEE Concurrency **8** (2000)
26. Schmidt, D.C., Huston, S.D.: C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks. Addison-Wesley, Reading, Massachusetts (2002)
27. Yellin, D.M.: Competitive algorithms for the dynamic selection of component implementations. IBM Systems Journal **42** (2003)

Two Ways of Implementing Software Connections Among Distributed Components

Selma Matougui and Antoine Beugnard

ENST-Bretagne, Technopôle Brest-Iroise,
CS 83 818, 29 238 Brest, France
{selma.matougui, antoine.beugnard}@enst-bretagne.fr

Abstract. Software architecture has emerged in the last decade. Applications are represented as set of interconnected components. The way to realize components has reached a certain maturity in both industrial and academic approaches; it has almost the same consideration or definition in the two domains. The way to implement the interconnections between components, however, is not as well understood as implementing components. The experience of implementing the interconnections between components is dispersed since interconnection models are integrated in component models. Every component model defines its own interconnection model without basing on any reference model. This makes the realizations ad-hoc and non uniformed.

We propose to make more standard the realization of the connections between components and to distinguish two different entities that differ in nature. This difference of nature implies a different way in using them and a different way in implementing them. We propose to distinguish between communication abstractions embodied in components with explicit interfaces, and communication abstractions embodied in connectors with implicit interfaces. This difference enables a better understanding of the interactions and how to implement them. The realization of the load balancing communication abstraction is used to illustrate the two entities.

1 Introduction

Increasingly, software systems or applications are being described and specified in a high level before being designed and implemented. Applications are described as a software architecture which represents, henceforth, an upstream stage in the software development process. Designing an application by adopting a "software architecture" approach, in opposition of adopting a "software design" approach, is a tendency being confirmed in software engineering. While the second approach refers to techniques or development methods leading to rapid realizations of implementation, the first approach defines intermediary stages between the requirements and the implementation, like abstractions, architecture styles, etc. Examples of software design approaches are the Component Based Development (CBD) and industrial approaches like CORBA Component Model (CCM) [1] and Enterprise Java Beans (EJB) [2]. They go from a small description of the application to implementation. Software architecture is rather concerned by academic

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 997–1014, 2005.

approach like Architecture Description Languages (ADL) [3]. These languages make a large study about the structure of the application, study many models in the middle and then pass to implementation.

A common entity between the two approaches is the notion of component. Although there is no universal definition of a component, the two approaches almost agree on its global features. There is a kind of consensus about what a component is: it is a software entity that performs computation and defines explicit offered or required services by hiding their implementation. However, there is not the same consideration about connections between the components; each approach defines them differently. On the one hand, CBD and industrial approaches define simple interactions at design level that are implemented; *i.e.* they exist until deployment time. The inexistence of an entity for complex interactions, however, makes their realization strongly hardwired with the components. On the other hand, some ADL can define complex interactions with a connection model but this latter depends strongly on the defined component model. Rare are the approaches that decouple the two models, hence there is no *reference model* for connections; each ADL defines its own connection model. Moreover, when complex interactions are defined, they are not implemented. They can be specified but their realization is ad-hoc and uses simple connections. They do not stay as an entire entity until deployment, they are lost during the life cycle of the development process.

Considering the connections as an entire entity from architecture level to deployment level has several advantages and we support this idea. In addition to interaction description, a connection model can participate in the description and the realization of non functional properties like security, reliability, etc. It favors separation of concerns and so software reuse, maintenance and evolution. Therefore, it becomes a key determinant in software quality. As we have argued above, current approaches either do not define a connection model and offer implementation of simple connections, or stay at a high level definition and do not offer implementation. This creates a gap between the two levels. In this paper, we propose two different entities to hold the communication abstractions: communication components and pure connectors. They are two different connection models that are independent from any component model. They represent two different ways to go from communication specification to communication implementation in order to fill the gap between these two levels.

In the reminder of this article, section 2 motivates the problem. Then section 3 gives the definitions and details of the two proposed entities and a comparison of them. After that, section 4 illustrates the use of the communication component and the connector in a load balancing example. Next, section 5 makes relationships with other works, and finally section 6 concludes.

2 Motivation

As already recalled in the previous section, two main entities are handled in software architecture, the components and their connections. The components

Fig. 1. Simple Connection

perform computation and functional properties. The connections are concerned by the communication and the coordination between components. They can also participate to ensure non functional properties. Application architectures are typically expressed informally by box-and-line drawings indicating the global organization of the application [4]. The components are represented by boxes and the connections are represented by simple lines as illustrated in figure 1. In such representations, there exist some architecture style conventions that give indications about the nature of the represented components and help to put some constraints. For example, if we are in an application context handling filters and pipes, the filters are the components represented by boxes and the connections are made by the pipes represented by the lines. In this architecture style, if one box is related to several lines an error will be indicated because one filter can not be attached to several pipes; the cardinality is a single sender and a single receiver. Moreover, the pipe allows interaction via unformatted stream data, it represents a unidirectional communication. A bidirectional communication should be represented with two lines, one for transmitting data and the other for receiving data. In another architecture style, client/server for example, the client and the server are the components represented by the boxes. When the client needs a service offered by the server, the communication represented by a line refers, usually, to a Remote Procedure Call (RPC).

When examining closer these connections later in the life cycle we notice that they are very different. At runtime, in the case of the pipe and filter application, a filter A uses the interface *write()* of the pipe in order to transfer data. At the other side, the filter B should use the interface *read()* of the pipe in order to recover data. Hence, the two interacting components (filters) do not possess any knowledge about one another. The components need to interact, but they ask the connection to do it for them. They address their request explicitly to the connection for doing it by using the explicit interfaces of the connection. This is what we name a *communication component*. At architecture level it defines its own interfaces that are preserved until deployment. This kind of connection is assimilated to a component with explicit interfaces that has a functionality of making communicating and interacting other components.

In the example of client/server architecture style, we can represent the connection at deployment as in figure 2. The connection at this stage, *i.e.* the RPC, is divided into three important elements[1]. It is elaborated with the TCP/IP component, the stub and the skeleton. At this stage, they are all considered as independent interacting components. Indeed, every component uses the offered

[1] In fact, it is more complicated. We are reducing to these elements in order to make the explanations simple.

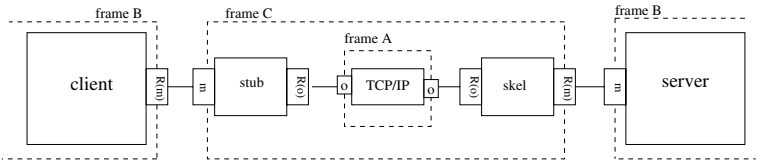


Fig. 2. RPC connection at deployment time

or required interfaces of the other components that hide their implementation. The RPC is then embodied in the component expliciting as interfaces those of the stub and the skeleton (frame C in figure 2). These interfaces are concrete (implemented) at this stage because they represent the first interfaces with which the components actually interact in order to access the low level components, like TCP/IP, and to simulate a local call. When going back in the life cycle and corresponding this entity with the existing elements at architecture level, by a reverse engineering method for example, we find that these interfaces are identical to the interfaces of the interacting components, notably the client and the server. Hence, the RPC does not have its own and concrete interfaces at architecture level. It has just generic interfaces destined to be specified later in the life cycle, by a so called stub compilation. They take their final form thanks to a generation process. Being not a component at architecture level because it depends on other components to have its own interfaces, we name this entity the *connector* with implicit interfaces. The interacting components know one another and do not explicitly require the connection services. This latter adopts interacting components interfaces in order to satisfy the needs of each component by ensuring the appropriate property. In this example, the property to ensure is distribution.

These two kinds of connections exist and are simple. We want to generalize the two processes for realizing more complex and rich communication abstractions. Therefore, we identify two types of entities to hold the connections between components at architecture level that have different nature, different life cycle and different process of development. Therefore, we say that every entity at architecture and design level is not a component, and connectors are not components. Their difference is not only a difference of functionality, as it is usually expressed, but there is also a difference of nature. The components have explicit interfaces and the connectors have implicit ones. In the next section we describe in more details these two models of connection.

3 Two Types of Communication Abstraction

In order to make the implementation of the connections more uniform and to preserve them from specification to implementation, *i.e.* to define a refinement process, we propose two models representing two different communication abstractions. In Figures 3 and 4, we propose to isolate the description of interactions into two different entities which will allow a better organization. We have

changed the notation of the connection, always represented by lines, into ellipses. Hence, we change the boxes-and-lines model into the boxes-and-lines-and-ellipses model, where the ellipses represent complex connections. By complex connections we mean connections that use protocols or that make interacting more than two participants. The line, henceforth, represents a simple local communication. We distinguish two kinds of complex communication abstractions:

1. Communication components with explicit attachment points called ports, represented as squares around the communication component (the ellipse) in Figure 3. The ports implement component interfaces, and therefore the signatures of offered and required methods are fully specified. In Figure 3, the communication component requires the offered operation OO(x) of the component C1 and offers two operations OO(m) and OO(n). These offered operations are used by the required operations RO(m) and RO(n) of components C2 and C3 respectively. The operations x, m and n of C1, C2 and C3 are completely different; they do not know one another and, by using a simple interaction, interact directly with the operations x, m and n of the communication component.

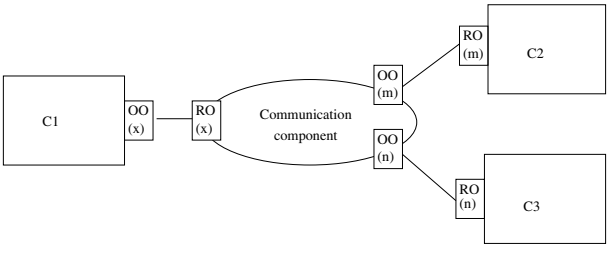


Fig. 3. A communication component

2. Connectors with implicit attachment points called plugs, represented as circles around the connector (the ellipse) in Figure 4. The plugs define implicit or generic interfaces, they do not directly specify any required or offered operation. These interfaces are adaptable to the interfaces of components

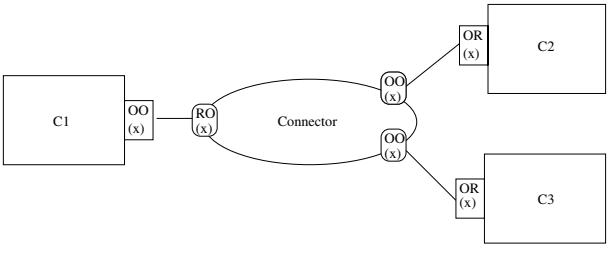


Fig. 4. A connector

connected at assembly time. An isolate connector is put on the shelf without any interface, only generic holders and the property it has to ensure. When connected with other components, like in figure 4, the plugs are specified and adopt the interfaces of interacting components. The offered operation OO(x) of component C1 is required by the RO(x) required operations of components C2 and C3. The connector adopts the interface of the components C1 and offers it to the components C2 and C3 augmented with the property it has to ensure. Hence, this interface is brought to the components by following a special property. In opposition to the communication component, the interacting components know one another and do not see the connector. They explicitly interact one with another through the connector.

This notation by ellipses keeps the difference of functionality between the conventional components and the communication abstractions. Indeed, expressing an interaction entity as a component could make us representing it graphically as a box. As we will see in the next section, the conventional components are different from communication components. We make a graphical difference between the two distinguished communication abstractions by their interfaces, represented by squares and circles, which are the explicit and implicit interfaces. The two notions and their differences are described in more detail in the following.

3.1 Communication Components

A communication component, also called medium in [5], is a reification of an interaction, a communication or a coordination system, a protocol or a service, into a software component. These mediums are various in type and complexity. An event broadcast, a consensus protocol, coordination through a shared memory, a multimedia stream broadcast or a vote system, for instance, can be considered as a communication component. A distributed application is then built by interconnecting "conventional" components with mediums that manage their communication and distribution. A communication component is first of all a component. It satisfies the principle properties of the component paradigm [6]:

- It is an autonomous and deployable software entity.
- It clearly specifies the services it offers and those it requires. This allows the use of a component without knowing how it works.
- It can be combined with other components.

Thus, a medium has all these properties and, moreover, is especially designed to be used for communication. By nature, it is a component with distributed interfaces, which distinguishes it from standard component models. Indeed, models like EJB, .NET or CCM realize distributed components with co-localized interfaces, *i.e.* in spite the implementation can reside in different sites, the interfaces should reside on the same site. Communication components, however, have both interfaces an implementation parts on different sites.

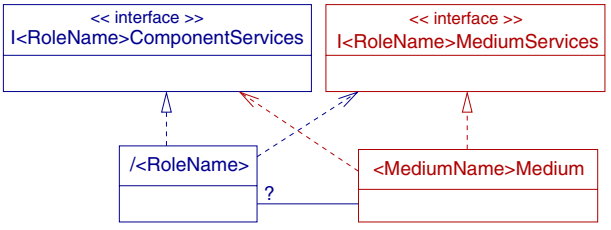


Fig. 5. Generic UML model of a communication component

To specify a communication component, we rely on the UML language. Figure 5 shows a generic form of a specification that highlights the offered (IMediumServices) and required (IComponentServices) interfaces by the communication component. The roles will be played, after assembly, by other components that fit to the interfaces' requirements.

The life cycle of a communication component is close to a conventional component life cycle. It is specified, designed, implemented, assembled with other components, and then deployed. Its particularity is that it relies on an underlying protocol that offers the communication services it needs to work. The whole process is detailed in [7].

3.2 Connectors

A connector is a reification of an interaction, communication or coordination system and exists to serve the communication needs of the interacting components. It describes all the interactions between components relying on their own specification and abstraction. It also offers application-independent interaction mechanisms like security and reliability for instance. Three points distinguish a connector from a component:

- It is an autonomous entity at architecture level but neither deployable nor compilable (it does not exist in conventional programming languages). The connector must be connected in order to be compiled and deployed.
- It does not specify offered or required services but patterns of services. Hence, the interfaces are abstract and generic. They become more concrete later in the life cycle, on assembly with the components. This is done by adopting the interfaces type of the interacting components.
- In order to distinguish services grouped in interfaces of component ports, the word *plug* is introduced to name the description of connector ends. A plug is typed with a name and can have a multiplicity.

To specify a connector we only precise the property it has to ensure, the number of different participants that are in position to interact, their status and their multiplicity. It is an architecture element that indicates the semantic of the communication and that evolves and changes in time to be more specified. It is designed, implemented and put on the shelf without any specified interfaces.

Table 1. Vocabulary by level of abstraction and refinement

Level	Architecture (abstract)	Implementation (concrete)
On the shelf	*Connector*	*Generators*
Assembled	*Connection*	*Binding components*

These latter are specified at assembly with other component thanks to a generation process. After that, they are deployed all together. We introduce a new vocabulary in order to designate each entity in this life cycle. We reduce the use of the term connector to the architecture element and give different names to the following entities in order to differentiate and to not use it to designate every thing related to the interaction entities as it is commonly made. They are different entities so they should be named differently. Table 1 summarizes this vocabulary.

We identify three entities that designate the transformations of the connector in its life cycle:

Generator: An isolated connector is concretized as a family of off-the-shelf generators. These generators are in charge of implementing the connector abstraction over different communication protocols or platforms. The plugs are still abstract because the interacting component are not specified yet. When activating a generator, the plugs are destined to be transformed into proxies towards which the effective communication will take place.

Connection: At architecture level, the connector is linked with other components and form all together the connection. At this stage, the generic interfaces of the connector, the plugs, become concrete by adopting the applicative interfaces (APIs) of the interacting components. They form the connection interfaces. Hence, we connect the components without worrying about the underlying platform, only the semantic of the communication is needed. A connection is different from a connector. The connector exists alone. The connection appears when the components are specified, the connector is chosen and the unit assembled.

Binding component: It is the result of the code generation. The connection interfaces, obtained by the connector and the components assembly, are provided to the generator and fill its unspecified plugs. The generator generates the proxies that are to be deployed and form the binding component[2]. They represent the realization of a communication or a coordination property for the connected components' requirements (APIs) over the underlying system.

Hence, the life cycle of a connector is different from the one of the communication component and independent from the one of a conventional component. Every entity in this life cycle has concrete and abstract elements [9] that become all concrete at the end, at deployment time.

[2] It is a refinement of the *binding object* defined in Open Distributed Processing (ODP) [8].

From an application developer's point of view, the software architecture definition involves in selecting components and connectors on the shelf, assembling, *i.e.* establishing connections, selecting the generators appropriate to the target system, and generating the deployable application. From a connector developer's point of view, defining a connector involves in specifying the abstract properties of the communication (load-balancing for instance), and developing the generators that use the interface definition of connected entities (IDL for instance) over a target system (TCP/IP for instance). As we can see, a family of generators could be developed for a single abstract connector.

An example of an existing generator as defined in this section is the CORBA generator as an instance of the RPC connector. The properties it ensures are distribution, language interoperability, and an RPC semantics. It can rely on several communication protocols like TCP/IP or other low level protocol. The proxies it generates are the stub and the skeleton. The binding component gathers these proxies and the communication protocol.

These two forms of communication abstraction exist as simple ones. We propose to generalize these processes for more complicated properties. The following section gives some differentiation elements between the two connection models.

3.3 Comparison

We have presented, in the two sections above, the two connection models that we propose in order to reduce the distance between the definition of the connection at an architecture level and its implementation; *i.e.* to define complex communications and to keep them as an entire entity until deployment. The two models are the communication component and the connector. They are both architecture elements that support the definition of complex interactions and define a refinement process that avoids to the abstraction to be lost during the life cycle. However, they have different natures and evolve differently in the life cycle.

The communication component represents a well known entity at architecture and we know how to implement it. It is a component dedicated to communication that defines explicit interfaces. It has the same life cycle as conventional components. It can be used to define coarse grained communications that are fairly related to the application, since the interacting components deal explicitly and directly with the medium. An example of a newly built communication component is the reservation medium detailed in [10].

The connector, as defined in this article, is a new concept and is detailed in [9]. It has implicit interfaces that become concrete during the life cycle. It changes during the life cycle and we give for each entity a different name. It can be used for fine grained communication abstractions that are not related to the application functionality. It is destined to automate the integration of the semantic of the communication and non functional properties transparently to the application by the generation process, it is adaptable to the components' needs.

This difference of nature implies differences in description, differences in the nature of the entities to put on the shelf, and differences in using them at assembly with other components. We give solutions to integrate these complex

interactions into industrial approaches to make them used and implemented by two different ways.

The development of a connector can obviously be realized over a medium. The generator is dedicated to generate the appropriate glue that is compatible with the explicit interfaces the medium offers. In that case, the medium is the target of the generator.

We encourage the development of such connectors with complex properties since they have the advantage of automating the integration of complex and repetitive properties. It is obvious that connections can be realized by today available solutions. But these current solutions lack generality and rely on low level solutions, which means difficulties for software maintenance and evolution. Accumulating know-how in generators design makes the software evolution easier than adapting complex and ad-hoc components implementations of connections.

4 Application to Load Balancing

The aim of this section is to illustrate throughout an example the advantages and the limits of the two previous entities: the communication component and the connector.

4.1 Load Balancing Features

Some applications, like e-commerce systems and online stock trading systems, concurrently service many clients transmitting a large, often bursty, number of requests. In order to improve the overall system responsiveness, an effective way to deal with this great demand is to increase the number of servers — replicas offering the same service — and to apply a load balancing technique. Load balancing mechanisms distribute client workload equitably among back-end servers by indicating which server will execute each client's request. These mechanisms may follow either a *non-adaptive* or an *adaptive* policy. Non-adaptive policies do not take the servers' workload into account. The round robin is such a policy. Adaptive policies, however, take the servers' workload into consideration. The policy of the least loaded server is an example. Many load metrics can be used; the CPU's idle cycles or the number of requests, for instance. In the following example, we assume a distributed application where a client needs a defined service offered by a server. As a client needs to send a huge number of requests, these requests would be balanced among n replicas of servers following a special load balancing policy.

Performing load balancing at an OS or a network level does not allow to use application metrics or control the load balancing policy without modifying the application itself. However, achieving load balancing at a high level (eg. application level), with available communication abstractions, implies using a complicated combination of simple and low level connectors like RPC. In order to decouple the application from load balancing, we reify the connection abstraction as an independent entity. For instance, we consider load balancing as

a complex interaction that involves many participants and is achieved with different protocols and policies. In the following, we are going to see the realization of this communication abstraction with the two connection models.

4.2 Load Balancing with a Communication Component

The first method describes the realization of the load balancing service as a communication component or medium, as described in section 3.1. The component has explicit and well defined interfaces that are to be used by the client and the servers. When knowing these interfaces, it is not necessary to know the internal implementation of the component. Hence, we are not concerned by knowing how it is implemented, and so how the load balancing policy is realized.

On the one hand, the Load Balancing Medium (LBMedium) offers the service *getReference()* that gives the reference of the elected component that will serve the request, by applying a specific load balancing policy. On the other hand, it communicates permanently with the servers with the service *load()* (pull or push) to recover their workload, in the case of an adaptive load balancing policy.

The sequence diagram in figure 6 shows the different interactions between the conventional components, notably the client and the server, and the LBMedium.

1. The servers explicitly communicate with the LBMedium in order to subscribe and to communicate their workload by a push or a pull mode,
2. The client explicitly calls the method *getReference()* offered by the communication component in order to recover the reference of the server that will serve the client request,
3. Knowing all the servers participating in the interaction, the LBMedium chooses the server reference to send to the client according to the used policy,
4. The client sends the effective requests to the concerned server, by establishing a session.

This approach preserves the advantages of realizing load balancing at application level. Il allows using application metrics. Moreover, it ensures separation

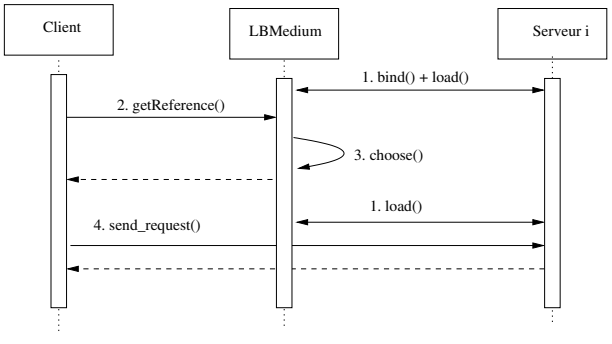


Fig. 6. Load balancing with a communication component

of concerns and avoid the semantic of the communication to be hardwired with the components of the application. This makes easier the programs reuse, maintenance and evolution. If the load balancing policy changes, it does not affect the client and the server. However, there is some drawbacks. First, as the services of the communication component are explicit, the load balancing service is not transparent neither at the client side nor at the server one. The interaction between the medium and the component is completely explicit. If the medium is replaced with another medium expressing different interfaces, this affects the client and the servers. Then, the service should be envisaged in advance and integrated to the application. Finally, this approach is efficient in a connection per session, *i.e.* the client sends all his requests to the first chosen server. However, it is not relevant in a connection per request, where the client can change the server that serves his request when the workload changes.

4.3 Load Balancing with a Connector

The second example consists on realizing load balancing as a connector associated to a generator as described in section 3.2. This approach enables to ensure transparency to the communication. The connector hides all the load balancing mechanisms. It hides also the existence of this service.

The off-the-shelf connector specifies only the nature of the future interacting components (the plugs) and their multiplicity. In this example, the connector has two plugs: a client and a remote server; this latter has a multiplicity of n. The connector is implemented as a generator that ensures the load balancing property in addition to the distributed property, the RPC. Once the connector assembled with the client and the servers, the connector adopts the interfaces the these interacting components; the plug at the client side adopts server interfaces and the plug at server side adopts client interfaces. These connection interfaces are then transformed into proxies by the generator; the proxy PxC at

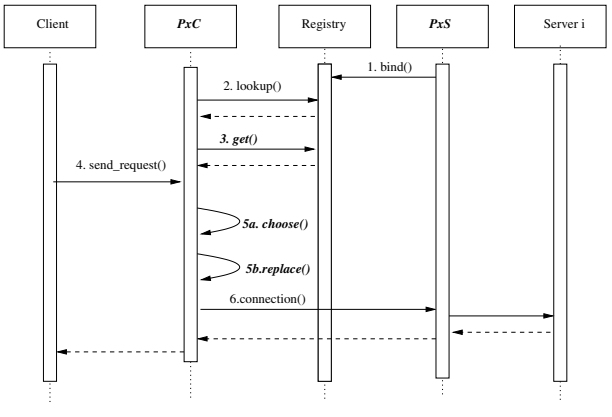


Fig. 7. Load balancing with a connector

client side and the proxy PxS at server side. These proxies have the responsibility of hiding the distribution details, choosing the server that will reply to the request, and recovering the workload. Indeed, monitoring load is not any more the responsibility of the servers but of their attached proxies. In figure 7, the sequence diagram illustrates the the sequence of events at deployment time for a round robin policy, after the generation of the proxies.

1. The servers register in the RMI Registry as usual by a *bind()* operation via the proxy PxS,
2. The references of all the registered servers are recovered by the proxy PxC,
3. This latter handles the list of the servers participating to the interaction,
4. When the client sends a request by invoking (locally by a procedure call) the server's service, the proxy PxS intercepts the call,
5. The proxy applies the logic of the policy:
   (a) It chooses the next component that will service the request,
   (b) It replaces the current reference by the chosen server reference,
6. Finally, the stub continues its initial tasks, marshalling and unmarshalling parameters, sending and receiving them towards and from the network, and returning the result to the client.

So this new proxy will have the same role of marshalling and unmarshalling parameters, the sole difference being that the internal reference of the remote object is changed by following the chosen load balancing policy.

This later solution is more adapted for this load balancing example. As this property is not related to the application functionality, it is recommended to use a connector since it makes the load balancing property transparent. We can use the same client and servers in an application with another communication property, *i.e.* using another connector like consensus for example, by a simple regeneration. The following section describes briefly the implementation of the load balancing connector and some experimental results.

4.4 Implementation of the Load Balancing Connector

We have realized the load balancing connector by specifying its properties and the associated generators to put on the shelf. The connector has to ensure both the distributed and load balancing properties. The associated generators can be implemented over different technologies. We can create them from an existing connector, like RMI, or by using another component. In the following, we are going to see two implementations of the load balancing connector, and the results of an experimentation.

Implementation: The first implemented generator is for the Round Robin policy. In this non adaptive policy, monitoring load is not necessary. We have realized the generator as an extension of the RMI generator. The load balancing code is injected in the generated proxy PxC. This latter is in charge of recovering all the references of the servers participating to the application that are registered

in the RMI registry. The client sends its requests to a server and the proxy intercepts them locally. When the proxy receives a request from the client, it performs the same marshalling and unmarshalling operations already existing in RMI. In addition, it has the responsibility of applying the load balancing policy. It forwards every new request of the client to the next server in the list (registry). This is done by replacing the current reference after each request.

The second implementation is for another load balancing connector, with this time an adaptive load balancing policy of the least loaded. In this policy, client requests are transmitted to servers according to their workload. In order to implement and realize the associated generator, we have used a component following the publish/subscribe model. This component receives the workloads from the different servers (the future generated proxies have this responsibility). It implements the algorithm of the policy to choose the least loaded server that will provide the service, and assigns the reference of this server to the proxy at client side that will establish the effective connection. We have combined this component with the RMI generator in order to realize the load balancing generator. The component publish/subscribe, the proxies, and the registry form henceforth the binding component.

Load Balancing Connector Results: In order to evaluate the whole connector life cycle, we have implemented a load balancing connector in the open source project Jonathan [11]. This framework basically offers a RMI-like connector. We have added generators to realize the load balancing connector with 2 variant strategies: the previously presented round-robin and the least loaded.

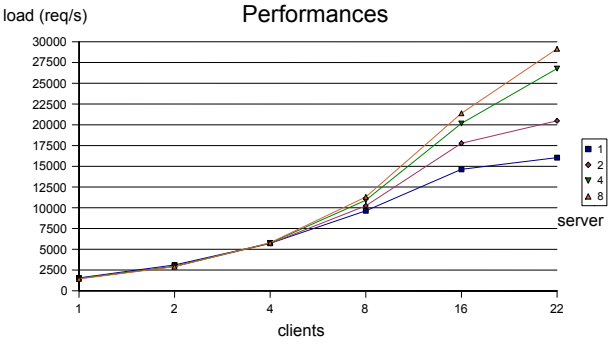


Fig. 8. Load Balancing performance (request per second)

We have evaluated performances of the new connector over a network of 32 machines with 1 to 8 servers and 1 to 22 clients[3]. The results The results obtained from a latency test of figure 8 showed no overhead due to reference switchings and demonstrated a good scalability. For example, the time needed to serve the

[3] One machine was reserved for a registry and another one for the control on the benchmark.

22 clients with one server using the load balancing connector was 1.85 ms, and the time needed to serve the same 22 clients with 8 servers was 0.75 ms. All the results can be consulted in [12].

It is worth noticing that previous Jonathan's generator can still be used to implement point-to-point RMI connections. So, introducing load balancing, or changing the load balancing strategy, requires a simple proxy regeneration from the same interface description.

5 Related Work

There are three main areas of related work: connector classification, generating implementation and the model of the Open Distributed Process (ODP).

The main work that has provided a significant classification of connectors is [13]. The authors give a conceptual framework for classifying connectors in order to make them easier to understand. The framework describes the connectors in terms of functional layers. The topmost layer is the service category which specifies the interaction services that a connector provides. The services are then refined into connector types, dimensions, subdimensions and values to help reasoning about their underlying, low level interaction mechanisms and to help create such complex interactions. However, there is no indication about the nature of these services. Are they explicit, in which case the communication or coordination abstraction would be implemented as a component, or implicit, in which case realized as a connector with generators? Besides the purpose of the interaction entities, our work proposes a reflection on the structure of these objects.

The second area is related to generating implementations of connectors. In [14], Garlan and Spitznagel define operators to create new complex connectors by composing simple ones. In our approach we can build complex connectors from simple ones (extension of RMI). We can also build them from a combination of a component with a generator (using a publish subscribe component [12]). Their approach is simple to implement, because they just reuse existing connectors without any modification. However, this involves several intermediate steps because of the combination of several connectors. Thanks to the generation process, our approach enables us to change some interaction mechanisms that are not appropriate to the interaction requirement. Moreover, Garlan and Spitznagel's implementation passes directly from the specification to the binding component step through the connection—according to our life cycle. Their approach does not have off-the-shelf generators for different technologies, so they have to make the same transformations whenever a composition has to be created over a different platform. We argue that the two approaches are complementary. Owing to the fact that our approach is more complex to realize, we believe it to be more efficient, since the transformations they propose could be automated with the development of generators.

In ODP [15] the connection between components is realized through binding objects. At an abstract level, the Conceptual Model is defined and is mapped to the implementation, i.e. the Engineering Model. But no distinction is made on the

nature of connections as we do. The ODP standard does not define any process to derive the binding objects from the conceptual model. In [16], Blair and Stefani said: "There is a strong correspondence between a description of a system in the conceptual model and the corresponding description in the engineering model in ODP". This is a strong assumption. In our work we propose to distinguish 2 ways of reusing and building connections. One through communication components, and the other through connectors and their implemented form, generators. The way the gap is filled is either through a refinement process (communication component case) either through a generation process (connector case).

6 Conclusion

We argued in this paper that the well accepted architectural concept of connection needs a better model. If software architecture relies on both components and connections, the latter have many forms and uses; they need to be studied more precisely. On the one hand, software architecture is somehow so abstract that it hides too many details and creates ambiguity and difficulty of reuse. On the other hand, at implementation and deployment level everything appears as components linked by procedure calls (PC) or remote procedure calls (RPC).

In order to fill the gap between the architecture level and implementation one, we propose to distinguish two kinds of connections, both of them being responsible for communication and coordination activities as architectural connectors should, but with different development process.

The first one is called medium. They can be considered as components since *they possess explicit interfaces*, but, are different from usual application components since *their interfaces are offered on different sites*. The last point makes them different from usual component implementations such as .NET, EJB or CCM. They are developed with a classical refinement process but their deployment process is unusual and not necessarily atomic. Protocols, publish/subscribe are examples of mediums.

The second kind, is called connector. *It possesses implicit interfaces* called plugs that adopt the explicit interfaces they are linked to. A connector is implemented as a code generator and is not refined. It has its own life cycle. The reusable part of a connector is not a component that has to be deployed, but the generator that generates the components that implement the connection and are actually deployed. Load-balancing, consensus are examples of connectors. Connectors are connection entities that affect communication properties such as reliability, security, efficiency, precision for instance.

We argue that both kinds of connections are necessary at software architecture level. And we claim that components are not the only way of developing reusable parts of software architecture, but that a significant part of the effort should be made in the development of connector generators in order to improve the abstraction of software architecture connections. The choice of a medium (explicit interfaces) or connector (implicit interfaces) is a design choice that affects the whole architecture. For instance, if we choose to use a publish/subscribe

medium, either the client component is compatible with the medium and the assembly is immediate either the client is incompatible and an adapter (glue) need to be generated or developed to realize the assembly.

There are two reuse ways: once that relies on (prefabricated) components and one that relies on connectors implemented as generators that produce implementation components. For each connector, a set of generators may propose variants for different system targets or with different qualities (security, reliability, etc.) Hence, the shelves contain components and generators.

In order to change some properties of the system without changing the abstract (or functional) architecture it is possible to change the connector and to re-generate the connection. The interest to identify and implement high-level connector is to simplify the substitution of their implementation by a simple regeneration. We have implemented and tested the feasibility of such an approach with the installation of load balancing in a client server architecture by a simple substitution.

The limitation of our approach is that connections that do not appear at the architecture level can not be generated. We support a limited kind of dynamicity since component instances can be dynamically created or destroyed as long as they satisfy the specified architecture. In order to reach a fully dynamic architecture, a solution would be to use some kind of reflection (or introspection) over the architecture itself.

Until now, the deployment and the generator development are ad-hoc. Hence, future work includes studies on the deployment process and its mechanisms and on generator implementation from a connector specification.

References

1. OMG: CORBA Component Model RFP (1997) `http://www.omg.org/docs/ orbos/97-05-22.pdf`.
2. Inc, S.M.: (Enterprise java beans technology) `http://java.sun.com/products/ejb/`.
3. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languges. In: IEEE Transaction on Software Engineering. (2000) 70–93
4. Abowd, G.D., Allen, R., Garlan, D.: Using style to understand descriptions of software architectures. ACM Software Engineering Notes **18** (1993) 9–20
5. Cariou, E.: Contribution à un Processus de Réification d'Abstractions de Communication (in french). PhD thesis, Université de Rennes 1, école doctorale Matisse (2003)
6. Szyperski, C.: Component Software. Addison-Wesley (1999)
7. Cariou, E., Beugnard, A.: The Specification of UML Collaborations as Interaction Components. In: UML 2002 – The Unified Modeling Language. (2002)
8. Putman, J.: Architecting with RM-ODP. Prentice Hall (2001)
9. Matougui, S., Beugnard, A.: How to implement software connectors? a reusable, abstract and adaptable connector. In: Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece (2005)

10. Cariou, E., Beugnard, A., Jézéquel, J.M.: An Architecture and a Process for Implementing Distributed Collaborations. In: The 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002). (2002)
11. Middleware, O.O.S.: (Jonathan: an Open Distributed Objects Platform) `http://jonathan.objectweb.org/`.
12. Sanchez, F.J.I., Matougui, S., Beugnard, A.: Conception et implémentation de connecteurs logiciels : Une expérimentation pour le calcul de performance sur une application de répartition de charge. Technical report, ENST-Bretagne, Brest, France (2004)
13. Mehta, N.R., Medvidovic, N., Phadke, S.: Towards a taxonomy of software connectors. In: the 22nd International Conference on Software Engineering (ICSE 2000). (2000) 178–187
14. Spitznagel, B., Garlan, D.: A compositional approach for constructing connectors. In: The Working IEEE/IFIP Conference on Software Architecture (WICSA'01). (2001) 148–157
15. IUT: Open Distributed Processing - Reference Model Part 1: Overview. Volume ITU-T Rec. X.901 — ISO/IEC 10746-1. (1995)
16. Blair, G., Stefani, J.B.: Open Distributed Processing and Multimedia. Addison Wesley (1997)

On the Notion of Coupling in Communication Middleware

Lachlan Aldred[1], Wil M.P. van der Aalst[1,2],
Marlon Dumas[1], and Arthur H.M. ter Hofstede[1]

[1] Faculty of IT, Queensland University of Technology, Australia
{l.aldred, m.dumas, a.terhofstede}@qut.edu.au
[2] Department of Technology Management,
Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. It is well accepted that different types of distributed architectures require different levels of coupling. For example, in client-server and three-tier architectures the application components are generally tightly coupled between them and with the underlying communication middleware. Meanwhile, in off-line transaction processing, grid computing and mobile application architectures, the degree of coupling between application components and with the underlying middleware needs to be minimised along different dimensions. In the literature, terms such as synchronous, asynchronous, blocking, non-blocking, directed, and non-directed are generally used to refer to the degree of coupling required by a given architecture or provided by a given middleware. However, these terms are used with various connotations by different authors and middleware vendors. And while several informal definitions of these terms have been provided, there is a lack of an overarching framework with a formal grounding upon which software architects can rely to unambiguously communicate architectural requirements with respect to coupling. This paper addresses this gap by: (i) identifying and formally defining three dimensions of coupling; (ii) relating these dimensions to existing communication middleware; and (iii) proposing notational elements for representing coupling configurations. The identified dimensions provide the basis for a classification of middleware which can be used as a selection instrument.

1 Introduction

Distributed application integration has some longstanding problems. For instance technology and standardization efforts supporting distributed interactions (Web services [3], MOM [11], MPI [20], RPC/RMI [14]) have made it possible to implement solutions to the difficult problems of distributed communication, however a general framework/theory for integration remains elusive. The problem seems to be one of finding the *right* abstractions [2].

Researchers seems to agree that the problem of communicating autonomous systems is not well understood [6, 2, 7, 13] and yet middleware vendors appear

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1015–1033, 2005.

confident that the problem is well understood, at least with respect to their tools. While there are incredibly complex problems yet to be solved in the area of middleware, perhaps those issues and aspects related to coupling/decoupling lie at the very heart of the problem. In their paper about the semantics of blocking and non-blocking send and receive primitives, Cypher & Leu state that "unfortunately, the interactions between the different properties of the send and receive primitives can be extremely complex, and as a result, the precise semantics of these primitives are not well understood" [6].

Vendors and standards bodies offer many solutions and ideas within this domain, however each appears to be embroiled in the paradigm it emerged from. For instance CORBA is founded and based on an RPC paradigm, whereas MOM[1] platforms are based on a strongly decoupled paradigm involving a hub architecture, with a MOM service at the centre. Despite their different origins each appears to have technical and conceptual limitations.

Due to the lack of a widely accepted formal theory for communication and the fact that each type of solution is based on a totally different paradigm, models and implementations of middleware based communication are disparate, and not portable. Rephrased, the models over deployed distributed systems are not portable in the implementation sense, and not even in the conceptual sense. Due the lack of formality in this domain, solutions and tools for integration are difficult to compare as well.

Objectives of the Paper. This paper aims at contributing to address the lack of foundation for expressing architectural requirements and for assessing middleware support in terms of decoupling. The main contributions are:

- A detailed analysis of the notion of (de-)coupling in communication middleware.
- A collection of notational elements for expressing architectural requirements in terms of (de-)coupling. These notational elements are given a visual syntax extending that of Message Sequence Charts [18] and can thus be integrated into UML sequence diagrams[2]. In addition, the notational elements are given a formal semantics in terms of Coloured Petri Nets (CPNs) [12].
- An approach to classify existing middleware in terms of their support for various forms of (de-)coupling. This classification can be used as an instrument for middleware selection.

Scope. A complete, formal analysis of communication middleware would be a daunting task. The list of options and functionality of middleware is incredibly long, particularly when one considers, for example privacy, non-repudiation, transactions, reliability, and message sequence preservation. Therefore this work chooses not to take too broad an analytical view of the domain. We have chosen to focus in on the aspect of *decoupling* because it seems to lie at the very heart

[1] Message Oriented Middleware: middleware systems capable of providing support for decoupled interactions between distributed endpoints.

[2] http://www.uml.org

of the problem of making two or more endpoints communicate effectively, and is central to the design of distributed applications.

In a recent survey of publish/subscribe technologies Eugster et. al. [7] identified three primary dimensions of decoupling offered by MOM. These are:

- *Time Decoupling* - wherein the sender and receiver of a message do not need to be involved in the interaction at the same time.
- *Space Decoupling* - wherein the address of a message is directed to a particular symbolic address (channel) and not the direct address of an endpoint.
- *Synchronisation Decoupling* - wherein the threads inside an endpoint do not have to block (wait) for an external entity to reach an appropriate state before message exchange may begin.

These three dimensions of decoupling by Eugster et. al. form the base of our work. We believe, that they operate as a core middleware paradigm. Despite their crucial role they are not well understood [6], and to the best of our knowledge have not been the subject of a formal analysis.

Organisation of the Paper. The paper is structured as follows. Section 2 defines some basic concepts. Section 3 identifies a set of decoupling dimensions and defines basic elements for expressing architectural requirements in terms of these dimensions. Next, Section 4 shows how these elements, as well as their formal definitions in terms of CPNs, can be composed in order to capture requirements across multiple coupling dimensions. Section 5 concludes the paper with a discussion, and it outlines related and further work.

2 Background

This section provides an overview of background knowledge related to the study of decoupling in communicating middleware. It defines some essential terms used throughout the paper.

An *endpoint* is an entity that is able to participate in interactions. It may have the sole capability of sending/receiving messages and defer processing the message to another entity, or it may be able to perform both.

An *interaction* is an action through which two endpoints exchange information [17]. The most basic form of this occurs during a message exchange (elementary interaction).

A *channel* is an abstraction of a message destination. Middleware solutions such as JMS [10], WebsphereMQ [15], and MSMQ [16] use the term "queues" to mean basically the same thing, but the crucial point here is that a channel is a logical address, not a physical one, thus they introduce a space decoupling to traditional point-to-point messaging. Thus the sender is able to address a message to a symbolic destination, and the receiver may register the intention to listen for messages from the symbolic destination. Interestingly such a decoupling means that a channel is not necessarily restricted to one message receiver - many *endpoints*, may share one channel, in which case they may either share/compete for

each message, depending on whether or not the channel is a publish/subscribe. Channels have been enhanced and extended with many functions (e.g. sequence preservation [7, 6], authentication, and non-repudiation [11]).

A *message* is a block of data that gets transported between communicating endpoints. Depending on the middleware it could contain header elements such as an message ID, timestamp, and datatype definition; or it could just contain data. The message may contain a command, a snapshot of state, a request, or an event among other things. It may be transactional, reliable, realtime, or delayed, and it often is transported over a "channel". However, it could just as easily be sent over sockets.

3 Decoupling Dimensions of an Interaction

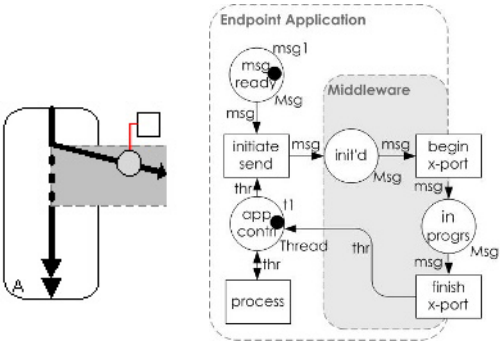
In this section we will provide a Coloured Petri net (CPN) based analysis of fundamental types of decoupling for interacting endpoints. These *dimensions* of decoupling have relevance to all communication middleware we are familiar with, including MOM, space-based middleware [8], and RPC-based paradigms. Coloured Petri nets were chosen for their ability to explicitly model state, parallelism, and data. Coloured Petri nets also have a strict formal semantics giving the communication primitives a level of precision that would otherwise be difficult to attain.

3.1 Synchronisation

The critical concept behind synchronisation decoupling is that of "non-blocking communication", for either, or both of, the sender and receiver. Non-blocking communication allows the endpoints to easily interleave processing with communication. In the following paragraphs we introduce some notational elements for denoting various forms of synchronisation decoupling as well as a formalisation of these notational elements in terms of CPNs.

Send
A message send can either be blocking or non-blocking. *Blocking send* implies that the sending application must yield a thread until the message has truly left it. Figure 1(b) is a CPN[3] of a blocking send. The outer dashed line represents the endpoint while the inner dashed line represents middleware code that is embedded in the endpoint. When a message is ready (represented by a token inside the place "*msg-ready*") and the application is ready (represented by a token inside the place "*app-contrl*") the endpoint gives the message to the embedded middleware. The endpoint in blocking send also yields its thread of control to the embedded middleware, but it gets the thread back when the message has

[3] Note: This paper presents a range of Coloured Petri nets (CPNs) [12] that model important aspects of decoupled systems. All CPNs were fully implemented and tested using CPN Tools [4].



(a) Notation. (b) CPN - Blocking send.

Fig. 1. *Blocking send.* After initialising send, the transition "*process*" cannot fire until a thread is returned at the end of message transmission.

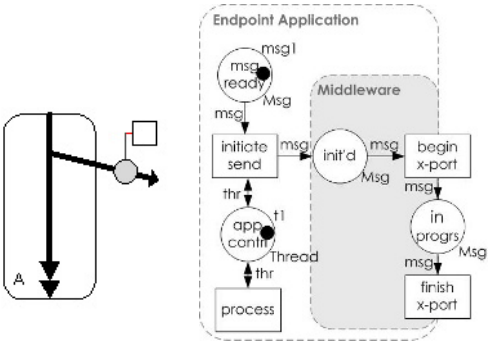
completely left the embedded middleware. Note that inside the embedded middleware the transitions "*begin-x-port*", "*in-progress*", and "*fin-x-port*" hang over the edge of the embedded middleware. This was done to demonstrate that the remote system (receiver endpoint or middleware service) will bind to the sender by sharing these transitions and one state. This implies that hidden inside the middleware, communicating systems exchange information in a time coupled, synchronisation coupled manner, regardless of the behaviour that's exposed to the the endpoint applications. In CPN terminology certain nodes inside the embedded middleware are "transition bounded"[4].

In a blocking send there is a synchronisation coupling of the sender application (endpoint) with something else - but not necessarily the receiver as we will show in Section 3.2.

Synchronisation decoupling is achieved from the viewpoint of the sender only if it is possible to perform a *non-blocking send*. A non-blocking send is observable in a messaging interface if the message send operation can be initiated from the application and then the middleware embedded inside the sender returns control immediately, (i.e. before the message has left the application place). See Figure 2 for an illustration and CPN of the concept. Note that this Figure like that of blocking send (Figure 1) is transition bounded with remote components through the transitions in the embedded middleware of the application. Snir and Otto provide a detailed description of non-blocking send [20].

Non-blocking send is a necessary condition, but not a sufficient condition to achieve total synchronisation decoupling, which is to say that the receive action must also be non-blocking. If both send and receive are blocking (non-blocking) then a total synchronisation coupling (decoupling) occurs. A partial synchronisation decoupling occurs when the send and the receive are not of the same blocking mode (i.e. one is blocking with the other being non-blocking).

[4] "Transition bounded", in this context, means that two distributed components share a transition (action), and must perform it at exactly the same moment.



(a) Notation. (b) CPN - Non-blocking send.

Fig. 2. *Non-blocking send.* The transition "*process*" can be interleaved with communication because a thread is not yielded to the embedded middleware.

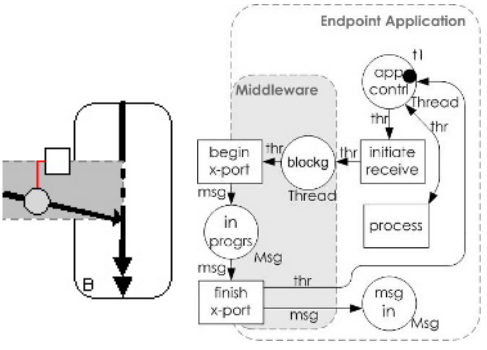
Non-blocking send is a fairly uncommon feature of middleware solutions. For instance all RPC-based implementations use blocking send, and many/most MOM implementations use a blocking send as well. This is the case because even though MOM decouples the sender from the receiver through time, the senders are typically synchronisation coupled to the middleware service. This is acceptable when the sender is permanently connected over a reliable network to the provider, however mobile devices, for instance, typically need to interoperate on low availability networks, hence they require the ability to store the message locally, until the network is available (i.e. store and forward) [13]. This problem should obviously not be too great a burden on the mobile applications developer, and should be part of the middleware functionality.

Receive

Like message send, message receipt can either be blocking or non-blocking [6]. The definition of *blocking receive* is that the application must yield a thread into a waiting state in order to receive the message (the thread is usually returned when the message is received). This means that the receiving application is synchronisation coupled to either the message sender or the middleware service (depending on the whether the middleware is peer-to-peer or server-client oriented). Figure 3 presents a model of this concept.

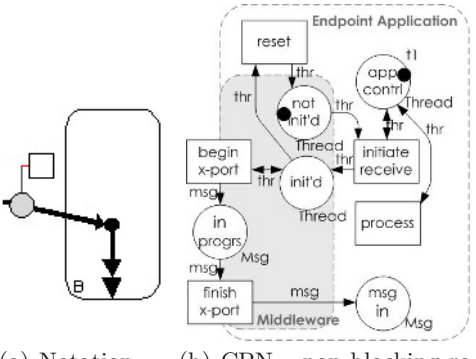
On the other hand, *non-blocking receive*, is a messaging concept wherein the application can receive a message, without being required to yield a thread to achieve this. This concept is illustrated in Figure 2. A well known example of non-blocking-receive is that of the event-based handler, as described in the JMS [10]. A handler is registered with the middleware and is called-back when a message arrives. MPI [20] provide an equally valid non blocking receive (without callback) that is not event based[5].

[5] Essentially the receiver application polls its own embedded middleware.



(a) Notation. (b) CPN - Blocking receive.

Fig. 3. *Blocking receive.* A thread must be yielded to the embedded middleware until the message has arrived.



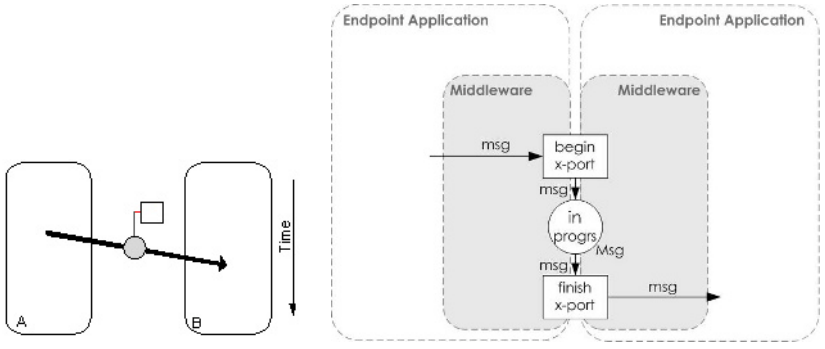
(a) Notation. (b) CPN - non-blocking receive.

Fig. 4. *Non-blocking receive.* A thread need not be yielded to the middleware in order to receive.

Non-blocking receive seems less frequently used than the blocking receive. This is probably because blocking receives are simpler to program and debug [11]. One frequently observes statements in the developer community that MOM enables asynchronous interactions (which is true in the it allows time decoupled interaction), however this general usage of "asynchronous" for MOM is misleading because MOM usually connects endpoints with a blocking-send and blocking receive (synchronisation coupled).

3.2 Time

The dimension of time decoupling is crucial to understanding the difference between many peer-to-peer middleware paradigms and server oriented paradigms



(a) Notation. (b) CPN - time coupled messaging.

Fig. 5. *Time coupling* is characterised by transition-bounded systems

(e.g. MPI versus MOM). In any elementary interaction time is either coupled or decoupled.

Time coupled interactions are observable when communication cannot take place unless both endpoints are operating at the same time. Hence peer-to-peer systems are always time coupled. In time coupled systems the message begins by being wholly contained at the sender endpoint. The transition boundedness of endpoints can guarantee that the moment the sender begins sending the message, the receiver begins receiving. The concept is presented in Figure 5 wherein the endpoint applications are joined directly at the bounding transitions ("*begin x-port*" and "*fin x-port*").

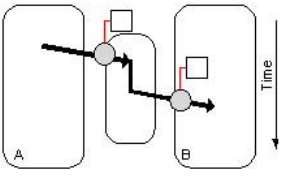
Time decoupled interactions allow messages to be exchanged irrespective of whether or not each endpoint is operating at the same time. Therefore simple peer-to-peer architectures cannot provide true time decoupling - by definition. What is required for time decoupling is a third participant in the interaction where the sender can deposit the message, and the receiver can retrieve it. This is why many MOM implementations use a client-server architecture. Servers may be redundant, and even use "store and forward" semantics between servers [15], in which case the term "peer-to-peer" is often used ambiguously. Though this could be better described as a polygamous architecture where many machines host both peer-endpoints, and a messaging server.

The concept of time decoupling is presented in Figure 6. Note that in the CPN and the illustration the separate systems are transition bounded in the same way as before, however this time there are three of them, with the middle one being a middleware service that is able to buffer the message.

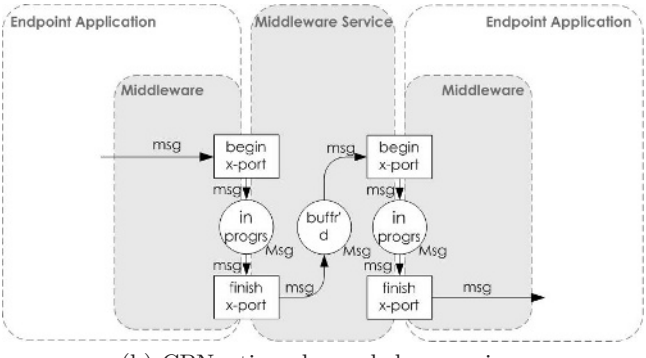
3.3 Space

Space is the third dimension describing the connectivity of distributed systems.

For an interaction to be *space coupled* the sender uses a direct address to send the message to. Therefore the sender "knows" exactly where to address the

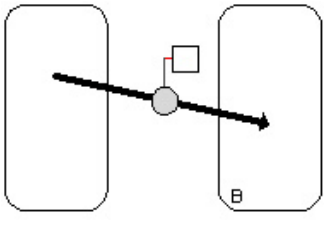


(a) Notation.

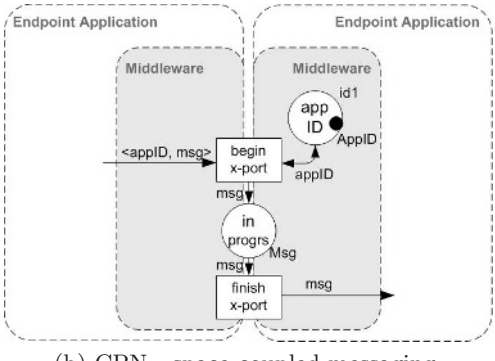


(b) CPN - time decoupled messaging.

Fig. 6. *Time decoupling* is characterised by systems that can be strictly non-concurrent (endpoint to endpoint), and still communicate



(a) Notation.

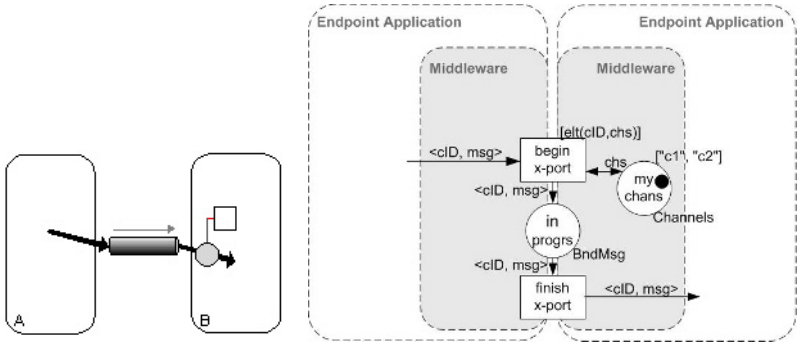


(b) CPN - space coupled messaging.

Fig. 7. *Space coupling.* The sender directly addresses the receiver.

receiver application. Figure 7 presents the concept of space coupling. This can be seen in the CPN by introducing a new type of token ($< appID, msg >$), and a new place ("*id*") as input to the transition "*begin x-port*". Only when two conjoined systems match on the value of "*appID*" will the bounding transition fire.

Space decoupled interactions on the other hand allow for a sender to have no explicit knowledge of the receiver's address. This makes it possible for parts of distributed systems to be extended, and replaced at runtime without shutting



(a) Notation. (b) CPN - space decoupled messaging.

Fig. 8. *Space decoupling.* The sender does not directly address the receiver, but directs the message along a channel.

everything down. Hence space decoupling is highly desirable from the viewpoint of enterprise integration due to its support for maintenance and management.

Figure 8 introduces the concept of an abstract message destination - or channel for space decoupled point to point messaging[6]. This is the logical message destination, but the actual message destination obtains its message off the same channel. The CPN demonstrates this by linking the transition "*begin x-port*" to a new input place ("*my chans*") and a slightly different token containing the message ($< cID, msg >$), of type BoundMessage (BndMsg). Hence, this transition shall only fire when two systems are bound together that satisfy the transition guard "*[elt(cID,chs)]*".

3.4 Summary

The dimensions of decoupling include synchronisation-decoupling (with its four options), time-decoupling, and space-decoupling. Each has its own precise behaviour and semantics. These were rendered using a reasonably intuitive graphical notation and a more precise formal semantics as presented by the CPNs.

4 Combining Synchronisation, Time, and Space

The dimensions of decoupling presented in the previous section are orthogonal to each other. Therefore designs for interactions can be composed from them arbitrarily while preserving the innate semantics, as fully defined for each - contributing to a precise overall behaviour. This set of configurations can then be used as a palette of possible interaction behaviours and thus applied to an integration problem or to the selection of an appropriate middleware product.

[6] Note that this series of CPNs models point to point messaging (i.e. as opposed to publish/subscribe). Extending the CPNs to describe publish/subscribe, while beyond the scope of this paper, is straightforward.

4.1 Compositional Semantics

Any type of synchronisation-decoupling (for both send and receive) can be combined with any type of time-decoupling, which in turn can be combined with any type of space-decoupling. This means that for one directional messaging there are sixteen possible interaction behaviours, definable according to the decoupling properties ($2^2 * 2 * 2 = 16$), and we believe that these dimensions are orthogonal. Meaning, for example, that you can have a time-coupled, synchronisation-decoupled interaction, and it is equally possible to have a time-decoupled synchronisation-coupled interaction.

Composing the Coloured Petri Nets
The CPNs for each coupling dimension from Section 3 can also be composed, or *overlayed* to form a complete model of any of the sixteen possible interaction behaviours. For example to create a CPN of a synchronisation-decoupled,

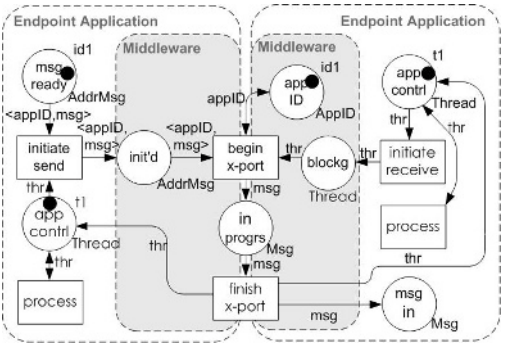


Fig. 9. CPN of a synchronisation-coupled, time-coupled, space-coupled interaction between endpoints

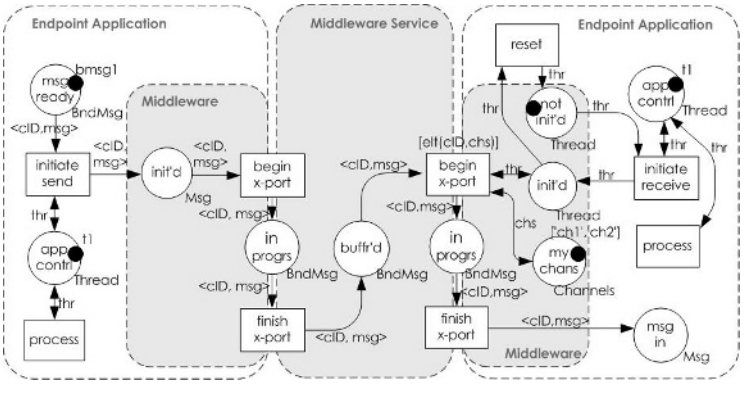


Fig. 10. CPN of a synchronisation decoupled (non-blocking send/receive), time-decoupled, and space-decoupled interaction

time-decoupled, and space-decoupled interaction one may use the CPNs from Figures 2(b), 4(b), 6(b), and 8(b), and overlay them. Such a net is presented in Figure 10. A CPN for a synchronisation-coupled, time-coupled, space-coupled also shown (Figure 9), however, for space reasons we do not present the remaining fourteen composed CPNs.

Graphical Notation of Compositions

The entire set of sixteen possible decoupling configurations possible are enumerated in graphical form in Figures 11 and 12. These graphical illustrations of the

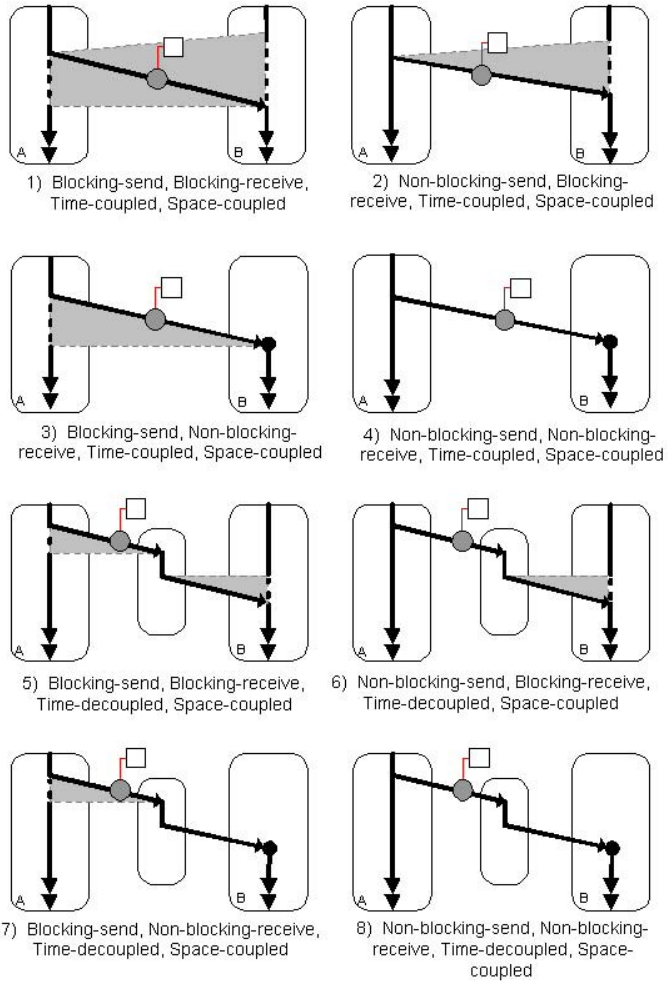


Fig. 11. Notations 1 - 8 of the coupling configurations for one way communication

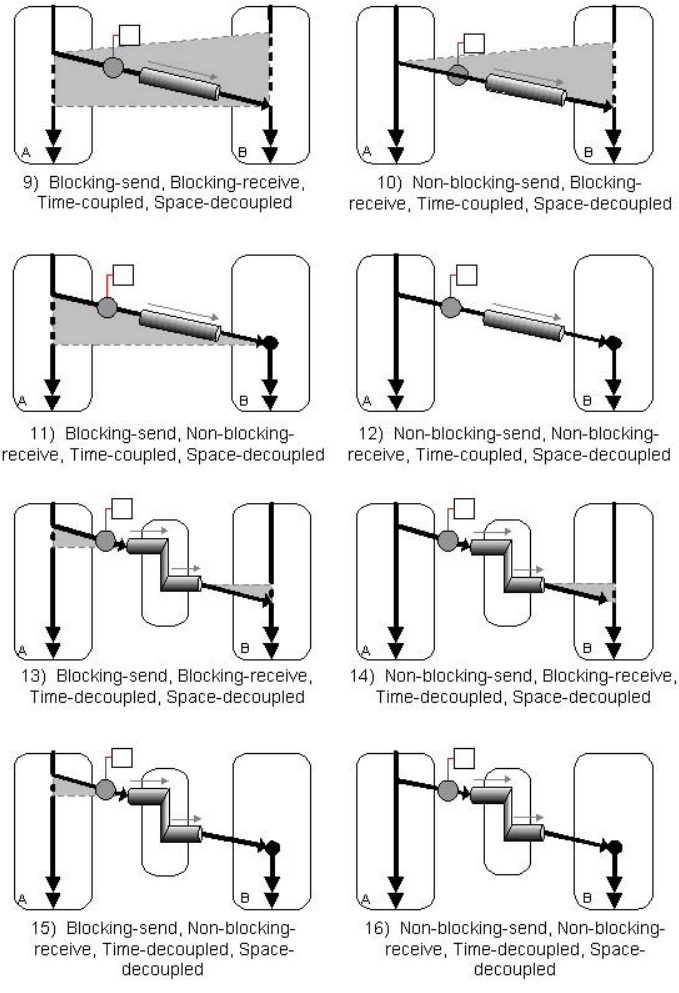


Fig. 12. Notations 9 - 16 of the coupling configurations for one way communication

possibilities are essentially arrived at by overlaying the illustrative vignettes of the dimensions of decoupling as presented in Section 3.

This graphical notation for the different types of coupling for elementary interactions could prove to be useful in defining requirements, or system analysis and design. The configurations are varied, and each one has its own specific behaviour. Furthermore they are sufficiently different that some will be more suitable to a given integration problem than other. Put another way not all possible coupling configurations would be useful in any situation. For instance a multiplayer realtime strategy game would not have much use for a time-decoupled configuration.

4.2 Example of Capturing Coupling Requirements in an Integration Project

Imagine that a hospital needs to integrate a new BPM system and a new proximity sensor system to its existing IT services. Each doctor, and nurse is given a mobile device which can inform a central system of the location of that person inside the hospital. This is linked to the BPM system so that the nearest staff member with the requisite skills can be notified of new work and notified during emergencies.

The challenge is to design a conceptually clean, integration model showing the types of connectivity between the different endpoints in such a system. Clearly the mobile devices will not always be connected to the central systems (due to possible signal interference), and therefore non-blocking send is advisable, this way messages from the device could be stored until the signal is restored. New mobile devices might need to be added to the system, and device swapping may occur, and shouldn't break the system. Therefore space decoupling is required. Finally, device batteries might go flat and therefore time decoupling between mobile devices and the central system is necessary. Hence the architecture of this endpoint interconnection should be either configuration '14' (Non-blocking-send, Blocking-receive, Time-decoupled, Space-decoupled) or '16' (Non-blocking-send, Non-blocking-receive, Time-decoupled, Space-decoupled). During requirements analysis we can proceed through all communication endpoints this way.

4.3 Comparison of Middleware Systems

We postulate that any type of middleware could be plotted against the 16 coupling configurations proposed in this Section with respect to whether they directly support the defined behaviour through their API or interface. As part of this research we have plotted the coverage of middleware solutions and standards against the proposed coupling configurations. The set of solutions and standards includes MPI[7], Biztalk Server 2004 (Product Documentation), Websphere MQ[8], Java-NIO[9], Java-RMI[9], Java-Sockets[9], Java-Mail[10], JMS[10], Javaspaces[11], CORBA [9], RPC[12]. In most cases the documentation (as opposed to implementations) for these standards and solutions was used as a guide to determine their coverage.

Table 1 represents our initial assessment of various well known middleware solutions and assesses each one's ability to directly support each coupling configuration (hence that communication behaviour).

The hospital scenario previously introduced, requires either configurations 14 or 16, these are both empty in our tables, however MobileJMS [13] is a proposal that will support them.

[7] MPI Core: V. 2, [20].
[8] Websphere MQ V 5.1, [15].
[9] JDK V. 1.4, `http://java.sun.com/j2se/1.4.2/docs/api`, accessed June 2005.
[10] J2EE-SDK V. 1.4, `http://java.sun.com/j2ee/1.4/docs/api`, accessed June 2005.
[11] Java Spaces `http://java.sun.com/products/jini`, accessed June 2005.
[12] DCE-RPC V 1.1, `http://www.opengroup.org/onlinepubs/9629399/toc.htm`, accessed June 2005.

Table 1. Support for coupling configurations by some well known middleware solutions and standards

Space coupling	Synch coupling	Partial synch decoupling		Synch decoupling
	B-Send, B-Rcv	NB-Snd, B-Rcv	B-Snd, NB-Rcv	NB-Snd, NB-Rcv
Time coupled	MPI, Sockets 1	MPI, RPC-Reply 2	MPI, CORBA, RPC-Request 3	MPI, Java-NIO 4
Time decoupled	Java-Mail 5	6	7	8

Space decoupling	Synch coupling	Partial synch decoupling		Synch decoupling
	B-Send, B-Rcv	NB-Snd, B-Rcv	B-Snd, NB-Rcv	NB-Snd, NB-Rcv
Time coupled	9	Java-RMI-Reply 10	CORBA, Java-RMI-Request 11	12
Time decoupled	JMS, Websphere-MQ, BizTalk-MSMQ, JavaSpaces 13	14	JMS, Websphere-MQ, BizTalk-MSMQ, JavaSpaces 15	16

4.4 The Case of Two-Way Interactions

This work, while presented in terms of one directional communication, has application in compound interactions as well - for instance two-way communication. In RPC style interactions there are two roles being played. The role of requestor performs a "solicit-response", and the role of service provider performs a "request-response" [1]. The requestor typically blocks for the response, hence the interaction is generally considered synchronous. However, if such an interaction is modelled using the proposed notation it becomes clear that this broad definition is not totally precise. In actual fact the interaction is partially synchronisation decoupled, in each direction. The service provider uses non-blocking receive and in the reply uses non-blocking send (B-send → NB-receive → NB-send → B-receive). Therefore any model of such an interaction should capture these subtleties.

Using the proposed notation there are 16 possibilities in each direction for two way interactions. Hence if one enumerated all possible configurations of decoupling, for two way interactions, there are $16^2 = 256$ possibilities.

5 Conclusions

Discussion. This paper has presented a set of formally defined notational elements to capture architectural requirements with respect to coupling. The proposed notational elements are derived from an analysis of communication

middleware in terms of three orthogonal dimensions: space time and synchronisation. This analysis goes beyond previous middleware classifications by identifying certain subtleties with respect to time coupling. In previous communication middleware analyses, when two endpoints are coupled in time, they are generally considered to be synchronous, and in the reverse case they are considered to be asynchronous (e.g. [11]). However, such an imprecise definition does not provide any differentiation between sockets and RPC, which are both time-coupled. Clearly there is more to the problem than the generally held belief (that time-coupled implies synchronous). We consider that 'synchronous' and 'asynchronous' are too imprecise for using to create clear abstract models of integrations. The framework that we provide is the first that we know of that presents synchronisation coupling and time coupling as independent but related concepts.

By their very nature publish/subscribe based paradigms fit into the realm of space-decoupled systems. Space-decoupled system have publish/subscribe behaviour if their delivery semantics allow multiple copies of one message to be sent to more than one subscriber. This is contrasted with space-decoupled point-to-point systems, which restrict message delivery to one "subscriber". We consider space/time/synchronisation decoupling to be orthogonal to message multiplicity. Therefore classical multiplicities (i.e. point-to-point, and multicast) are possible irrespective of the chosen coupling configuration - provided that a middleware platform supports them.

Currently the rating of tools against the coupling configurations are binary (supported or not-supported), however an improvement to the rating system would include partial support (where the tool directly supports a restricted version of the concept; or the tool supports the concept, but requires some extra effort to implement it) would make the ratings more useable. The assessment would also benefit from a more detailed explanation of why a rating was given.

Related Work in Middleware Classification. Cross and Schmidt [5] discussed a pattern for standardizing quality of service control for long-lived, distributed real-time and embedded applications. This proposal briefly described a technology that would be of assistance. They outlined the technology as "configuration tools that assist system builders in selecting compatible sets of infrastructure components that implement required services". In the context of that paper no proposals or solutions were made for this, however the proposals of our paper perhaps provide a fundamental basis for the selection of compatible sets of infrastructure.

Tanenbaum and Van Steen [22] described the key principles of distributed systems. Detailed issues were discussed such as (un-)marshalling, platform heterogeneity, and security. The work was grounded in selected middleware implementations including RPC, CORBA, and the World Wide Web. Our work is far more focussed on coupling at the architectural level, and its formal basis allowed the formation of a well defined set of coupling primitives.

Tai and Rouvallou [21] created a classification structure for middleware based on a delivery model, a processing model, and a failure model. These models are very relevant and provide a solid framework for middleware comparison at many levels. Our work, on the other hand, is more of a conceptual aid in understanding the different possibilities for connecting systems together, and helps put the limitations of existing middleware solutions into perspective.

Schantz and Schmidt [19] described four classes of middleware: *Host infrastructure middleware* provides a consistent abstraction of an operating system's communication and concurrency mechanisms (e.g. sockets). *Distribution middleware* abstracts the distribution, and generally allows communication over heterogenous systems as if they were running in one stand-alone application (e.g. CORBA [9], and RMI [14]). *Common Middleware Services* group together middleware that can provide higher level services such as transactions, and security (e.g. CORBA and EJB). *Domain Specific Middleware Services* classify those that are tailored to the requirements of a specific real world domain, such as telecom, finance etc. (e.g. EDI and SWIFT). This classification provides a compelling high-level view on the space of available middleware, but it does not give a precise indication of the subtle differences between alternatives in the light of architectural requirements.

Thompson [23] described a technique for selecting middleware based on its communication characteristics. Primary criteria include blocking versus non-blocking transfer. In this work several categories of middleware are distinguished, ranging from conversational, through request-reply, to messaging, and finally to publish/subscribe. The work, while insightful and relevant, does not attempt to provide a precise definition of the identified categories and fails to recognise subtle differences with respect to non-blocking communication, as discussed later in the paper.

Cypher and Leu [6] provided a formal semantics of blocking/non-blocking send/receive which is strongly related to our work. These primitives were defined in a formal manner explicitly connected with the primitives used in MPI [20]. This work does not deal with space decoupling. Our research adopts concepts from this work and applies them to state of the art middleware systems. Our research differs from the above, by exploiting the knowledge of this work in terms of non-blocking interactions, and combining this with the principles of time and space decoupling originating from Linda [8]. Our work is also unique in its proposal for compositional dimensions of decoupling, and our communication primitives may be used as a basis for middleware comparison.

Disclaimer. The assessments we made of middleware products and standards with respect to the coupling configurations are based on the tool or standard documentation. They are true and correct to the best of our knowledge.

Acknowledgement. This work is partly funded by an Australian Research Council Discovery Grant "Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages". The third author is funded by a Queensland Government Smart State Fellowship.

References

1. W. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, Feb 2003.
2. A. Beugnard, L. Fiege, R. Filman, E. Jul, and S. Sadou. Communication Abstractions for Distributed Systems. In *ECOOP 2003 Workshop Reader*, volume LNCS 3013, pages 17 – 29. Springer-Verlag Berlin Heidelberg, 2004.
3. R. Chinnici, M. Gudgin, J. Moreau, J. Schlimmer, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, 2004. `http://www.w3.org/TR/wsdl20` accessed June 2004.
4. CPN tools homepage. `http://wiki.daimi.au.dk/cpntools/_home.wiki` accessed March 2005.
5. Joseph K. Cross and Douglas C. Schmidt. Applying the quality connector pattern to optimise distributed real-time and embedded applications. *Patterns and skeletons for parallel and distributed computing*, pages 209–235, 2003.
6. R. Cypher and E. Leu. The semantics of blocking and nonblocking send and receive primitives. In H. Siegel, editor, *Proceedings of 8th International parallel processing symposium (IPPS)*, pages 729–735, April 1994.
7. P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.
8. David Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.
9. Object Management Group. *Common Object Request Broker Architecture: Core Specification*, 3.0.3 edition, March 2004. `http://www.omg.org/docs/formal/04-03-01.pdf` accessed June 2004.
10. M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Haase. *Java Messaging Service API Tutorial and Reference*. Addison-Wesley, 2002.
11. G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Boston, MA, USA, 2003.
12. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
13. M. Kaddour and L. Pautet. Towards an adaptable message oriented middleware for mobile environments. In *Proceedings of the IEEE 3rd workshop on Applications and Services in Wireless Networks*, Bern, Switzerland, July 2003.
14. Sun Microsystems. Java remote method invocation specification. `http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmi-title.html` accessed March 2005, 2003.
15. Websphere MQ family. `http://www-306.ibm.com/software/integration/wmq/` accessed June 2005.
16. Microsoft Message Queue (MSMQ) Center. `http://www.microsoft.com/windows2000/technologies/communications/msmq` accessed October 2004.
17. D. Quartel, L. Ferreira Pires, M. van Sinderen, H. Franken, and C. Vissers. On the role of basic design concepts in behaviour structuring. *Computer Networks and ISDN Systems*, 29(4):413 – 436, 1997.
18. E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.
19. R. Schantz and D. Schmidt. *Encyclopedia of Software Engineering*, chapter Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications. Wiley & Sons, New York, USA, 2002.

20. M. Snir, S. Otto, D. Walker S. Huss-Lederman, and J. Dongarra. *MPI-The Complete Reference: The MPI Core*. MIT Press, second edition, 1998.
21. S. Tai and I. Rouvellou. Strategies for integrating messaging and distributed object transactions. In *Proceedings of Middleware 2000: IFIP/ACM International Conference on Distributed Systems Platforms, New York, USA.*, pages 308–330. Springer-Verlag, 2000.
22. A. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
23. J. Thompson. Toolbox: Avoiding a middleware muddle. *IEEE Software*, 14(6):92–98, 1997.

A Task-Type Aware Transaction Scheduling Algorithm in J2EE

Xiaoning Ding[1,2], Xin Zhang[1,2], Beihong Jin[1], and Tao Huang[1]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, China
{dxn, zhangxin, jbh, tao}@otcaix.iscas.ac.cn
[2] Graduate School of Chinese Academy of Sciences, Beijing, China

Abstract. A large number of J2EE applications use entity beans as their application persistence mechanism, while current J2EE scheduling to entity beans is still a simple First-Come First-Served (FCFS) policy. Due to this simple policy, the scheduling does not distinguish the key transactions from the trivial ones. A task-type aware scheduling algorithm should be adopted to provide better service for key transactions. However, some characteristics of J2EE middleware transaction processing, such as the absence of necessary scheduling information and the interactive executing model, made it impossible to apply the traditional priority-driven scheduling algorithms. This paper puts forward a scheduling algorithm named TMPBP, which offers a new dynamic priority assignment algorithm HRS that can recognize the key tasks at runtime heuristically. Additionally, TMPBP add some extra techniques to the traditional concurrency control policies according to the features of J2EE. The algorithm is proven to be safe and will not produce starvation or priority inversion. The experimental evaluations indicated that TMPBP improves the Quality of Service (QoS) of key tasks effectively, especially when the server is under a heavy load.

1 Introduction

J2EE[1] is a primary middleware platform for distributed object-oriented applications. An increasingly large number of J2EE applications use entity beans as their persistence mechanism, which is an object-style encapsulation to the relational data from the underlying databases. The J2EE application server usually manages the concurrency control of entity beans by way of a locking mechanism. Additionally, the application server may adopt some extra techniques to improve the accessing performance, such as caching.

Since the locality of references, lots of applications concentrate on a small group of entity beans when the application server is under a heavy load. With the conflicts of accessing, a considerable part time of transaction life cycle is consumed on waiting for locks. However, the J2EE transaction scheduling nowadays is still a simple First-Come First-Served (FCFS) algorithm. The FCFS algorithm cannot distinguish the key tasks from the trivial ones and thus reduce the performance of key tasks.

The system should have a preference for key tasks when it is under a heavy load, which suggests that a priority-driven scheduling policy should be adopted to provide a

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1034 – 1045, 2005.

better Quality of Service (QoS) for key tasks. There is still no a commonly agreed definition to the QoS of transaction service until now [2], but some necessary metrics should be included in transaction QoS, such as average complete time, missed deadline percentage, etc. In this paper, we do not intend to discuss the executing mechanism of transaction QoS, and just define some QoS metrics to scale the scheduling in a quantitative way.

Lots of priority-driven scheduling algorithms have been proposed, such as the classical Least Slack First (LSF) [3,4] and Earliest Deadline First (EDF) [5,6]. However, the tasks submitted to J2EE middleware are not associated with some necessary scheduling information, such as the estimated executing time, the value of the task, etc. The absence of necessary scheduling information makes it impossible to apply these existing algorithms to J2EE. Furthermore, the transactions in J2EE are executed in an interactive way and the transaction manager cannot restart the transaction directly, which also prevents the application of some existing algorithms.

To overcome the above obstacles, this paper puts forward a task-type aware transaction scheduling algorithm, which can recognize the key tasks heuristically at runtime and improve the QoS of key tasks effectively. The rest of this paper is organized as following: the next section reviews the related work; section 3 presents the scheduling algorithm TMPBP (Threaded Multi-Queue Priority-Based Protocol) including the priority assignment algorithm HRS (Heuristic Resource Statistics); section 4 proves some characteristics of the algorithm; section 5 presents the performance experiments; and we conclude the paper in the last section.

2 Related Work

Most of the existing research on priority-driven scheduling comes from the area of database and real-time system, while little is available in J2EE middleware.

The tasks submitted to J2EE middleware are not associated with some necessary scheduling information, such as the estimated executing time, the value of the task, etc. And we cannot expect the users to provide static priorities for every transaction instances because: (1) The value of a task may change during its execution. (2) We must support the numerous legacy application codes that did not involve priority. What we need is a dynamic priority-driven scheduling algorithm.

There are many dynamic priority-driven scheduling algorithms have been proposed in real-time system, such as Least Slack First (LSF) [3,4], Earliest Deadline First (EDF) [5,6], Earliest Feasible Deadline First (EFDF) [7], Earliest Release First (ERF) [7], Highest Value First (HVF) [8], Greatest Value Density First (GVDF) [8,9], and so on.

Some algorithms (such as LSF and EFDF) cannot adapt to the J2EE middleware environment. For example, we cannot apply LSF without the estimated executing time. Some other algorithms (such as EDF and ERF) are not appropriate for our situation. For example, EDF does not consider the type of tasks and the key tasks cannot get a better service.

Underlying the HVF algorithm is that every task is associated with a value function of time t, and the scheduler should first schedule the task with highest value. However, the difficulty is how to construct a reasonable value function under J2EE. The problem also exists in GVDF.

As to concurrency control, usually a Two-Phased Locking (2PL) protocol is employed. Some optimistic concurrency control protocols are also applied in J2EE application servers such as JBoss [10,11]. However, when the server is under heavy load, there are many failures in the validation phase and the performance decreases sharply.

Priority Inheritance (PI) [12] is a traditional approach to prevent priority inversion. The basic idea of PI is that when a transaction blocks one or more higher-priority transactions, it is executed at the highest priority of all the transactions it blocks.

A priority-driven scheduling algorithm may produce a starvation, i.e. a transaction instance is lunched again and again, but every time it has timed-out before it completes. Most of the existing approaches [13] are restarting the instance and limiting the times of restarting. However, in J2EE middleware, the transaction is executing in a interactive way and the transaction manager cannot restart the transaction directly. We have to bind some scheduling information to the thread context so that we can recognize the timed-out instance when it is launched again.

As to deadlocks, some advanced approaches can destroy the preconditions of deadlocking. Data-Priority-Based Locking Protocol (DP) is such a deadlock-free locking protocol based on prioritizing data items proposed in [13]. In DP, each data item carries a priority which is equal to the highest priority of all transactions currently in the system that includes the data item in their access lists. When a new transaction arrives at the system, the priority of each data item to be accessed is updated if the item has a priority lower than that of the transaction. When a transaction terminates, each data item that carries the priority of that transaction has its priority adjusted to the highest priority active transaction that is going to access that data item.

DP assumes that the list of data items that are going to be accessed is submitted to the scheduler by the arriving transaction. However, as mentioned above, this assumption is not satisfied in J2EE. Such other deadlock-free algorithms also include Priority Ceiling Protocol [12,14]. But all of these approaches may decrease the executing efficiency remarkably in reality. The traditional approach, detecting the cycles in wait-for graph [15], may be more practical in reality.

3 Scheduling Algorithm

3.1 Notations

For the conveniences of discussing, we define some notations as following:

Definition 1(Notations on entity beans). All of the entity beans form the set BS. Each entity bean has the property *weight*, denoted as wt(bean), and wt(bean) $\in N^+$, where N^+ is the natural numbers set. All of the weight values in the system are marked as set WT, and the max value of weights is marked as max(WT).

Definition 2(Notations on transaction). The arrival time of a transaction instance α is marked as at(α), and the end time is marked as et(α). Each transaction instance has property *priority*. The priority of instance α at time t is marked as priority(α,t), and priority(α,t)$\in N^+$. All of the resources have been accessed by instance α until time t is denoted as the set rh(α,t), and rh(α,t) $\subseteq BS$.

Definition 3(QoS Metrics). Suppose that M transaction instances are submitted during a certain time, while finally N instances（denoted as 1, 2,…,n）committed successfully and K instances rolled-back because of deadline missing or deadlock, then some transaction QoS metrics are defined as following:

(1) Average Complete Time, ACT = ($\sum_{i=1}^{n}(et(i)-at(i))$) / N.

(2) Weighted Average Complete Time, WACT =

($\sum_{i=1}^{n}((et(i)-at(i))*priority(i,et(i)))$) / ($\sum_{i=1}^{n}priority(i,et(i))$).

(3) Missed Deadline Percentage, MDP = (K / M)*100%.

Definition 4(Blocker). If the transaction instance t1 is blocked because of waiting for the lock of a bean, while another instance t2 owns a lock of the bean conflict to the desired lock, we call instance t2 as the blocker of instance t1. A transaction instance may have several blockers and its blocker may have its own blockers, too. If a transaction instance is represented by a node, and the blocking relationship is represented by an arc, then we get a Directed Acyclic Graph (DAG). A blocker chain of a node is a path beginning from the node. As illustrated in figure 1, the red node serial <t2, t4, t3, t5> is a blocker chain of instance t2.

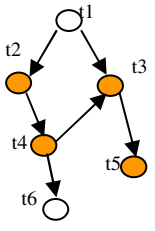


Fig. 1. Blocker Chain

3.2 Heuristic Priority Assignment

In this section we introduce a dynamic priority assignment algorithm. The algorithm can distinguish the key tasks from the trivial ones at runtime and the users can also assign the priority manually if they assure the value of the tasks.

The difficulty is determining the value of a task in the absence of explicit type information. Different type tasks access different type resources. This fact suggests that we can get some hints from the resource accessing history of task. In J2EE middleware, the Enterprise JavaBeans (EJB) specification has defined how to store some extra properties of beans in their descriptor files [16]. If we assign a weight value to each bean, we are able to figure out the value of tasks from its accessed resource list heuristically. Our HRS algorithm is based on this idea.

Definition 5(HRS). The priority of a transaction instance at time *t* is calculated by the following formula:

Priority(α ,t) = StaticPriority(α ,t) + AgePriority(α ,t) + ResourcePriority(α ,t)

StaticPriority is a natural number in a finite set, such as [0, 1000]. It is optional on transaction creating, with a default value 0. Being an initialization parameter, it can also be adjusted at runtime. For example, if a session bean *TransferMoney* is invoked by a transaction instance, we can ensure the type of the task at that time, and increase its *StaticPriority* in the code body of *TransferMoney*.

AgePriority is a linear function of time, which expresses the age of a transaction instance. AgePriority(α ,t) = k * (t – at(α)), *k* is the linear factor and $k \in N^+$. Many

researches [17,18] indicate that a priority-driven scheduling algorithm should consider two independent causes: *task-critical* and *task-urgency*. The factor k is a balance between the above two causes.

ResourcePriority(α,t) is defined as the sum of weight values of beans which have been accessed by instance α until time t. If a bean has been accessed more than one times, only one time is counted.

According to EJB specifications, every entity bean is associated with a deployment descriptor file. We extend the descriptors set by adding the item *BeanWeight* to it, which denotes the weight of the bean. The item is configured by the administrator manually at deployment time, and its default value is 0.

3.3 Scheduling Algorithm

We put forward the scheduling algorithm TMPBP based on a basic 2PL policy. In J2EE, each transaction instance is attached to a specific thread. TMPBP defines two variables in the thread context, *TimeoutTimes* and *LastPriority*, respectively represents the times of timed-out in history of the thread and its last priority. The waiting transaction instances are put into multi queues according to the *TimeoutTimes* value and the transaction manager schedules the queue with highest *TimeoutTimes* value first.

In TMPBP, each entity bean carries a property *priority* equals to the highest priority of all instances in the waiting queue. Attention should be paid that the *priority* of a bean is different to the *weight* of a bean. The weight is assigned during the deployment time statically, while the priority is generated and updated at runtime dynamically.

TMPBP does not employ any advanced approaches to avoid deadlocks. It just detects the possible cycle in a wait-for graph and aborts the instance with lowest priority. As mentioned in section 2, the approach is practical and more efficient.

The complete description of TMPBP is listed as following. Among the description, the attached thread of current transaction instance is denoted as *self*, and the current time is denoted as *now*:

Table 1. The description of TMPBP

```
on schedule(tx, bean):// instance tx would like to visit bean
  lookup the bean's lock table;
  if (tx is permitted to grant the lock)
  {  // go ahead
        if (bean∉ rh(tx, now))
     tx.ResourcePriority = tx.ResourcePriority + wt(bean);
    grant tx the lock;
  }
  else
  {  //blocked, waiting…
    if (tx.priority>bean.priority)
    {
       for all tx's blocker chains, do
       {  //update the blocker chain
         from the first node to the last node of the chain, do
```

```
            {
                if (node's blocker's priority<node's priority)
                  node's blocker's priority = node's priority;
            }
          }
        }
        put tx to corresponding waiting queue;
        waiting for bean's lock;
        if (a deadlock is detected)
            abort the instance with lowest priority in the cycle.
    }
```

on release lock(bean): //pick a tx to run

```
    select the queue Q with highest timeoutTimes;
    select highest priority transaction instance tx from the Q, if there are several
    instances with the same priority, select a random instance from them;
    if (bean∉ rh (tx, now))
        tx.ResourcePriority = tx.ResourcePriority + wt(bean);
    grant tx the lock;
    //refresh the bean's priority
    bean's priority = max value of instances' priority in bean's waiting queues.
```

on transaction create(initPriority):

```
    ResourcePriority = 0;
    if (has no timeoutTimes defined in Thread Context) //init the Thread Context
    {
        self.timeoutTimes = 0;
        self.lastPriority = 0;
    }
    StaticPriority = initPriority + self.lastPriority;
```

on transaction commit(tx):

```
    release tx's all locks;
    self.timeoutTimes = 0;
    self.lastPriority = 0;
```

on transaction rollback(tx):

```
    release tx's all locks;
    if ((reason == TIMEOUT) || (reason == DEADLOCK))
    {
        self.timeoutTimes = self.timeoutTimes + 1;
        self.lastPriority = priority(tx,now);
    }
    else
    {
        self.timeoutTimes = 0;
        self.lastPriority = 0;
    }
```

on calculate priority(α ,t):

```
    return  StaticPriority( α ,t) + k*( t-at( α )) + ResourcePriority( α ,t);
```

4 Algorithm Analysis

Subsequent paragraphs analyze some important properties of the TMPBP algorithm, including the time complexity and the characteristics of scheduling.

4.1 Time Complexity

As we can observe from table 1, most of the operations in TMPBP are linear, except the operations of searching lock table, updating blocker chains and detecting deadlocks in section "*on schedule (tx, bean)*".

Cost of the above operations is depending on how the relevant data structures are organized. Suppose we organize the lock table as a doubly linked list with Bean ID hash value as its list header, and the number of active transaction instances in system is *n*. In the worst case, all of the other instances are blockers of the current instance (for example, current instance requires a write lock, while the other n-1 instances hold the read lock) and the time complexity of the first two operations is O(n). As to detecting deadlocks, we adopt the data structure and algorithm presented in JBoss EJB container [11] and its time complexity is $O(n^2)$. Thus the overall time complexity of TMPBP is $O(n^2)$.

4.2 Characteristics of Scheduling

Property 1. There is no starvation in TMPBP.

Proof: first we prove that TMPBP does not produce starvation if we do not take timeout into account.

Suppose the finite set S {$StaticPriority_1$, $StaticPriority_2$,…, $StaticPriority_n$} contain all of the possible static priorities. Let α be any given transaction instance, there must exist a time *t* ($t > at(\alpha)$) where

$$AgePriority(t\text{-}at(a)) = Max(S) + \sum WT \quad (1)$$

Since AgePriority(t) is a linear function, as we can learn from the formula (1), if any transaction instance has a arrival time larger than *t*, its priority must be less than priority of α.

Suppose set P { α , p_1, p_2, … , p_n } include all of the active transaction instances at time *t*, it is a finite set with a size of n+1. Any possible instance which priority is larger than priority of α is in set P. When time is larger than *t*, if instance α is blocked in a waiting queue of any bean, in the worst case, it will get the lock at the $(n+1)^{th}$ round scheduling. So the instance α cannot be starving.

The above discussion does not consider the deadlocks. If a deadlock occurs, the instance with lowest priority will be aborted. If the victim is another instance instead of α, the above conclusion is still true. If the victim is α, the conclusion is identical with the following conclusion on timeout circumstance.

Now we take timeout into account. The instance α may time out before time *t* and become starved, i.e. the transaction is lunched again and again, but every time it has been timed out and is aborted before time *t*. (Of course the time-out value must be large enough to complete its work, otherwise it is impossible to commit successfully).

In TMPBP, we store the timeout history information in the context of client thread so that we can recognize the timed-out transaction instance when it is launched again. The instance inherits the final priority before its latest rollback. When the transaction is submitted again, its priority increases continuously as if there is no timeout. And we have proved that there is no starvation if there is no timeout. Thus the starvation is prevented.

Property 2. There is no priority inversion in TMPBP.

Proof: In TMPBP, each blocked transaction instance calculates its priority before entering the waiting queue. If the calculated result is larger than the priority of the bean, the transaction manager updates the priority of the bean and the priority of all of the blockers of current instance. After the updating of blocker tree, the priority sequence is guaranteed to be monotonically non-decreasing when we travel follow any blocker chain in the blocker tree; hence the priority inversion is prevented.

The priority of instances in waiting queue increases with time, but we do not update the blocker tree when we schedule in a future time. Now we prove that it does not affect the scheduling result and does not produce the priority inversion.

Suppose the transaction instance blocked is α .The StaticPriority and ResourcePriority of α are not changed since α has been blocked, and the only cause that changes its priority is AgePriority. But AgePriority is a linear function of time *t*, and during the same period of time, the increase produced by AgePriority to α and any instance in its blocker tree is identical. Hence the property sequence of blocker tree is still maintained to be monotonically non-decreasing when we schedule in a future time.

Property 3. The scheduling produced by TMPBP is serializable.

Proof: In TMPBP, a transaction instance does not release its locks until it ends. In other words, all of the locking operations are prior to any unlocking operations, i.e., the TMPBP is compliant to Two Phased-Locking (2PL) protocol, and more specifically, is compliant to a strict 2PL protocol. According to paper [19], the scheduling produced by a scheduler compliant to 2PL is serializable.

5 Experimental Evaluations

In this section, we investigate the performance of TMPBP algorithm and discuss the influence of factor *k* in HRS by some simulation experiments. The computer used in the experiments consisted of a Pentium 4 2.4G CPU and a 512M DDR RAM, with an operating system of Microsoft Windows 2000 Professional.

5.1 QoS Comparison

We simulated a concurrent execution with varied number of transaction instances. In each round, the first five transaction instances carry different static priorities while the rest carry zero. The first ten of thirty beans carry different weights, and the other twenty beans carry zero. Each transaction accessed 5 beans during its life cycle and each accessing consumed 0.5 seconds.

The experiment parameters are listed as following:

Table 2. Experiment Parameters List

Parameter	Value
Weights of Beans	(50,50,100,100,150,150,200,200,300,300,0….0)
StaticPriority of transactions	(200,200,300,300,300, 0…0)
Age Factor k	10
Time-out(seconds)	30

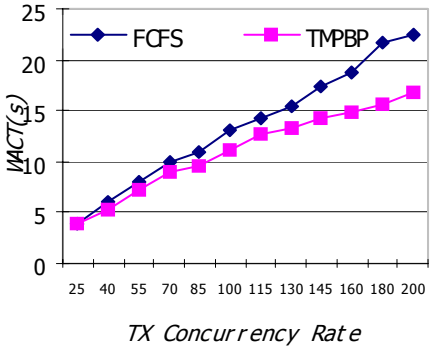


Fig. 2. WACT comparison on concurrency

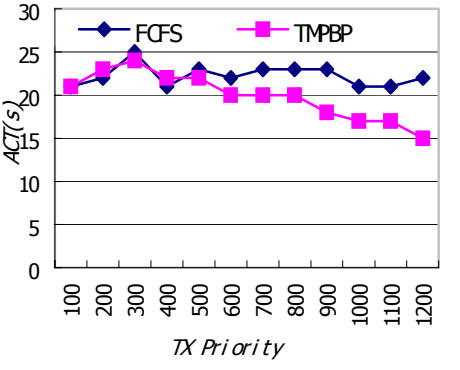


Fig. 3. ACT comparison on priority

Figure 2 shows the WACT comparison on different transaction concurrency rate. As we can observe from the figure, the difference of WACT under two scheduling algorithms increases with the increase of concurrency rate. When the server is under heavy load, the waiting queue is long and the transactions cost a large part time on waiting for locks. The TMPBP algorithm schedules the transaction instance with higher priority first, hence the waiting time of key tasks is decreased and as a result their WACT and MDP are reduced.

Figure 3 and figure 4 shows the advantage more clearly. Figure 3 gives the ACT comparison on different proprieties and figure 4 gives the MDP comparison on different priorities. Both experiments are under a concurrency of 200 transaction instances.

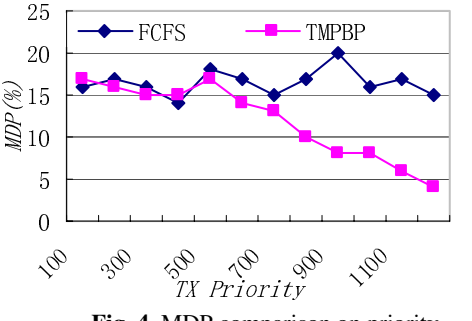


Fig. 4. MDP comparison on priority

From the above three figures, we can conclude that the TMPBP improves the QoS of higher-priority transaction instances remarkably, especially when the server is under the heavy load.

5.2 Factor Adjustment

We did not give the exact value of factor *k* in HRS algorithm until now. As mentioned in section 3.2, *k* is a balance between *task-critical* and *task-urgency*. In general, *k* should be adjusted according to the distribution of completion time, timeout values and the length of waiting queue.

We study the performance of TMPBP under two somewhat extreme values of *k*, 5 and 500. The other parameters are same to the ones used in the last experiment and the transaction concurrent rate is 200.

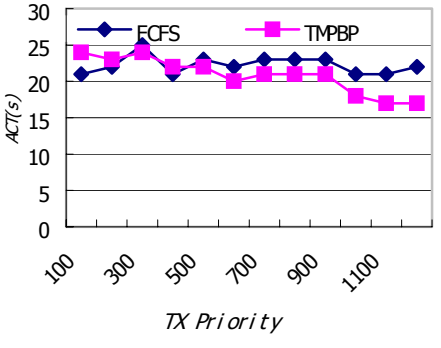


Fig. 5. ACT comparison on k=5

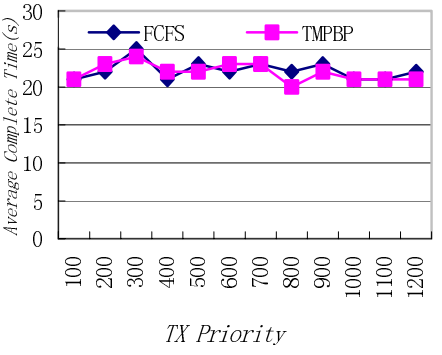


Fig. 6. ACT comparison on k=500

From the above two figures, we can draw the conclusion that there is a notable difference on ACT under different values of *k*. In figure 5, the performance of TMPBP is still better than the performance of FCFS, but is obviously worse than the performance of TMPBP in figure 3, where *k* is set to 10. While in figure 6, the performance of TMPBP is similar to the performance of common FCFS and it indicates that TMPBP lose the ability to recognize the key tasks.

We can explain the two figures from the essence of *k*. When k→0, HRS does not take the age of transaction into account. It is suitable for the situations where time-out value is large and the possibility of time-out is low. In such situations, the age of transaction is not sensitive. On the contrary, when k→∞, the value information figured from the resource accessing history of a transaction instance is submerged. The TMPBP algorithm degenerates into a common EDF algorithm and the key tasks lose their advantages. Furthermore, the time-out values are identical in the experiment, thus the performance of TMPBP is similar to the performance of FCFS in figure 6.

6 Conclusion

In this paper, we import a priority-driven scheduling algorithm to provide a better service for key tasks in J2EE. We face two problems: (1) How to recognize the key tasks dynamically in the absence of some necessary scheduling information. (2) How to produce a safe and efficient scheduling.

Our contributions are the solutions to the above two problems. First, we propose a new dynamic priority assignment algorithm HRS, which can recognize the key tasks at runtime heuristically. The idea underlying HRS is assigning a weight to each resource and figuring out the value of tasks through the resource accessing history of the transaction instance. Second, we introduce some new techniques to the traditional concurrency control policies according to the features of J2EE middleware, such as binding scheduling information with client thread, and present the scheduling algorithm TMPBP. It is proven to be safe and the experimental evaluations show that a considerable improvement on QoS of key tasks was achieved.

Acknowledgements. This paper was supported by the National High-Tech Research and Development Program of China (863 Program) under Grant No.2001AA113010, 2003AA115440, and the Major State Basic Research Development Program of China (973 Program) under Grant No.2002CB312005.

References

1. Sun Microsystems Inc., Java™ 2 Platform Enterprise Edition (J2EE) specification, v1.4, (2003)
2. Wanxia Xie, Shamkant B. Navathe, Sushil K. Prasad, Supporting QoS-Aware Transaction in system on a mobile device, Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), (2003)
3. Abbott R, Garcia-Molina H., Scheduling real-time transactions: A performance evaluation. ACM Transactions on Database System, 17:513-560, (1992)
4. Dertouzos ML, Mok AK. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. on Software Engineering, 15(12):1497-1506, (1989)
5. Liu C, Layland J, Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 20(1): 46-61, (1973)
6. Haritsa JR, Livny M, Carey MJ. Earliest deadline scheduling for real-time database systems. In: Proceedings of the 12th IEEE Real-Time Systems Symposium. Los Alamitos, CA: IEEE Computer Society Press, 232-243, (1991)
7. Liu YS, He XG, Tang CJ, Li L. Special Type Database Technology. Beijing: Science Press, (2000)
8. Jensen ED, Locke CD, Toduda H, A time-driven scheduling model for real-time operating systems. In: Proceedings of the IEEE Real-Time Systems Symposium. Washington, DC: IEEE Computer Society Press, 112-122, (1985)
9. Abbott R, Garcia-Molina H. Scheduling real-time transactions. ACM SIGMOD Record, 17(1):71-81, (1988)
10. JBoss Group, http://www.jboss.org
11. JBoss Group, JBoss Administration And Development, (2004)
12. L.Sha, R.Rajkumar, J.Plehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, 39:1175-1185, (1990)
13. Özgür Ulusoy, Geneva G. Belford, Real-Time Transaction Scheduling In Database Systems, Proceedings of ACM Computer Science Conference, (1992)
14. L.Sha, R.Rajkumar, J.Plehoczky., Concurrency Control for Distributed Real-Time Databases. ACM SIGMOD Record , 17:82-98, (1988)
15. P.A.Bernstain, V.Hadzalacos, N.Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesly, (1987)

16. Sun Microsystems Inc., Enterprise JavaBeans™ specification, v2.1, (2003)
17. Buttazzo G, Spuri M, Sensini F., Value vs. deadline scheduling in overload conditions. In: Proceedings of the 19th IEEE Real-Time Systems Symposium. Pisa: IEEE Computer Society Press, (1995)
18. Huang JD, Stankovic JA, Towesly D, Ramamritham K. Experimental evaluation of real-time transaction processing. In: Proceedings of the 10th Real-Time Systems Symposium. Santa Monica: IEEE Computer Society Press, (1989)
19. K.P.Eswaran, J.N.Gray, R.A.Lorie, I.L.Traiger, The notations of consistency and predicate locks in a database system, Comm.ACM 19(11):624-633, (1976)

Application Object Isolation in Cross-Platform Operating Environments

Stefan Paal[1], Reiner Kammüller[2], and Bernd Freisleben[3]

[1] Fraunhofer Institute for Media Communication,
Schloss Birlinghoven, D-53754 St. Augustin, Germany
stefan.paal@imk.fraunhofer.de
[2] Department of Electrical Engineering and Computer Science, University of Siegen,
Hölderlinstr. 3, D-57068 Siegen, Germany
kammueller@pd.et-inf.uni-siegen.de
[3] Department of Mathematics and Computer Science, University of Marburg,
Hans-Meerwein-Strasse, D-35032 Marburg, Germany
freisleb@informatik.uni-marburg.de

Abstract. Many efforts have been spent to overcome the single-application hosting limitation of the regular Java Virtual Machine (JVM). A common approach is the introduction of custom class loaders to separate *application classes* and to enable multi-application hosting. A yet unresolved problem is the separation of *application objects* once the object references have been exposed to other applications within the same and across different JVMs. In this paper, we present a novel approach towards Java object isolation in cross-platform operating environments. We introduce so called Java Object Spaces which control the communication among particularly encapsulated object groups and enable the custom separation of inbound object instances and outbound object references. The realization of the approach is described and its application for ad-hoc execution migration is illustrated.

1 Introduction

The Sun Java Virtual Machine (JVM) has been initially designed to host a single Java application only [1]. Consequently, many efforts have been spent to overcome this limitation, such as using custom class loaders [2]. They can be used to build a class loader hierarchy for sharing common application classes and separating incompatible class variants. This enables the concurrent hosting of multiple applications within the same JVM. Custom class loaders represent a common approach to isolate *application classes*. A yet unresolved problem is the isolation of *application objects* and the related object references once they have been exposed to other applications within the same JVM [3].

In a cross-platform operating environment with several JVMs, local application objects may refer to remote application objects and pass returned object references to other local objects, as shown in fig. 1. This makes it impossible to track each object reference linked to an object instance. In fact, there is no way to update all existing object references to a moved or replaced object instance. A remedy is the interception of object communication by isolating object instances and tracking the passed object references.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1046 – 1064, 2005.

Common object-oriented middleware approaches, such as Java RMI, rely on the stub/skeleton pattern to handle object communication with remote peers. On the one hand, they track the passed object references between local and remote JVMs to seamlessly create new stub/skeletons pairs. On the other hand, there is typically no support to track the passing of object references within the same JVM. In a multi-application hosting scenario, it is not possible to determine the object instances and object references which belong to a running application. This makes it difficult to support online updates and execution migration without affecting concurrently hosted applications.

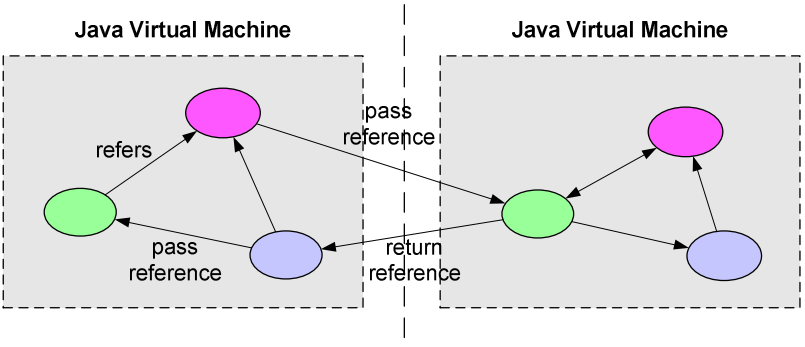


Fig. 1. Unmanageable Tracking of Object References and Object Instances

In this paper, we present a novel approach towards Java application object isolation in cross-platform operating environments. We introduce so called *Java Object Spaces* [4] which encapsulate specific groups of application objects. They enable the separation of outbound object references and inbound object instances within the same and across various JVMs. The realization of the proposed approach is based on *Java Dynamic Proxies* [5] and a particular tracking of serialized method calls by using custom object streams. We show how this approach can be used to support multi-application hosting, seamless implementation updates and transparent object migration in a cross-platform operating environment.

The paper is organized as follows. In section 2, we define application object isolation, discuss its objectives and challenges and examine related work. In section 3, we introduce Java Object Spaces to support application object isolation in a legacy JVM. The conceptual approach and its realization using Java Dynamic Proxies are described. In section 4, we demonstrate the application of the approach for ad hoc execution migration in a cross-platform operating environment. Finally, section 5 concludes the paper and outlines areas for future work.

2 Java Application Object Isolation

In this section, we introduce the definition of Java application object isolation used in this paper and present the related objectives. Furthermore, the challenges imposed by the use of multiple JVMs in a cross-platform operating environment are presented and related work is discussed.

2.1 Definitions

In our understanding, a Java application object is an entity that can be accessed via a regular Java object reference. It is typically referenced by other application objects in a one-way direction and is not removed from the JVM as long as a corresponding object reference is in use. While some application objects may be created from classes loaded by the same class loader, others are associated with different class loaders. In addition, application objects may have to be virtually grouped by their origin or use, e.g. as part of a component or an application. From this point of view, we distinguish various types of application object communication in a cross-platform operating environment, as illustrated in fig. 2.

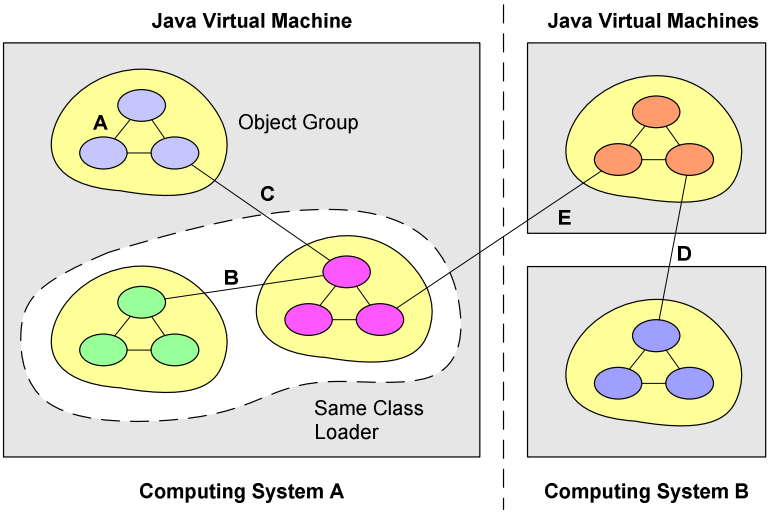


Fig. 2. Types of Application Object Communication

There are objects which are virtually grouped and tightly coupled. They use the same class loader, and object communication will not leave the object group (A). Another scenario is formed by object groups which belong to different origins but still use the same class loader. Object communication may pass group boundaries, e.g. when accessing objects of an application plugin (B). In the next scenario, object communication involves objects associated with classes that have been loaded by different class loaders, e.g. calling methods on a shared software component (C). This could cause problems with colliding or missing classes depending on the class loader hierarchy and thus requires particular treatment. In case application objects reside in different JVMs, object communication has to cross process boundaries and use object-oriented middleware approaches, such as RMI or CORBA (D). Finally, application objects may be also hosted in JVMs started on different machines. Apart from particular communication issues, such as establishing a suitable network link to the remote host (E), the employed middleware approach has to dynamically resolve the location of an object if objects migrate to another host while they are in use, e.g. by means of an object registry.

In summary, there are various types of application object communication which have to be addressed when considering object updates, replacements or movements without being able to modify related object references. A remedy is the transparent interception of object communication which allows the use of regular object references on the caller side but enables the dynamic relinking of objects by the callee. As a result, the overall goal of application object isolation is the custom separation of object references from referenced object instances.

2.2 Objectives

There are different objectives of application object isolation. In this paper, we focus on the seamless handling of object references and communication among replaced and moved object instances within a cross-platform operating environment.

Application Object Groups. A basic issue of application object isolation is setting up which object groups should be treated as atomic, e.g. if either all or none of the objects of a shared library may have to be updated. It eases the handling of large numbers of objects and helps to identify groups of objects which are not related otherwise, e.g. by using the same class loader. In addition, there is no need to isolate these objects from each other and to intercept the communication among them.

Online Implementation Update. Java supports the dynamic loading of classes, and custom class loaders are often used to virtually replace already loaded classes. In fact, objects may be created using the new class loader and classes. However, it is not possible to update the implementation of an existing object once it has been created. A particular objective of application object isolation is the support of online component updates which replace the implementation of already created objects.

Transparent Object Migration. In a cross-platform operating environment, objects may be moved from one host to another one while they are in use. Local and remote object references are not implicitly updated, and it is also not feasible to notify each object reference about the movement of referenced objects. A corresponding objective of application object isolation is the support of transparent object migration without invalidating fixed and moving object references.

Seamless Object Access. While objects in the same object group may refer to each other using regular object references, the communication with objects located in different object groups requires particular efforts to handle application object isolation. Nevertheless, new objects should be seamlessly integrated into their object group, and corresponding object references are supposed to be transparently transformed and exchanged across various object groups without application intervention.

2.3 Challenges

There are several challenges for application object isolation in a cross-platform operating environment, as described below.

On-Demand Operation. Application objects may be dynamically replaced or moved by request. A particular challenge is the on-demand object isolation without actually knowing the involved object groups in advance. The encapsulated objects could

become unavailable at any time and related object references may be unexpectedly blocked until the isolation is released. In turn, object groups and contained object references must be prepared to be serialized in case they are moved to another host.

Custom Class Loaders. They are used to enable the concurrent loading of multiple class variants. In this context, an object can use only classes which can be loaded via its class loader or one of the parent class loaders, e.g. for casting an object reference. This is typically achieved by configuring shared class loaders. However, some object groups may have to use different class loaders. The challenge is how to pass objects of compatible variants of the same class without sharing a custom class loader.

Distributed Object Communication. The possible separation of object references and referenced objects into different JVMs results in distributed object communication. The use of common object-oriented middleware approaches introduces particular application objects, such as stubs and skeletons, to mask the distributed object communication. This raises the challenge of how these objects can be isolated in the same way as regular application objects, e.g. for updating a stub implementation.

Legacy Runtime Environment. While a proprietary JVM could realize object isolation without application modification, in practice it is not feasible to provide and support suitable JVM implementations for different types of operating systems and hardware platforms. The same is valid for applying programming tools which create custom Java byte code. A particular challenge is the implementation of object isolation for use in a legacy runtime environment and with regular Java byte code.

2.4 Related Work

The overall goal of object isolation is the separation of an object reference from the related object instance. This leads to the custom handling of object communication which has already been addressed by many Java object middleware approaches. In the following review, we consider related work with respect to the objectives and challenges of Java application object isolation in a cross-platform operating environment.

The first approach considered is legacy *Remote Method Invocation (RMI)*. It is the proposed solution of Sun for binding distributed Java objects and calling methods remotely. There is basic support to relink RMI object references on the fly, although the custom separation of an RMI object reference from its corresponding remote object is formally not supported. In addition, the grouping of tightly coupled objects and the custom interception of object communication from and to this object group is not supported. Another drawback of RMI is the explicit compilation of stubs/skeletons for each object implementation. This is not feasible for on-demand operation. There are various approaches which rely on RMI and extend it with custom features [6]. As an example, TRMI [7] uses Java dynamic proxies to wrap the RMI communication and to enable dynamic remote object linking without using the `rmic` compiler. The same is valid for RMI in Sun J2SE 5. However, the RMI-based approaches do not address multi-application hosting. There is no separate handling of different object instances and custom tracking of object references within the same JVM.

A well-known middleware approach is the *Common Object Request Broker Architecture (CORBA)* [8]. In contrast to RMI, it readily supports the dynamic linking of

yet unknown remote objects and the custom interception of object communication via interceptors. It is part of the Sun J2SE and does not have to be installed manually. From this point of view, CORBA represents a suitable approach to realize application object isolation on top of it. On the other hand, the application of CORBA requires some modification of legacy Java application code, such as using the narrowing approach to cast a remote object reference. This may complicate the use in a cross-platform operating environment with legacy application code migrating from one host to another. The update of object implementations leads to a similar problem that CORBA stubs and skeletons have to be exchanged whenever the target object implementation is modified.

There are many custom approaches which are based on the introduction of custom class loaders and the modification of how classes are selected, loaded and arranged in the JVM. Popular examples include servlet engines, such as *Jakarta Tomcat* [9], and J2EE application servers, such as *JBOSS* [10] which benefit from the ability to load different class variants at the same time. Custom class loaders may also be used to update already loaded classes and use new variants, e.g. for reloading and restarting a Java servlet. A common drawback of custom class loaders is their focus on class separation. Another one is the introduction of a specific programming model like *Enterprise Java Beans (EJB)* or *OSGi*. It is not possible to update the implementation of already linked objects. A custom class loader is not aware which object belongs to a certain application, and there is no way to determine object groups of an application, e.g. scheduled for migration.

Approaches based on a custom Java Virtual Machine, such as the *Multi-Tasking Virtual Machine (MVM)* [1], *NOMADS* [11] and *Camel* [12], may offer particular features, e.g. a multi-tasking runtime environment, code sharing and a mobile application framework. On the other hand, they require additional effort to install the JVM and suffer from the lack of portability in a heterogeneous cross-platform operating environment. In a similar way, code rewriting approaches, such as *JOrchestra* [13] and *J-Seal2* [14], are able to incorporate extra features like automatic application partioning, advanced resource control and protection domains. Due to byte code modification, they inherit the problem that this cannot be performed on-demand during runtime, e.g. when passing object references among uncertain and dynamically loaded components. Furthermore, they relay on a different programming model which hinders the dynamic hosting of unmodified legacy Java applications.

In summary, there are various approaches which address and support application object isolation to some extent. Native approaches, such as RMI, support the seamless integration of the approach in a legacy runtime environment but lack support for customizing the object communication. Standard object middleware approaches, such as CORBA, can be used to dynamically link remote object instances and are able to separate the tight coupling of object reference and object instance. However, they fail for tracking the object references within the same JVM. Custom approaches, such as the MVM and J-Seal2, provide particular features but often rely on a proprietary JVM or a different programming model. They cannot be used easily in a cross-platform operating environment with unknown applications. Consequently, an approach towards legacy application object isolation that supports seamless integration, transparent migration, online update and virtual grouping of application objects on-demand is still missing.

3 Java Object Spaces

Considering the particular challenges of a cross-platform operating environment, we propose a new way to address Java application object isolation. We illustrate the conceptual approach of so called *Java Object Spaces* and describe its realization based on legacy Java features. In addition, performance issues are discussed.

3.1 Conceptual Approach

The basic element of application object isolation is the virtual object group; the overall goal is the interception of object communication from and to the contained objects. Based on this idea, we introduce so called *Java Object Spaces* which represent a custom group of application objects and control the object communication via dynamic stubs and skeletons, as shown in fig. 3.

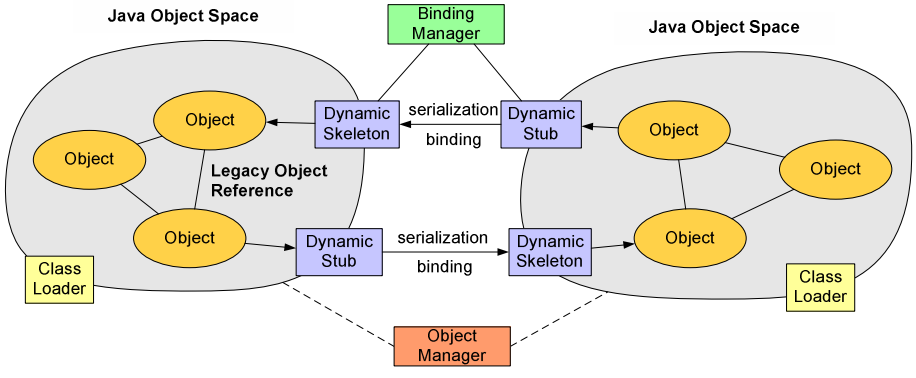


Fig. 3. Java Object Spaces

The objects within the same object space can refer to each other using legacy Java object references. Objects from another object space are not directly referenced but bound by a stub/skeleton pair which is dynamically created whenever an object reference passes the boundary of an object space. In this context, a stub and a skeleton form an invariant *object binding* which may be intercepted for object isolation. An object manager is used to create an object space and to associate objects and a class loader with it. An object may be referenced by any object from the same or a different object space; in the same or another JVM.

3.2 Realization

The presented approach is part of ongoing work towards the development of an autonomic cross-platform operating environment [15]. In particular, we use features of our previous work towards *method streaming* [16] for the transparent handling of network communication among distributed objects.

Object Communication. A crucial problem we have already identified in our previous work on method streaming is the transparent binding of local and remote objects independent of the underlying middleware approach. We have introduced so called *method streams* which can be used to call methods on local and remote objects across different JVMs and distributed hosts. In effect, method streams establish an abstract Java communication layer which can be used to dynamically switch from one middleware approach to another, e.g. from CORBA to RMI, and to relink migrated objects transparently [16]. We also benefit from its ability to dynamically resolve the current location of the object in a cross-platform operating environment by using an invariant binding identifier. An object reference to a remote object actually refers to a *Java Dynamic Proxy* which masks itself to implement the interfaces of the target object. It delegates all method calls to a single method `invoke` of a dynamic stub implementing the Java standard `InvocationHandler` interface, shown in fig. 4.

```
public Object invoke(Object proxy, Method m, Object args[])
                                                  throws Throwable {
    String[] arr = new String[args.length];
    for (int i=0;i<args.length;i++)
       arr[i] = m.getParameterTypes()[i].getName();

    IRequest req = new CRequest(m.getName(),bindingId,arr,args);
    IResponse resp = binding.invoke(req);
    if (resp instanceof Exception)
        throw (Throwable) resp.getReturnObject();

    return resp.getReturnObject(); }
```

Fig. 4. Separating the Method Call from Class Instances on the Caller Side

The argument `proxy` is passed by the delegating Java dynamic proxy but it is not used in our method. Instead, the argument types of the method call are extracted and the names of the related classes are encapsulated in a request object `req` along with the name of the method, an identifying `bindingId` of the target object and the argument values. The dynamic stub does not contain a reference to the target object but the `bindingId` is used to dynamically establish a communication link and pass the method call to the right business logic object. When the method call has been performed, the return value is extracted from the response object `resp` and passed back to the caller. On the callee side, a dynamic skeleton receives the request object and performs the method call on the linked business logic object `blo`, as shown in fig. 5.

```
public IResponse invoke(IRequest req) {

    int nArgs = req.getParameterTypes().length;
    Class[] arrCls = new Class[nArgs];
    for (int i = 0; i < nArgs; i++) {
      String tp = req.getParameterTypes()[i];
      Class cl = blo.getClass().getClassLoader().loadClass(tp);
      arrCls[i] = cl; }

    m = blo.getClass().getMethod(req.getMethodName(),arrCls);

    return new CResponse(m.invoke(blo, req.getParameters())); }
```

Fig. 5. Getting Class Instances for a Method Call on the Callee Side

First, the class names of the method arguments are extracted and the corresponding classes are loaded using the class loader of the callee object space. Next, the corresponding method `m` of the business logic object `blo` is determined by using Java reflection, and the method call is issued. Finally, the return value is encapsulated in a response object and passed back to the caller.

While method streams can be easily used to connect remote objects in a cross-platform operating environment, there is a particular problem when passing method calls among objects instantiated by different class loaders in the same JVM. A method stream can be bound to only one class loader and it is not aware of object spaces using different class loaders. This imposes each object communication to be serialized even if the caller and the callee use compatible class loaders. This decreases the overall performance of the approach. A further development addresses this problem and is used in the presented approach to connect objects across various object spaces while using the appropriate class loaders on either sides, as illustrated below.

Binding Objects. When a method call leaves an object space and enters another one, it is passed down a method stream and the corresponding request and response objects are transparently serialized. At this point, we add particular binding streams which take care of different class loaders of the involved object spaces, as shown in fig. 6.

```
public class CBindingOutputStream extends ObjectOutputStream {

  protected Object replaceObject(Object obj)throws IOException {

     if (obj instanceof Serializable)
        // do nothing;
     else if (obj != null)   {
        CBindingId bindingId = bdgMgr.bind(space, obj);
        obj = bindingId;   }
     return super.replaceObject(obj); }
```

Fig. 6. Binding Output Stream

The legacy Java class `ObjectOutputStream` is used to derive a custom binding output stream which processes the serialized request objects passed from the caller to the callee. In doing so, the overridden method `replaceObject` is called for each object in the stream. While a legacy object output stream would throw an exception if a non-serializable object is found, we take a non-serializable object out of the stream, create a dynamic skeleton and bind it to the object space of the method stream. Then, the globally unique binding id of the skeleton is put into the output stream in place of the actual object. In effect, all leaving object references to non-serializable objects are encapsulated with dynamic skeletons and the related binding ids are passed to the callee. There, the binding ids are replaced with method streams to the remote objects left on the caller side. The corresponding binding input stream is shown in fig. 7.

Similar to the binding output stream above, the overridden method `resolveObject` of the binding input stream is called for each deserialized object. The corresponding Java class is evaluated to identify the binding ids which actually represent remote object references. Next, the binding manager is used to connect the related object. We want to point out that the binding manager evaluates the binding id and the

```
public class CBindingInputStream extends ObjectInputStream {

  protected Object resolveObject(Object obj) throws IOException{

     if (obj instanceof CBindingId) {
       obj = bdgMgr.connect(((CBindingId)obj), space); }
     return super.resolveObject(obj); }

  protected Class resolveClass(ObjectStreamClass desc) throws
                          IOException, ClassNotFoundException {
     return space.getClassLoader().loadClass(desc.getName());}}
```

Fig. 7. Binding Input Stream

object space where the new object reference should be created. If the target object resides in the same object space, the binding manager returns an object reference which directly refers to the target object, and no dynamic stub is introduced. In case of different object spaces, a dynamic stub is introduced; this is appropriately wrapped by a Java dynamic proxy. If the object is located in a different JVM or on another host, a method stream is established, and the dynamic stub is linked to the method streams. Subsequent method calls will be serialized using the presented binding input and output streams while they are passed through the method streams.

In addition, the method `resolveClass` is overridden to modify the class loading required to appropriately deserialize received objects. The class loader is taken from the object space that contains the callee object. As a result, all serialized objects and dynamic stubs of an object space are created by the same class loader. This is important for seamlessly casting object references, updating object implementations and serializing application object spaces, as illustrated below.

Transparent Serialization. A challenging objective of application object isolation is the serialization and deserialization of an entire object space. The Java Runtime Environment already offers an easy-to-use serialization approach which is enabled by the tagging interface `Serializable`. Each object can be decorated with this interface without imposing the implementation of any particular method. It is actually a marker for the Java compiler to create specific methods which are called by the JVM when this object is serialized. Besides serializing primitive attributes, the built-in serialization approach tracks each object reference and automatically serializes the linked objects (persistence-by-reachability). Though an object reference to a Java dynamic proxy can be generally used in place of the regular object reference, the Java serialization approach still differs between a regular object and a proxy object. Instead of transparently passing the serialization calls to the linked object, Java attempts to serialize the dynamic stub.

While this approach eases the transparent serialization of regular Java objects, it may cause problems for particular objects**,** such as dynamic stubs. Because of that, Java allows to add custom methods `writeObject` and `readObject` to the object implementation which are seamlessly called instead of the compiler-generated serialization methods. We exploit this particular feature and customize the serialization of the stub by introducing a custom method `writeObject`, as shown in fig. 8.

```
public class CDynamicStub implements InvocationHandler,
                                          Serializable   {
   IObjectSpace space;      // the containing object space
   CBindingId bindingId;    // unique id of the remote object
   String[] szInterfaces;   // implemented interfaces
   IBinding binding;        // the next binding in the stream

   private void writeObject(ObjectOutputStream out) throws
                                            IOException {
      out.writeObject(bindingId);
      out.writeObject(szInterfaces); }
```

Fig. 8. Serializing a Dynamic Stub

The method `writeObject` writes the unique binding id and the names of the remote interfaces to the stream. They are used to recreate and relink the stub to the method stream. For this purpose, a method `readObject` is introduced which is called whenever a dynamic stub is deserialized, as shown in fig. 9.

```
   private void readObject(ObjectInputStream in) throws
                     IOException, ClassNotFoundException {

     bindingId = (CBindingId) in.readObject();
     szInterfaces = (String[]) in.readObject();

     space = ((IObjectSpaceStream)in).getObjectSpace();
     binding = bdgMgr.connect(bindingId,space,szInterfaces); }
```

Fig. 9. Deserializing a Dynamic Stub

A custom object input stream `IObjectSpaceStream` is used to read in the serialized objects. The basic reason is the association with the target object space wherein the objects and stubs should be deserialized. On the one hand, all regular objects are reinstantiated using the class loader of the object space and the standard Java deserialization approach. On the other hand, the custom method `readObject` is invoked whenever a dynamic stub is about to be deserialized. There, the binding id and the interfaces of the remote object are read from the stream. Next, the stub is connected to the remote object by calling `connect` on the binding manager. In addition to the binding id and the names of the remote interfaces, a reference to the target object space is passed whose class loader should be used to create the Java dynamic proxy. This ensures that the returned object references can be seamlessly casted and passed among objects within the object space wherein this stub resides.

3.3 Use

After having illustrated the realization, we describe the use of the presented approach in various application scenarios.

Object Binding. The central element of the presented approach is the application object space which manages the object communication from and to contained objects. Consequently, an object must be associated with an object space before it can be managed, as shown in fig. 10.

```
IService aService = ...
ClassLoader cl1 = ...
IObjectSpace space1 = spaceManager.createSpace(cl1);
CBindingId bindingId = bdgMgr.bind(space1, aService);
```

Fig. 10. Object Binding

An object space manager is used to create a new object space `space1` and to initialize it with the class loader `cl1` to be used later by the object space. The binding manager is requested to bind the object `aService` to the object space and returns the unique `bindingId` appropriate for connections to the object, as shown below.

Object Connection. While object references passed back from method calls are transparently connected to corresponding objects via the presented binding output stream, the initial reference to an object can be retrieved using the related binding id, as shown in fig. 11.

```
CBindingId bindingId = ...
CObjectSpaceId spaceId2 = ...
IService aService = null;
IObjectSpace space2 = spaceManager.openSpace(spaceId2);
   aService = (IService) bdgMgr.connect(space2, bindingId);
spaceManager.closeSpace(space2);
```

Fig. 11. Object Connection

Similar to the binding of an object to a specific object space, the binding manager must know the object space in which a new object reference should be located. Assuming that the object space has been already created, the object space manager is used to get the related object reference `space2` which is then passed to the method `connect` of the binding manager. The location of the object is resolved by means of the method stream approach, and a dynamic stub is created which is wrapped by a Java dynamic proxy. The caller may cast down the reference without particular action and use it like any other reference located in its object space.

Object Passivation. Once the object is bound to an object space or an object reference is connected to an object in another object space, the further handling of the objects is transparent to the application developer. Binding and connecting of objects and object references is performed in the background when the method calls pass the binding input and output stream. A different application scenario is the passivation of an object space, as shown in fig. 12.

```
ObjectOutputStream os = ...
CObjectSpaceId spaceId3 = ...
IObjectSpace space3 = spaceManager.openSpace(spaceId3);
   space3.lock();
   space3.save(os);
spaceManager.deleteSpace(space3);
```

Fig. 12. Object Passivation

The object space is locked and written to an object output stream. The method **`save`** writes all contained objects to the output stream by using legacy Java serialization. This implicitly calls the custom serialization methods of the stubs and skeletons which disconnect remote objects and unregister local objects. At the end, the object space is deleted and removed from the object space manager. It should be pointed out that no remote object reference has been notified or has become invalid. As long as no method call is issued on a passivated object, they are not aware of the passivation.

Object Migration. A passivated object space can be easily transferred from one JVM to another. The use of method streams moreover enables the seamless relinking of object references from and to objects migrated across distributed platforms. Similar to the object binding scenario, an object space is created and associated with a class loader, as shown in fig. 13. The method `load` reads the serialized objects of the passivated object space and binds them to the current object space. Along with that, serialized stubs are reconnected with their remote counterpart via a method stream, and skeletons are bound to the new object space.

```
ClassLoader cl2 = ...
ObjectInputStream is = ...
IObjectSpace space4 = spaceManager.createSpace(cl2);
   space4.load(is);
   space4.unlock();
spaceManager.closeSpace(space4);
```

Fig. 13. Object Migration

It should be noted that the required classes of the serialized objects are loaded by a new class loader, as described in the realization. This also enables the use of compatible class variants as long as they are able to read in the serialized data. Instead of migrating objects to another host, object passivation may be also used to update the object implementation without invalidating existing object references.

3.4 Performance

An important issue is the performance of our realization in practice. We have carried out various tests following the basic communication scenarios discussed in section 2, namely a native method call (A), a proxy method call using the same class loader (B), a serialized method call using different class loaders (C), a local method call across different JVMs (D) and a remote method call across different hosts (E). The test program performs various method calls to examine the performance for passing different types of arguments, as presented in fig. 14.

```
public String echo (String val) {return val.toUpperCase();}
public long echo (long val) { return val + 1; }
public long[] echo (long[] source) { return source; }
public Object echo() { return this; }
```

Fig. 14. Test Program

Each method is called 10000 times and the elapsed time in milliseconds is shown in fig. 15.

	A	B	C	D	E
`String echo(String)`	20	40	4186	4436	4722
`long echo(long)`	10	60	5658	5879	6103
`long[] echo(long[1000])`	10	30	4026	4176	4340
`long[] echo(long[1000000])`	10	10	6499	6670	6832
`Object echo()`	10	20	4119	4947	5328

Fig. 15. Performance Results. Native Method Call (A), Proxy Method Call (B), Serialized Method Call (C), Local Method Call (D), Remote Method Call (E).

The performance results exhibit a large difference between method calls that are passed to objects with and without serialization. The values in columns A and B are comparable and indicate that the use of Java dynamic proxies slows down the method call somewhat. The other group, consisting of the columns C, D and E, uses object serialization for various reasons and are in the same range although the method call in E had to cross the network. Realizing that the values in C, D, and E differ less among each other than in A and B shows the great impact of the serialization which almost covers the different treatment of argument types. The size of the passed objects only shows up when the difference is relatively high, such as between row `long[1000]` and `long[1000000]`. Similar to serialized arguments, passing object references in the same JVM is quite fast for A and B but slow when it is bound, transferred and connected as in C, D and E. In contrast to local objects, the performance in E may further decrease depending on the network link used to connect the remote object.

3.5 Discussion

Our Java realization addresses the objectives of application object isolation with respect to the challenges of a cross-platform operating environment. It enables the separation of object reference and object instance while the object communication is customized to support appropriate serialization within the same JVM and across different hosts as well. From this point of view, it supports multi-application hosting, object implementation update and object migration. The application developer benefits from the transparent handling and easy-to-use application. He or she does not have to deal with various types of object communication, different object spaces and class loaders. An autonomic application framework is able to transparently passivate object spaces while the contained objects are in use. Subsequently, the serialized object space can be moved to another host or be reloaded using an updated object implementation. Moreover, the class loaders associated with the object spaces do not have to be chained in a hierarchy. They can be organized in different ways as long as they are able to read the serialized method calls passed to an object within the object space. This particularly supports the independent and asynchronous class loader configuration needed for on-demand application hosting.

The implementation is limited to Java objects which can either be serialized or declare an interface containing the methods a caller wants to use. The latter is due to the inherent use of Java dynamic proxies which can be casted to interfaces only. The separation of object reference and object instance as well as the creation of object duplicates during the serialization may cause problems concerning synchronization and object identity. There are also object references which do not refer to an object instance but to a class instance, e.g. for calling a static method. They cannot be encapsulated in an object space and there is no way to intercept the access to a class instance.

An outcome of the performance evaluation is that the presented approach is not well suited as a common object communication approach. There is a great difference between native and serialized method calls. In practice, the applicability of the realization will mainly depend on the business logic and the current application scenario. In case of a distributed application scenario, e.g. a cross-platform operating environment, the effort to serialize the arguments will be less important than in a single-platform scenario. In addition, the presented approach does not hinder objects in the same object space to communicate via native object references and avoid the serialization. Finally, we want to point out that the performance is only affected for object communication crossing different object spaces and class loaders. The realization automatically determines if compatible class loaders are used on the caller and callee side. Method calls are then passed directly and are not particularly serialized.

4 Application of the Approach

In this section, we describe the application of the approach for ad hoc migration in a cross-platform operating environment. The idea of execution migration is based on execution units that can be moved within a cross-platform operating environment and across various computing devices. We have implemented this approach in Java with so called *Java Execution Units (JEU)* that wrap the concerns of execution migration, namely resource binding, object serialization, code fragments and execution state in a self-managing cross-platform operating environment, as shown in fig. 16.

A key idea is the grouping of application objects into Java object spaces and the ability to isolate objects before they are migrated. A migration manager is introduced which controls migration issues of the execution units and uses the presented approach to control object interconnections among execution units. In this context, a Java execution unit does not simply reveal its objects to a caller but uses the binding manager of Java object spaces to create dynamic skeletons each time a reference to a contained object is requested. In turn, the caller receives a dynamic stub from the binding manager suitable to connect the formerly created skeleton as described above. In contrast to a binding manager, the resource manager is responsible to allocate local resources of the current host which cannot be migrated and have to be resolved on the target host when the objects are deserialized, such as access to environment settings or a graphical desktop system.

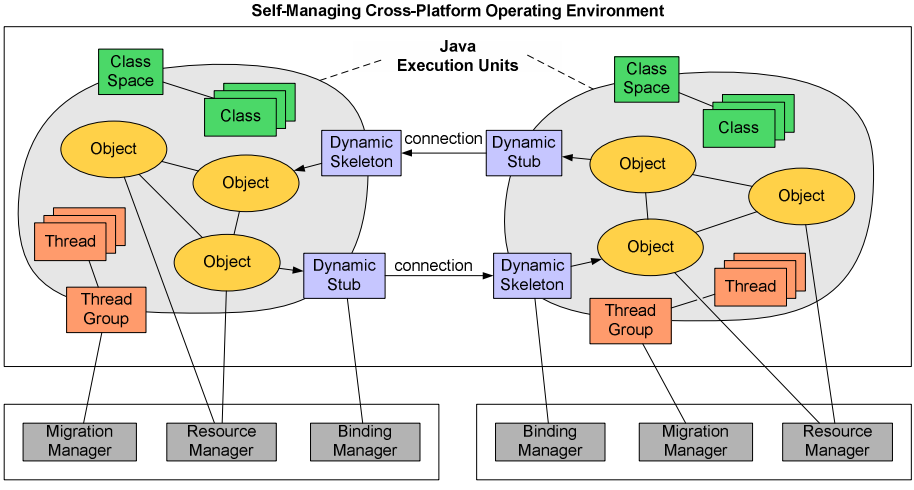


Fig. 16. Java Execution Migration in a Self-Managing Cross-Platform Operating Environment

The code handling of a Java execution unit is based on *Java Class Collections* (JCC) and *Java Class Spaces* as proposed in [17]. The basic idea is to separate the configuration where to actually find a Java class from the specification which classes are needed to run an application. Distributed code repositories are introduced which can be remotely queried for required classes, and a platform administrator is no longer specifying a static CLASSPATH but the locations of the remote repositories [17]. An application developer provides a class space configuration for an application which is then associated by the binding manager with the Java object space containing the applications objects. Next, each serialized argument passed to the Java object space is deserialized by loading the specific classes via the associated class loader of the object space. The same is valid for casting the dynamic stubs to interfaces available in the current object space. In case an interface of the deserialized object reference could not be loaded, the resulting Java dynamic proxy will at least provide limited access to the remote object via the available interfaces. As a result, migrating objects are always deserialized with the appropriate classes provided by the target object space.

A Java Virtual Machine currently supports only the execution of multiple threads which all reside in the same address space. The execution state of a Java execution unit is made up of all local variables created within a method execution, thread locals bound to a thread and the hierarchy of thread groups and threads. We associate a thread group to each Java object space which binds subsequent thread objects to the object space. They are serialized in the same way as data objects when the object space is moved from one host to another. However, before the application objects are isolated, the migration manager checks all skeletons if there are external threads currently processing methods of inbound objects and all stubs whether internal threads have called methods from outbound objects. It then notifies the external threads to

leave the object space and internal threads to return to the object space. As a result, the presented approach is also used to determine threads which have to be notified before the migration is actually performed.

5 Conclusions

In this paper, we have argued that Java application object isolation is a basic requirement to support multi-application hosting, seamless implementation updates and transparent object migration. We have discussed various types of object communication with isolated objects and have deduced the overall goal of application object isolation as the custom separation of object reference and object instance. An approach towards application object isolation in a cross-platform operating environment has been presented which is based on the idea of Java object spaces. It heavily depends on the use of Java dynamic proxies and customized object input and output streams. It transparently adjusts communication among objects located in different object spaces and supports the custom separation of object reference and object instance.

The realization of the approach and the use of Java object spaces in a legacy Java runtime environment have been described. The object spaces and the associated class loaders do not have to be arranged in a particular way, e.g. sharing a common class loader for passing method parameters or casting object references. Instead, the presented approach enables seamless access to any object instance as long as the passed arguments of a method call can be properly serialized and deserialized. Performance results have demonstrated the feasibility of the approach for use with distributed object spaces in a cross-platform operating environment. Finally, we have demonstrated the application of the approach for supporting ad hoc migration.

There are various areas for future research. The hosting of different applications in the same JVM raises particular security concerns. We plan to investigate how to introduce custom access control for Java object spaces. Another topic is the handling of particular migration issues like locking an object space and the synchronized migration of multiple object spaces. The presented approach may be also used to transform a running program and change its operation by reloading different classes. We will examine the employment of Java class collections and Java class spaces which allow determining compatible classes by custom properties, such as vendor and release version. Finally, we are currently working on an *Internet Application Workbench* which represents a graphical desktop interface for nomadic computing applications. The presented approach is supposed to migrate running applications from one workstation to another while the user is on the move. We are currently revising the implementation of the approach to include it into our *Crossware Development Kit (XDK)*. Further information and demos of ongoing work can be obtained from [15].

Acknowledgements

The presented approach is based on a cross-platform operating environment which has been evaluated and used in various projects, such as CAT [18], AWAKE [19] and

CROSSWARE [20]. They are funded by the German Federal Ministry for Education and Research and conducted by the research group MARS of the Fraunhofer Institute for Media Communication, Sankt Augustin in cooperation with the University of Siegen and the University of Marburg, Germany. Special thanks go to Monika Fleischmann, Wolfgang Strauss, and Jasminko Novak.

References

1. Czajkowski, G. Daynes, L. Multitasking without Compromise: A Virtual Machine Evolution. Proc. of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA). ACM 2001. pp. 125-138.
2. Liang, S., Bracha, G. Dynamic Class Loading in The Java Virtual Machine. Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 1998). ACM 1998, pp. 36-44.
3. Czajkowski, G. Application Isolation in the Java Virtual Machine. Proc. of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2000). ACM 2000. pp. 354-366.
4. Paal, S. Kammüller, R., Freisleben, B. Java Class Separation for Multi-Application Hosting, Proc. of the 3rd Intl. Conference on Internet Computing (IC 2002), CSREA Press, 2002. pp. 259-266.
5. Venners, B. Inside The Java 2 Virtual Machine. McGraw-Hill 1999.
6. Avvenuti, M., Vecchio, A. Embedding Remote Object Mobility in Java RMI. Proc. of the 8th Intl. Workshop on Future Trends of Distributed Computing Systems (FTDCS 2001). IEEE 2001. pp. 98-104.
7. Guy-Ari, G. Empower RMI with TRMI. JavaWorld. Nr. 9. IDG 2002. http://www.javaworld.com/javaworld/jw-08-2002/jw-0809-trmi_p.html
8. Orfali, R., Harkey, D. Client/Server Programming with Java and Corba. John Wiley & Sons, Inc. 1998.
9. Hunter, J., Crawford, W. Java Servlet Programming, O'Reilly & Associates.1998.
10. Fleury, M., Reverbel, F. The JBoss Extensible Server. Proc. of the ACM Intl. Middleware Conference. LNCS 2672. Springer 2003. pp 344-373.
11. Suri, N., Bradshaw, J., Breedy, M., Groth, P., Hill, G., Jeffers, R., and Mitrovich, T. An Overview of the NOMADS Mobile Agent System. Proc. of the 2nd Intl. Symposium on Agent Systems and Applications. ACM 2000. pp. 94-100.
12. Al-Bar, A. Wakeman, I. Camel: A Mobile Applications Framework. Intl. Conference on Computer Networks and Mobile Computing. IEEE 2003. pp. 214-223.
13. Tilevich, E., Smaragdakis, Y. J-Orchestra: Automatic Java Application Partitioning. Proc. of the 16th European Conference on Object-Oriented Programming (ECOOP). LNCS 2374. Malaga, Spain. Springer 2002. pp. 178–204.
14. Binder, W, Hulaas, J., Villazón, A., Vidal, R. Portable Resource Control in Java: The J-SEAL2 Approach. Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001). ACM 2001. pp. 139-155.
15. CROSSWARE - An Autonomic Cross-Platform Operating Environment for On Demand Internet Applications. Marburg, Germany. 2005. http://crossware.org
16. Paal, S., Kammüller, R., Freisleben, B. Self-Managing Remote Object Interconnections. Proc. of the 15th Intl. Conference on Database and Expert Systems (DEXA 2004). Zaragoza, Spain. IEEE 2004. pp. 758-763.

17. Paal, S., Kammüller, R., Freisleben, B. Customizable Deployment, Composition and Hosting of Distributed Java Applications. Proc. of the 4th Intl. Symposium on Distributed Objects and Applications (DOA 2002). LNCS 2519. Springer 2002. pp. 845-865.
18. Fleischmann, M., Strauss, W., Novak, J., Paal, S., Müller, B., Blome, G., Peranovic, P., Seibert, C., Schneider, M. netzspannung.org. Proc. of the 1st Conf. on Artistic, Cultural and Scientific Aspects of Experimental Media Spaces (CAST01). Germany 2001. pp. 121-129.
19. AWAKE - Networked Awareness for Knowledge Discovery. Fraunhofer Institute for Media Communication. St. Augustin, Germany. 2003. http://awake.imk.fraunhofer.de
20. Paal, S., Kammüller, R., Freisleben, B. Crossware: Integration Middleware for Autonomic Cross-Platform Internet Application Environments. International Journal on Computer Aided Engineering. 2005 (to appear).

Garbage Collection in the Presence of Remote Objects: An Empirical Study

Witawas Srisa-an, Mulyadi Oey, and Sebastian Elbaum

Computer Science and Engineering,
University of Nebraska-Lincoln,
Lincoln, NE 68588-0115
{witty, moey, elbaum}@cse.unl.edu

Abstract. Most virtual machine implementations employ generational garbage collection to manage dynamically allocated memory. Studies have shown that these generational schemes work efficiently in desktop-like applications where most objects are short-lived. The performance of generational collectors, however, has been rarely studied in the context of distributed systems.

Given the increasing popularity of such systems, and the distinct type of objects they introduce to support the distributed paradigm, providing insights into their memory allocation behavior could have a large impact on the design of future garbage collection techniques, and in the implementation of such distributed systems as well.

This work presents one of the first attempts to characterize the lifespan of objects in distributed systems. First, we empirically study the differences in lifespan of remote and local objects. Second, we investigate the effects of ephemeral heap size and workload on the lifespan or remote objects. Last, we utilize the insights gained through the experiment to improve the efficiency of a generational collection scheme through the segregation of objects based on their locality.

1 Introduction

Garbage collection (GC) is an automatic process to reclaim unused dynamically allocated memory. By using garbage collection, programmers are relieved from the task of managing dynamic memory—a fault prone activity that is likely to impact software robustness (1; 2) and productivity (2; 3). As a result, many modern languages such as Java and computing platforms such as .NET adopt garbage collection as a standard feature to support dynamic memory management.

Generational garbage collection (2; 4) is the method of choice for most virtual machine implementations (5; 6; 7). With the generational schemes, objects are segregated into two or more heap regions based on their "age" (2). Having separate generations enables the utilization of different heap-collection rates (2; 4), which is likely to increase collection efficiency. For example, by setting a small "ephemeral" heap for initial object allocation and a larger "mature" heap for long lived objects, we can reduce the scope of frequent collections to the small ephemeral heap and perform the more expensive mature collection less often.

Several studies have been performed to assess the performance of generational collection (8; 4). In most instances, shifting from approaches with long pauses such as

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1065–1082, 2005.

mark-sweep to generational collectors has resulted in substantial performance gain. A comparison between a mark-sweep-compact collector with a generational collector showed that generational collection can reduce the pause times by as much as 87% (8). Studies have also found that generational collection performs very well when the survival ratios of objects in the ephemeral generation are small. On the other hand, in applications with a high percentage of long-lived objects, the pause times of generational scheme can exceed the pause times of mark-sweep-compact (8; 9). However, applications with long-lived objects are not common in desktop environments (the basis for many of these studies), and thus, the generational scheme has been widely adopted.

As garbage collection is used to support larger and distributed applications (e.g., J2EE web services, legacy distributed applications migrated to .NET (10; 11; 12)), using features such as Remoting to invoke methods residing in various virtual machines, the lessons learned through previous studies must be revisited to determine whether they are still applicable, and whether new guidelines and heuristics to drive garbage collection are needed.

In this work, we explore whether the rapidly growing body of programming infrastructure that takes advantage of remote resources may challenge what we know about object lifespan and its corresponding impacts on garbage collection. **In this type of infrastructure, a large number of objects are created to mainly serve remote requests. We refer to these objects as remote objects.** Given the growing pervasiveness of distributed applications utilizing remote objects, and the potential distinct behavior of these objects, a re-characterization of what we know about object lifespan is likely to lead to the development of more efficient garbage collection schemes for this type of context.

The contributions of this paper are three-fold. First, we characterize distributed applications through a controlled experiment that includes three distributed applications written in C#. Our characterization, aimed to quantify the differences in the lifespan of remote and local objects, revealed that local and remote objects may have distinct and consistent lifespan traits.

Second, we investigate the factors that affect the lifespan of remote objects. These factors include the number of service requests, the heap size, and the number of concurrent clients. Understanding these factors will eventually allow us to provide programmer's guideline that can be used to assess the impact of these factors on the overall performance of the systems, and concentrate our optimization efforts on the factors that could yield the most positive impact.

Third, we utilize the lessons learned to explore variations of the generational collection that include pretenuring (13; 14; 15) and the addition of a separate generation for remote objects (Gen-R). Based on the initial set results, the Gen-R scheme outperforms default-generational and pretenuring approaches, and can dramatically reduce the number of mature garbage collection invocations by 75% and improve the memory usage by 37%.

The remainder of this paper is organized as follows. Section 2 discusses the details of our empirical study. Section 3 reports our findings. Section 4 details possible threats to the validity of our work. Section 5 discusses proposed variations to the generational garbage collection. Section 6 surveys related research efforts. Section 7 provides the conclusions and avenues of future work.

2 Empirical Study

Widely adopted virtual machines use generational garbage collection to manage both local and remote objects (5; 16). This usage reflects the implicit assumption that the lifespan of remote objects is as short as that of the local ones. If, however, remote objects are long-lived, current generational garbage collection techniques are likely to perform poorly in the presence of remote objects. Therefore, **our first research question (RQ1) is whether the lifespan of remote objects differs from the lifespan of local objects**.

Independently of the differences in lifespan, we need to understand whether the factors affecting the lifespan of remote objects are the same (and similar in magnitude) as of local objects. For example, we are currently able to estimate the lifespan of local objects for a certain application based on the scope of objects' references (2). On the other hand, we have yet to understand how such factor would affect remote objects. Furthermore, it is conceivable that other factors are more prevalent in determining the lifespan of remote objects. **Our second research question (RQ2) then attempts to identify some of the factors influencing the lifespan of remote objects**. In the next sections, we provide details on the apparatus necessary to address these research questions.

2.1 Experimental Platform

We conduct our experiment on the Microsoft Shared Source Common Language Infrastructure (SSCLI) (6). The SSCLI is shared source implementation of the Common Language Infrastructure standard, which offers a compact code base to facilitate quick build cycles and includes the Remoting framework that supports a growing body of .NET distributed applications. Similar to many Java Virtual Machines (5; 16) and the commercial CLR (17), the SSCLI utilizes generational garbage collection with two generations, ephemeral and mature. It also has an additional storage space for large objects. To conserve memory usage, the SSCLI provides a mechanism to quickly enlarge the mature heap space; thus, the initial size for the mature space is set to be very conservative and enlarged automatically as needed (6).

Table 1. Characterization of the Objects for Experimentation

Program	Size		Activation Mode	Initial Lease Time	Message Transport
	Line of Code	Methods Compiled			
Calendar.NET	1,137	3,699	Singleton	5 minutes	HTTP/Binary
eStore.NET	1,206	2,906	Hybrid	5 minutes	HTTP/SOAP
P2P.NET (1:3 - server)	876	2,030	SingleCall	N/A	FTP/Binary
P2P.NET (1:1 - neutral)	876	2,035	SingleCall	N/A	FTP/Binary
P2P.NET (3:1 - client)	876	1,860	SingleCall	N/A	FTP/Binary

In this computing platform, the term "remotable object" refers to objects that can cross or access through the .NET Remoting boundaries. Two activation processes can be used to instantiate and initialize remotable objects: *server activation* and *client activation*. Under the server activation, remotable objects are managed by the server. There are two server activation mode semantics: *Singleton mode* and *SingleCall mode*. Singleton mode ensures that there is only one instance of the Singleton-mode remotable

type activated at any time. The lifespan of Singleton object is managed by a leasing mechanism that can be set by the users (default lease time is five minutes). However, this time can be extended if the object is still in use (the extension time is also user-configurable[1]). The default lease time is used in our experiment. SingleCall mode on the other hand, guarantees that the server will activate a new instance of the remotable type for every client request, and this instance will be collected in the next garbage collection after the method exits.

Under the client activation, each client request creates a client-activated object on the server. Again, leasing is used to maintain liveness with the default lease time of five minutes. Studies have shown that pure client activation mode can introduce a major security risk since compiled objects have to be sent to the client (18). Therefore, a Hybrid mode, which is a combination of the Singleton and the pure client activation mode, is introduced. In the Hybrid mode, a Singleton object is exposed to provide methods to create client-activated objects. In effect, the Singleton object essentially acts as a manager to set up client-activated objects to serve the requests. The Hybrid mode is the one used in place of client activation mode in our experiments with the default lease time.

Users can also configure the communication channels and message formatters. The .NET Remoting infrastructure provides two types of channels: *TCP* and *HTTP*. The messages can be transmitted in either Binary or SOAP format. We use all possible channels and formatters in our experimental objects.

2.2 Variables and Measures

We conjecture that the independent variables that can affect the lifespan of remote objects include program attributes, heap size, and workload. Program attributes include the percentage of allocated local and remote objects, the activation mode, and the message transport mechanisms. As it will be described in Section 2.4, we utilize three experimental objects (one of them in multiple configurations). Each one of these objects has a distinct activation mode and a transport mechanism, but all have more than 50% of remote objects.

To address RQ1, we first experiment with different ephemeral heap sizes (ranging from 800 KB to 8 MB). Since the SSCLI has a policy to enlarge the ephemeral heap if GC is invoked too frequently, we monitor this mechanism and choose the size that all applications can operate without excessive enlargement requests. We find that 4 MB is a good ephemeral heap size for all applications. We then monitor the amount of objects that survive ephemeral and mature collections in our experiment.

To address RQ2, we also manipulate the size of the ephemeral generation to understand its impact on the objects lifespan. We consider three levels of ephemeral heap sizes to be defined in Section 3.

In addition, we manipulate the workload submitted to the experimental objects. Given a fixed number of requests per client, we simulate different loads by modifying the number of clients. Such load adjustments are expected to alter the allocation and lifespan of local and remote objects as the level of services required from the application varies. We consider three levels of workloads to be defined in Section 3.

[1] We use all default values for the leasing mechanism.

Our primary dependent variable is the object lifespan and we utilize two metrics for its characterization, the percentage of surviving objects in the ephemeral heap (ephemeral survival ratio, *es*) and the percentage of surviving objects in the mature generation area (mature survival ratio, *ms*).

$$es = \frac{survived}{allocated} \tag{1}$$

$$ms = 1 - \frac{collected}{matureObject} \tag{2}$$

Notice that $matureObject$ represents the **accumulation of surviving objects** in the mature generation. These metrics are collected after every GC invocation and their values are averaged over whole runs for posterior analysis.

Table 2. Runtime Characterization for the Baseline Settings

Program	Allocated Objects (MB)		Allocated Objects (%)		GC Invocations		Managed Threads
	Local	Remote	Local	Remote	Ephemeral	Full	Created
Calendar.NET	68.28	246.18	21.71	78.29	79	39	58
eStore.NET	297.37	375.52	44.19	55.81	94	46	34
P2P.NET (1:3 - server)	64.79	227.85	22.14	77.86	43	21	35
P2P.NET (1:1 - neutral)	34.25	71.49	32.39	67.61	19	8	34
P2P.NET (3:1 - client)	25.12	24.80	50.32	49.68	10	4	27

2.3 Hypotheses

Having identified variables and measures, we can now restate our research questions more specifically in terms of hypothesis, and supplement them with conjectures about what we expect to observe when the experiment is performed.

Our first research question inquired whether the average lifespan of remote objects is different from that of local objects. We conjecture that remote objects' average lifespan is longer than local objects'. Our conjecture is based on two observations. First, the common Singleton and client-activated objects tend to be long-lived. Singleton objects often live for the duration of the server program (19) and the lifespan of client-activated objects are commonly controlled by a conservative leasing mechanism.

Second, most connected objects tend to die together (20). For example, if a Singleton object is the root of a cluster of tightly associated remote objects, then these remote objects are likely to live as long as the Singleton object. On the other hand, a SingleCall object only lives for the duration of a service, and therefore remote objects connected to the SingleCall root should be shorter lived that those connected to a Singleton root. This observation also leads to another conjecture that the long term survival of remote objects depends on the activation mode. We would like to investigate the validity of these conjectures empirically which lead us to the following set of hypotheses.

H1.1: The survival ratios of remote objects in the ephemeral heap are greater than the ones for local objects.
H1.2: The collection ratios for remote objects in the mature heap are affected by activation modes.

Our second research question aims at identifying and characterizing the factors that may influence the lifespan of remote objects. We have identified two potential factors. First, we conjecture that application workload, as measured by the number of concurrent clients and requests, is likely to affect the lifespan of remote objects. Our goal is to investigate the impact of different workloads on the garbage collection performance which leads to the following hypothesis:

H2.1: Variations in workload level affect the survival ratio of remote objects.

Second, the size of the ephemeral generation is likely to impact the survival ratio of remote objects. In general, a larger heap is expected to lead to smaller survival ratios. However, its impact when managing objects with two distinct lifespans is currently not known. The result of our study can provide opportunities for different optimization techniques discussed in Section 5. We explore this further through the following hypothesis:

H2.2: Variations in heap size affect the survival ratio of remote objects.

2.4 Experimental Objects

To address our research questions we need a set of programs that are representative of real-world applications employing remote objects to perform their activities. Since these programs will be used for experimentation, we must be able to manipulate the way they operate. This implies accessibility to their source code and control on their execution context. We have invested a significant effort in assembling a first set of such experimental objects. (Section 4 discusses the potential impact of this approach in the validity of our findings and Section 7 provides a mechanism for other researchers to have access to these objects.)

Our effort began with the identification of application areas that use Remoting. This process consisted of surveying related books and articles (19; 1; 21; 22; 18; 23; 24), and contacting our Microsoft collaborators for applications utilizing this feature. We found that .NET Remoting, RMI, and Enterprise JavaBeans are most commonly used to support generic client/server applications, Peer-to-Peer (P2P) transactions, middleware for connecting web-based front ends with legacy database systems utilizing XML (25), and to provide Web services in Microsoft COM+ (26). This process resulted in several potential application areas. Given our expertise, the constraints of the platform (e.g., SSCLI does not provide all the features available in the commercial version), and the long-term goal of collecting a representative set of experimental objects, we selected the three applications listed in Table 1 to perform this study. Table 1 also describes the configuration of each application.

The first application is a calendar program that keeps track of appointments for different clients in multiple XML flat files. Calendar.NET is a three-tier application with front-end Remoting clients, an application server, and an XML-based information server. The application server exposes remotely-accessible methods that can be invoked by the clients via Remoting. The connection between the application server and clients depends on the chosen activation mode, whereas the connection between the application server and the information server is done in a Singleton fashion. We choose an XML

back-end to represent a common practice of converting data from a legacy repository to XML (e.g. XDB gateway(25)).

The second application is eStore.NET, a basic Web Services application. Similar to Calendar.NET, eStore.NET consists of three tiers, but it utilizes MySQL (27) as the database server[2]. A client can query the store's catalog, check for product availability, place an order, or complete a transaction by providing payment information (28). The database structure and content are adaptations from the Northwind sample database provided by Access 2003 (29).

The third application is P2P.NET, a non-centralized peer-to-peer file sharing application. The application represents a common peer-to-peer file sharing model where a peer node acts as a service provider and as a customer. We configure our application to continuously make sequential requests to other peers. At the same time, each node also has the responsibility of servicing concurrent requests from other peers. Thus, when the program acts as a customer, most objects created to fulfill this activity are local. On the other hand, when the program takes the role of a server, most objects created for this task are remote due to remote method invocations by other peers. It is worth noting that during the experiment we monitor the workload of one node. Since we make the workload of all nodes uniform, we can randomly pick any node to monitor. We experiment with three configurations by changing the balance of requests versus services for the observed node: 25% client/75% server, 50% client/50% server, and 75% client/25% server.

Runtime Objects Characterization. In this study, our activities mainly focus on the object behaviors of servers and peer-to-peer applications during the heaviest workload. During this period, the garbage collector needs to be extremely efficient to minimize pauses that can degrade response time. In our study, we configure our experimental objects to accept continuous requests made by a large number of clients and to provide a heavy workload to stress the garbage collection system. The workloads placed on our experimental objects are representative of the server environment.

Table 2 illustrates the runtime behaviors of our experimental objects responding to a heavy workload. We report the average allocation of remote and local objects between GC intervals. In most applications, the amount of allocated remote objects are larger than the amount of allocated local objects. This is as expected since most of our experimental objects are configured to provide various level of services.

For each experimental object, we experiment with different number of clients and requests until a continuous workload is sustained. The configuration is then used as the baseline to conduct more experiments (Section 3).

2.5 Instruments for Experimentation

We perform our experiments by running the experimental objects with the different combinations of independent varibles on a modified SSCLI. The modified SSCLI generates information summarized in Table 3.

[2] Notice that SSCLI does not have a ADO.NET library which limits its ability to connect to external database. We are able to modify an existing MySQL library written for the Commercial CLR to work with the SSCLI.

Table 3. Description of Generated Information

Context	Collected Information
Object allocation	Object id, size, class name, location, and allocator thread's id.
Thread	Id of created thread. Id of destroyed thread.
Garbage collection	Id of object that gets promoted. Id of pinned object (in ephemeral generation). Id of marked object (in mature generation).
Remote method	Id of thread entering remote method. Id of thread exiting remote method.

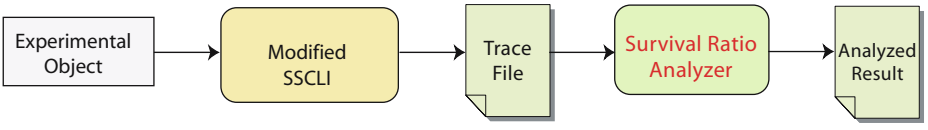


Fig. 1. Overview of the Experimental Platform

In the SSCLI, there is a common function that is used to build a remote method callstack. We instrument this function to identify all remote method calls. Each time a remote method is entered, the modified SSCLI will generate information in the trace file. The information includes the method name and a unique identification number of a particular thread executing the method. The basic overview of our experimental platform is provided in Figure 1.

We create a program that analyzes the trace to identify (i) the object type, (ii) effiencies of ephemeral and mature collections, and (iii) allocation ratios of local and remote object to heap size during each collection. Essentially, the analyzer has to detect whether an execution thread is currently executing a local or a remote method. If an object is allocated while a remote method is in scope (i.e. there is at least one remote method on the allocator thread's invocation stack), then the object is considered as remote. On the other hand, an object is classified as local if it is allocated when only local methods are in scope (i.e. no remote method on the invocation stack). We identify each object's type (i.e. local or remote), calculate the garbage collection efficiency for both ephemeral and mature regions, and compute the object size distribution.

3 Results

In the following sub-sections, we present the results from our experiments aimed at answering the research questions proposed in Section 2.

3.1 Lifespan of Remote and Local Objects

To answer this research question, we conduct a series of experiments to monitor the amount of objects that survive both ephemeral and full GC invocations. To validate our hypotheses (H1 and H2), these surviving objects are segregated into two types: local and remote.

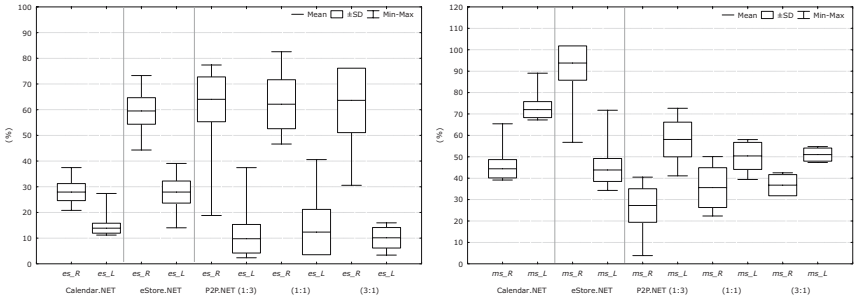


Fig. 2. Lifespan of Remote and Local Objects for Ephemeral (left) and Mature (right) Collections

Ephemeral Generation. The box plot in Figure 2 (left) summarizes the survival ratios for remote and local objects in the ephemeral generation across all experimental objects. The *x-axis* indicates the program name and object type (remote objects are represented by *es_R* and local objects are represented by *es_L*). The *y-axis* represents the survival ratio. The average survival ratios are represented by the horizontal line in the middle of each box. The upper and lower edges of the box define the standard deviations. The upward extending whiskers represent the maximum survival ratios in the observed data. The downward whiskers represent the minimum survival ratios.

For all programs, more remote than local objects are migrated from the ephemeral to mature generations. In Calendar.NET, we see the difference in survival ratio of approximately 15%. In eStore.NET, the difference widens to 30%. P2P.NET has the greatest differences in survival ratios with over 50%. We also find that when P2P.NET is configured as a server—appeared as 1:3 in Figure 2 (left)— there is a very wide range of survival ratios (from 18% to 77%. This wide variation may be caused at least partially by the use of SingleCall activation mode which allows the server to complete the initial requests very quickly, but struggles as the workload increases. Increases in the workload queue up more requests that thus leads to extended lives for remote objects.

Mature Generation. Figure 2 (right) provides a box plot corresponding to the the survival ratios for remote (*ms_R*) and local (*ms_L*) objects in the mature generation. For eighty percent of the programs, the surviving remote objects seem mostly semi-long-lived. That is, they are collected soon after they are copied into the mature generation. The survival ratios for remote objects in the mature heap range from 13% to 32% lower than those of local objects. This is not surprising since Calendar.NET and P2P.NET use Singleton and SingleCall activation modes, respectively.

In Singleton mode, there is only one remotable object that acts as root to clients' requests. However, there is a large number of objects created to synchronize accesses to the Singleton object. The lifespan of these synchronization objects mainly depend on the time taken to satisfy the clients' requests. Therefore, a portion of remote objects that survives ephemeral collection is for synchronization and is semi long-lived. On the other hand, objects that are connected to the roots are truly long-lived. This explains why we are able to collect a good amount of remote objects in each mature collection invocation. For SingleCall, remotable objects would die as soon as the requests are

completed. Thus, we find that most remote objects are semi long-lived and can be reclaimed by the mature collection. It is also interesting to note that the survival ratio for Singleton mode is much higher than that for SingleCall mode. The difference is about 18% when comparing Calendar.NET with P2P.NET [1:3].

In the case of eStore.NET, we find the nature of the application requires for remote objects to be truly long-lived (94% survival ratio). The three major sources for this prolonged lifespan are: 1) the states for each client's transactions are maintained thereby "stateful" objects are kept as long as the client is alive, 2) each client's initial remote method call creates one database connection and these connections remain active until the client disconnects, 3) the Hybrid activation creates one exclusive root for every client. When compared to Singleton, there are more root objects in this activation mode. Each of these roots is guaranteed to live for at least five minutes and therefore, many remote objects become very long-lived.

The behaviors reported in this section have led us to two important observations:

1. Remote objects consistently outlive the local objects in the ephemeral generation.
2. The activation mode and the nature of the application can affect the lifespan of remote objects in the mature generation.

3.2 Factors that Affect Lifespan of Remote Objects

In this section, we study two factors that can affect the survival ratio of remote objects. These factors include ephemeral heap size (we double and halve the baseline heap size) and workload (we lower and raise the number of simultaneous clients and number of uploads/downloads). Our goal is to compare the survival ratios as these parameters change.

Effect of Heap Size. The results of our experiments are reported in Figure 3 in which the *x-axis* represents the average survival ratio and the *y-axis* represents all the program and corresponding *es_L* and *es_R*.

A larger ephemeral generation results in a decreasing survival ratio for remote objects in all instances. However, the decrease is less noticeable in programs acting mainly as servers. For example, P2P.NET [1:3] (server configuration) shows a 2% survival ratio reduction while P2P.NET [3:1] (client) shows a 9% reduction.

On the other hand, the decrease in the survival ratios for local objects are not as pronounced as those of remote objects. This is due to the fact that the baseline heap size (4 MB) provides enough time for most of the short-lived objects to die. Providing additional time through a larger heap only affects a small percentage of the remaining objects. These behaviors have led to two additional observations:

1. Enlarging the ephemeral heap had a minor effect on the lifespan of local objects.
2. Enlarging the ephemeral heap caused minor reductions (2 - 7%) in the survival ratios for remote objects in applications that have been configured to operate as server (Calendar.NET, eStore.NET, and P2P.NET [1:3]).

Notice that when the heap size is reduced in half, the survival ratios in Calendar.NET, eStore.NET, and P2P.NET (server and neutral configurations) for both local

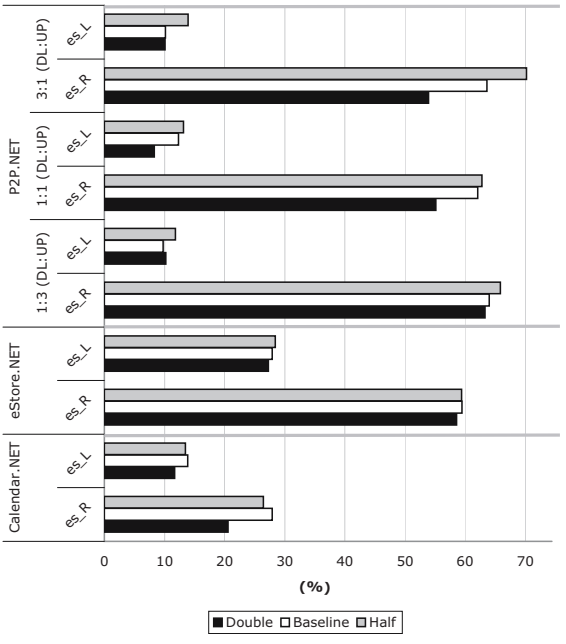


Fig. 3. Effect of Heap Size on the Survival Ratio

and remote objects are almost the same as the those of 4 MB ephemeral heap size. This is because 2 MB is too small for these programs to operate. Therefore, the SSCLI increases the ephemeral heap size to 4 MB after a few ephemeral GC invocations. For P2P.NET (client configuration), the program can operate with 2 MB so we see significant differences in the survival ratios when compared to 4 MB ephemeral size.

Effect of Workload. This experiment consists of manipulating the workload in all applications by reducing and increasing the number of requests. For Calendar.NET and eStore.NET, the task is accomplished by increasing the number of simultaneous clients. For P2P.NET, we increase the number of peers, resulting in an increasing number of files to be downloaded and uploaded. The detailed configurations used in our experiments are shown in Table 4.

Table 4. Workload Configurations

Program	Lower Workload	Baseline Workload	Higher Workload
Calendar.NET (100 requests/client)	15 clients	30 clients	45 clients
eStore.NET (50 requests/client)	15 clients	30 clients	45 clients
P2P.NET (1:3)	40 peers (120:360)	80 peers (240:720)	160 peers (480:1440)
P2P.NET (1:1)	40 peers (120:120)	80 peers (240:240)	160 peers (480:480)
P2P.NET (3:1)	40 peers (120:40)	80 peers (240:80)	160 peers (480:160)

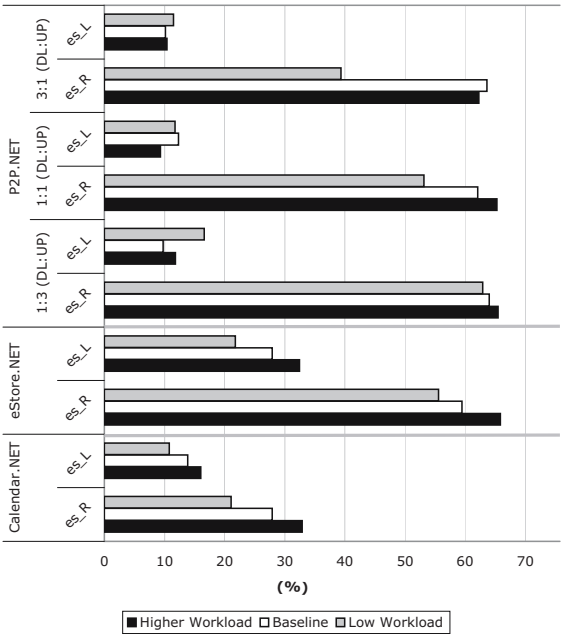


Fig. 4. Effect of Workload on the Survival Ratio

The results are summarized in Figure 4. We see that Calendar.NET and eStore.NET show consistent increases in the survival ratios of both remote and local objects when the workload becomes heavier. This is expected as more workload would result in longer time to complete a client request and prolonged object lifespan. Still, the increments are observably more noticeable for remote objects. Furthermore, P2P.NET presents similar tendencies for remote objects but not for local objects. We attribute such variation to uncontrolled factors such as network latency.

Based on the experimental results, we observe that the amount of workload may affect the lifespan of **both** local and remote objects in distributed applications, but its effect seems more consistent on remote objects.

4 Threats to Validity

In this section we present a synthesis of the potential threats to validity of our study and explain how we tried to reduce their impact on our findings.

As in any controlled experiment, external threats to the validity of our results are the major source of concerns as they limit the generalization of our findings. First, our set of programs only has three objects. However, the chosen objects are representative of a wide variety of programs utilizing remoting. It is also necessary to consider that this is the first group of programs prepared to perform this type of experiment. Second, the use of the SSCLI also limits generalization since it provides only part of the functionality available in Microsoft's commercial *Common Language Runtime* (*CLR*) (6) and other

distributed environments. We controlled this threat by extending the SSCLI functionality by adapting openly available packages (e.g., since ADO.NET is not part of the SSCLI, we modified the connectivity features in the MySQL package to support database access). Still, we realize there may be behavioral differences between infrastructure supporting distributing systems that were not accounted in the experiment. Third, our scenarios represented only the peak-load periods and our results should be analyzed in such context; still these periods are very interesting because they are likely to expose the system to worst case scenarios. The same limitation applies to our choices of memory configuration and collection policies. Last, our distributed scenarios were performed on a limited deployed hardware infrastructure. This was a conscious experimental decision to help us control other sources of variation (e.g., network latency).

The second set of threats, to internal validity, has to do with influences that can affect the independent variables with respect to causality without the researcher's knowledge. First, we had to modify the SSCLI to observe object behavior (the current built-in profiler interface in the SSCLI offered many capabilities that are not yet fully implemented), and to make the experimental objects operate under extreme loads (e.g., we had to extend the default thread-pool size from 25 to 75). Second, we had to build several tools to analyze and measure that could have faults, biasing the results. We addressed this threat by carefully inspecting these tools on small scenarios that could be monitored and measured through other mechanisms to gain confidence in their correctness. We are also aware of the threat posed by one of the authors being the primary developer of the tools to capture the data and analyze the results. Frequent inspections of random data points from the other authors were utilized to control this potential threat, and we hope that by making the infrastructure available (see Acknowledgements) other researchers can replicate the experiment to gain further confidence in our results.

Threats to construct validity are raised when the measurement instruments fail to capture the concepts they are supposed to capture. In our experiment, the metrics were computed at the intervals defined by GC invocations, which may not be enough to notice small differences in lifespan. Although several techniques could be used to mitigate this problem (e.g. adjust the memory size to cause more frequent collections, further instrument the SSCLI to obtain a more precise trace for analysis), our initial exploration put in evidence that each technique brings associated tradeoffs and cannot be considered in isolation.

5 Discussion

Earlier in the paper we conjectured that if remote objects have different lifespan than local objects, then garbage collection techniques that segregate based on object locality are likely to be more efficient. In this section we provide preliminary evidence to support that conjecture.

We simulate three[3] different garbage collection techniques on the Calender.NET program: 1) bi-generational which is commonly used across virtual machines, 2) pre-

[3] All three techniques maintain the same total initial heap size of 10 MB. This ensures that the results are not due to variances other than the technique itself.

tenuring which consists of moving remote objects directly to the mature heap (for more details see Section 6), and 3) three-generational approach called Gen-R that includes an intermediate generation for remote objects. Gen-R approach utilizes one ephemeral heap for all objects, with any surviving local objects being promoted to the mature generation, while the surviving remote objects are promoted to an intermediate generation (*genR*). Once the *genR* is full, copying collection is used to migrate objects to the mature region.

To perform the simulation we collected lifespan information through the instrumented SSCLI at fixed allocation intervals of 20 KB and generated the following data:

1. Number of GC invocations in the ephemeral and, if applicable, *genR* spaces. In Gen-R, collecting the *genR* space incurs an additional but inexpensive GC invocations.
2. Number of GC invocations in the mature space. A significant reduction in mature collection can present a substantial performance gain.
3. Heap usage. Performing ephemeral and *genR* collection efficiently can result in lower heap usage[4].
4. Traversal of lived objects in the mature space. In the SSCLI and Sun's HotSpot VM, mark and sweep is used to collect objects in the mature generation. Marking phase has been known to be the most time consuming part in the mark-sweep approach (2). Thus, reduction in marking would also improve the performance of mature collection.

Table 5. Simulation Results (Bi-generational, Pretenuring, and Gen-R Schemes)

Algorithm	Configuration [eph:genR:mature] (MB)	Minor Collection	Full Collection	Heap Usage (MB)	Marked Objects (Million)	Marked Objects / Full GC (Million)
Bi-generational	4:0:5	57	4	19	20.1	5.0
Pretenuring	4:0:5	14	34	39	181.2	5.3
Gen-R	4:2:3	57 + 9	1	12	2.7	2.7

Table 5 lists the basic heap configuration and the results of each GC scheme. We find that pretenuring all remote objects can cause a serious performance degradation. For example, the system needs to invoke full collection nearly 8 times more than the bi-generational scheme. It seems that simply pretenuring all remote objects is not a suitable approach in this type of environment.

On the other hand, Gen-R scheme reduces the number of full GC invocations by 75% compared with the bi-generational scheme, and the application uses 38% less heap space to operate. The average number of objects that have to be marked in each full GC are also reduced by from 5 million to 2.7 million objects or 46% reduction. As expected, Gen-R does invoke 16% more minor collections (i.e. 9 additional *genR* collections), but the costs associated with these collections are known to be significantly smaller than full collection.

[4] In the SSCLI, there is a policy that allows the mature heap to grow if the current size is too small. We also implement the same policy into our simulator.

6 Related Work

This section describes efforts related to this work categorized into three areas: object characterization, management of long-lived objects, and trace generation techniques.

In terms of object characterization, the extensive study of SPECjvm98 (30) benchmarks by Dieckmann et al. (31) yielded many insights into the allocation and garbage collection behaviors of Java programs. For example, their study was one of the first for Java to confirm the weak hypothesis that most objects die young. However, the effect were not as pronounced as functional languages such as SML/NJ (31; 32; 2).

In terms of lifetime management, there have been several studies in previous distributed computing platforms. For example, CORBA provides explicit means to manage lifetime of an objects, and classifies lifetime into two categories, *transient* and *persistent* (33). There have been many research efforts that study the implementations and behaviors of persistent objects (34; 35). However, such results do not apply directly to remote objects since remote objects are not persistent; they are long-lived transient objects. At the present time, we do not know of any efforts that have been done specifically to study the effect of remote objects on the overall performance of generational garbage collection. However, there have been a few efforts that use pretenuring technique to manage long-lived objects in desktop-like applications (13; 15; 14).

Pretenuring is an approach to manage long-lived objects by placing them directly in the mature region (13). The experiments often involve profiling the lifespan of each object and then annotate the information as advices during object creations (13; 15). Our preliminary simulation results lend some insights into the fact that distributed applications might not benefit from pretenuring alone but might otherwise benefit from other variations of multi-generational approach.

To simulate different generational garbage collection schemes, we need to collect lifespan information from our experimental objects. There are several ways to collect this information. For example, we could call the garbage collector after every allocation to ensure a "perfect" trace, or after certain allocation intervals to generate an approximate but more efficient trace collection (31), or after certain events occur such as the reachable time (36). We plan to utilize these mechanisms as we develop new collection techniques that discriminate between remote and local objects.

7 Conclusions

In this paper, we have performed a series of experiments to better understand the object lifespan in distributed applications. As we have discussed, these experiments, like any other, have several limitations to their validity. Keeping these limitations in mind, we draw several observations from this work, with implications both for practitioners and for researchers.

First, remote objects outlived local objects during ephemeral collection. In our experiments, differences in the survival ratios ranged from 15% to over 50%. However, most remote objects tend to be rapidly collected once reaching the mature heap. Hence, they could be classified as semi-long-lived. Second, activation mode can affect the lifespan of remote objects. We found that when SingleCall activation mode is used, approximately 32% of remote objects survived in the mature collection. However, when

Singleton or Hybrid are used, the survival ratios in the mature heap for remote objects become higher (from 45% to 95%). Third, enlarging the ephemeral heap size slightly affect the survival ratios of local and remote objects. Fourth, workload may have minor effects on the lifespan of local objects, but it is likely to greatly affect the lifespan of remote objects in server applications.

From our findings, we conjectured that garbage collectors that apply distinct collection strategies based on object locality (local or remote) could significantly enhance the overall efficiency of distributed applications. Our initial exploration of a garbage collection scheme that segregates the management of remote objects from local objects seems to support that conjecture. When applied to one of the experimental objects, this simple scheme reduced the heap usage by 38%, the full collection invocations by 75%, and the the average marked objects per full collection by 46%.

Our results suggest several avenues for future work. First, to address questions of whether these results can be generalized, further studies are necessary. We are gathering additional programs for use in such studies. Second, we will develop a more comprehensive set of collection techniques that can fully exploit the segregation of objects based on their locality and measure their performance in real-scenarios. Last, we will explore techniques for predicting and dynamically adjusting garbage collection and heap allocation strategies to gain maximum efficiency tailored for different applications and object distribution scenarios.

Acknowledgment

To gain access to our experimental infrastructure, please contact the first author of the paper. We would like to thank Shiu-Beng Kooi for helping with the creation of the experimental objects. This work was supported in part by the NSF programs under award CNS-0411043, and a CAREER Award 0347518 to University of Nebraska, Lincoln.

References

1. Conger, D.: Remoting with C# and .NET: Remote Objects for Distributed Applications. John Wiley (2003)
2. Jones, R., Lins, R.: Garbage Collection: Algorithms for automatic Dynamic Memory Management. John Wiley and Sons (1998)
3. Rovner, P.: On Adding Garbage Collection and Runtime Types to a Strongly-Typed, Statically-Checked, Concurrent Language. Technical Report CSL-84-7, Xerox PARC (1985)
4. Ungar, D.: The Design and Evaluation of a High Performance Smalltalk System. ACM Distinguished Dissertations (1987)
5. IBM: Jikes research virtual machine. http://www-124.ibm.com/developerworks/oss/jikesrvm/ (2004)
6. Stutz, D., Neward, T., Shilling, G.: Shared Source CLI Essentials. O'Reilly and Associates (2003)
7. Sun Microsystems: Java Embedded Server. http://www.sun.com/software/embeddedserver (2001)

8. Dykstra, L., Srisa-an, W., Chang, J.M.: An Analysis of the Garbage Collection Performance in Sun's HotSpot JVM. In: Proceedings of 21th IEEE International Performance Computing and Communications Conference (IPCCC), Phoenix, Arizona (April 3-5, 2002) 335–339
9. Yang, Q., Srisa-an, W., Skotiniotis, T., Chang, J.M.: Java Virtual Machine Timing Probes: A Study of Object Lifespan and Garbage Collection. In: Proceedings of 21st IEEE International Performance Computing and Communication Conference (IPCCC-2002), Phoenix Arizona (April 3-5, 2001) 73–80
10. Abramson, D., Watson, G., Dung, L.: Guard: A Tool for Migrating Scientific Applications to the .NET Framework. In: Proceedings of International Conference on Computational Science (ICCS 2002), Amsterdam, The Netherlands (2002) 834–843
11. Buss, LTD.: SIMPLEPLOT.NET. http://www.buss.co.uk/buss/dnet.htm (2002)
12. Nutt, G.: Distributed Virtual Machines: Inside the Rotor CLI. Addison Wesley (2005)
13. Blackburn, S.M., Singhai, S., Hertz, M., McKinely, K.S., Moss, J.E.B.: Pretenuring for java. In: Proceedings of the OOPSLA '01 Conference on Object Oriented Programming Systems Languages and Applications, Tampa Bay, FL (2001)
14. Huang, W., Srisa-an, W., Chang, J.: Dynamic Pretenuring Schemes for Generational Garbage Collection. In: Proceedings of 2004 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, Texas (2001) 133–140
15. Jump, M., Blackburn, S.M., McKinley, K.S.: Dynamic Object Sampling for Pretenuring. In: ISMM '04: Proceedings of the 4th International Symposium on Memory Management, ACM Press (2004) 152–162
16. Sun Microsystems: The java hotspot virtual machine, v1.4.1. http://java.sun.com/products/hotspot/ (2003)
17. Richter, J.: Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework. http://msdn.microsoft.com/msdnmag/issues/1100/GCI/default.aspx (2000)
18. Rammer, I.: Advanced .NET Remoting. Apress (2002)
19. Browning, D.: Integrate .net remoting into the enterprise. In: .NET Magazine. (2002)
20. Hirzel, M., Henkel, J., Diwan, A., Hind, M.: Understanding the connectivity of heap objects. In: ISMM, Berlin, Germany (2002)
21. McLean, S., Naftel, J., Williams, K.: Microsoft .NET Remoting. Microsoft Press (2003)
22. Obermeyer, P., Hawkins, J.: Microsoft .NET Remoting: A Technical Overview. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp (2005)
23. Srinivasan, P.: An Introduction to Microsoft .NET Remoting Framework. http://msdn.microsoft.com/library/en-us/dndotnet/html/introremoting.asp (2001)
24. Wiener, R.: Remoting in C# and .NET. Journal of Object Technology **3** (January 2004) 83–100
25. XML.com: XDB Gateway (Database-to-XML Gateway). http://www.xml.com/pub/p/760 (2003)
26. Long, J.: Remoting in microsoft product. Private e-mail correspondant (2004)
27. MySQL-AB: Mysql dbms. http://www.mysql.com (2004)
28. Pietrek, M.: Under the Hood, http://msdn.microsoft.com/msdnmag/issues/01/12/hood/. (2005)
29. Microsoft: Microsoft office online. http://office.microsoft.com (2004)
30. SPEC: Standard performance evaluation corporation jvm98. http://www.spec.org/osg/jvm98 (1999)
31. Dieckmann, S., Hölzle, U.: A Study of the Allocation Behavior of the SPECjvm98 Java Benchmarks. In: Proceedings of the European Conference on Object-Oriented Programming. (1999)

32. Stefanović, D., Moss, J.E.B.: Characterization of Object Behaviour in Standard ML of New Jersey. In: LFP '94: Proceedings of the 1994 ACM conference on LISP and functional programming, ACM Press (1994) 43–54
33. Vinoski, S.: New Features for CORBA 3.0. Commun. ACM **41** (1998) 44–52
34. Kleindienst, J., Plasil, F., Tuma, P.: Lessons Learned from Implementing the CORBA Persistent Object Service. In: Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM Press (1996) 150–167
35. Ryu, S.W., Neuman, B.C.: Garbage collection for distributed persistent objects. http://www.objs.com/workshops/ws9801/papers/paper015.html (1998)
36. Hertz, M., Blackburn, S.M., Moss, J.E.B., McKinley, K.S., Stefanović, D.: Error-free garbage collection traces: how to cheat and not get caught. In: SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, ACM Press (2002) 140–151

Peer-to-Peer Distribution Architectures Providing Uniform Download Rates

Marc Schiely and Pascal Felber

Computer Science Department, University of Neuchâtel,
CH-2007, Neuchâtel, Switzerland
marc.schiely@unine.ch, pascal.felber@unine.ch

Abstract. Peer-to-peer (P2P) networks have proved to be a powerful and highly scalable alternative to traditional client-server architectures for content distribution. They offer the technical means to efficiently distribute data to millions of clients simultaneously with very low infrastructural cost. Previous studies of content distribution architectures have primarily focused on homogeneous systems where the bandwidth capacities of all peers are similar, or simple heterogeneous scenarios where different classes of peers with symmetric bandwidth try to minimize the average download duration. In this paper, we study the problem of content distribution under the assumption that peers have heterogeneous and asymmetric bandwidth (typical for ADSL connections), with the objective to provide uniform download rates to all peers—a desirable property for distributing streaming content. We discuss architectures that fulfill this goal and achieve optimal utilization of the aggregate uplink capacity of the peers. We develop analytical models that provide insight on their performance in various configurations, and we compare them to architectures with non-uniform rates. Our results indicate that heterogeneous and asymmetric peers can achieve uniform download rates with little additional complexity and no performance penalty.

1 Introduction

The distribution of large or streaming content remains a challenging problem in today's Internet. A single source quickly becomes saturated when the number of clients requesting the content grows, which leads to degradation or loss of service. Solutions based on content delivery networks (CDNs) are prohibitively expensive and rather static in nature, while protocols like IP multicast suffer from several flaws and are not widely deployed.

P2P systems, in which peer computers form a cooperative network and share their resources, offer a promising alternative for content distribution. They reduce the load of the primary servers by leveraging the bandwidth of the peers, those receiving part of the content providing it to others. Their low cost, inherent scalability, and resiliency to "flash crowds" (a huge and sudden surge of request traffic that usually leads to the collapse of the affected server) make such systems very attractive for large scale deployments.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1083–1096, 2005.

This work focuses on P2P architectures designed for distributing content among large populations of clients. Unlike previous studies, we assume that the peers have heterogeneous and asymmetric bandwidth (typical for ADSL connections) and we aim at providing a uniform download rate to each of them. This property is crucial for applications like media streaming, for which users expect an uninterrupted stream of data at a constant rate.

We consider simple models with two classes of peers that differ in their uplink capacities. We study several architectures that achieve optimal utilization of the aggregate uplink capacity of the system and share it equally between all the peers. It obviously follows that fast peers must share more bandwidth than they receive, but we can balance this unfairness by placing them nearer to the source for increased reliability and shorter latency.

We develop analytical models that provide interesting insight on the performance of content distribution architectures with uniform download rates in various configurations. We compare them with other architectures providing non-uniform rates and we conclude that uniformity can be achieved with little additional complexity and no performance penalty.

The rest of the paper is organized as follows: We first discuss related work in Section 2 and we present the system model in Section 3. Then, we analyze three different architectures providing uniform download rates in Section 4 and compare them in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

There exist two main approaches for dealing with differences in uplink bandwidth in overlay multicast systems. Narada [1], CollectCast [2] and GnuStream [3] use bandwidth measurements to improve the overlay structure by dynamically replacing links. In contrast Scattercast [4], SplitStream [5], Overcast [6] and ALMI [7] use degree-constraint structures to deal with heterogeneity. If a peer's degree is saturated when a new peer wants to connect, then some reorganization needs to take place. CoopNet [8] uses both of these techniques. It deploys multiple parallel trees and reorganizes them based on performance feedbacks.

All of these systems do not try to uniformly distribute the download rate to all peers. Instead, they send distinct streams at different rates, or they consider bounded streams and use buffers to deal with timing problems. Our goal is to minimize these buffer requirements by evening out the download rate at all peers.

In [9], the authors investigate the impact of heterogeneous uplink bandwidth capacities on Scribe [10]. Their experiments show that heterogeneity may create distribution trees with high depths, which is not desirable. After proposing several ways to address the problem they conclude that heterogeneity in DHT-based multicast protocols remains a challenging open problem.

Analytical models have been proposed for peers with homogeneous bandwidth capacities [11, 12], as well as and for heterogeneous peers but for non-uniform download rates [13]. Different architectures for homogeneous and heterogeneous bandwidth constraints are analyzed. In contrast to this work, the

authors make the assumption that the downlink and uplink capacities are symmetric and do not consider uniform download rates.

To the best of our knowledge, no analytical models have been proposed to study P2P content distribution architectures providing uniform download rates to heterogeneous peers with asymmetric bandwidths.

3 System Model and Definitions

For the rest of this paper we use the following model. We assume that nodes in the network have different upload capacities. We analyze content distribution architectures with two classes of nodes, referred to as *fast* and *slow* peers according to their upload bandwidth. All nodes in a class have the same bandwidth. The data stream is sent by a single source which has the same bandwidth as fast nodes. To simplify the analysis, we assume that the source receives the data at the same uniform rate as the other peers before distributing it within the content distribution network. We shall ignore latency in our model.

As is the case for typical ADSL connection, we assume that the slow peers are essentially limited by their uplink capacity and have sufficient download bandwidth to receive the data at the same uniform rate as the other peers.[1] We consider N_f fast peers in class F with upload bandwidth B_f and N_s slow peers in class S with upload bandwidth B_s ($B_f > B_s$). For the sake of simplicity, we assume in our analysis that $B_s = \frac{B_f}{k}$ with k being an integer value. The total number of peers is $N = N_f + N_s$.

We analyze the behavior of different architectures when transmitting a large content. We assume that the file being transmitted is split into C chunks that can be sent independently: as soon as a peer has received a chunk, it can start sending it to another peer. We consider one unit of time to be the time necessary for transmitting the whole content at the uniform rate r that is provided to all peers. Each chunk is thus received in $\frac{1}{C}$ unit of time. For clarity, we shall describe the different architectures with the assumption that we transmit the whole file at once and we shall introduce chunks later in the analysis. As total download time is a function of the number of chunks, our main objective of supporting streaming data corresponds to situations where $C \to \infty$.

A peer may receive chunks from the source via different paths. For instance, in the case of SplitStream [5], the source splits the content into several layers and sends each of them along distinct trees spanning all the nodes. Two chunks sent *at the same time* by the source may thus traverse a different number of peers and be received at different times. This implies that each peer may have to buffer some chunks until all of those sent at the same time have been received. We compute δ_T as the maximal difference in distance between a peer and the closest common node along the paths to the source via distinct incoming links. This value indicates the buffer space needed at the peer. For instance, in Figure 1, the first node of the right chain receives chunks from the source in 1 (directly),

[1] As we shall see, this rate is no higher than the uplink capacity of the fast peers.

2 (via one peer), and 3 (via two peers) units of time and we have $\delta_T = 3 - 1 = 2$. Clearly, small values of δ_T are desirable and we shall also compare the different architectures with respect to this property.

Uniform Rate. As previously mentioned, our goal is to provide the same download rate to all peers in the network. Obviously, the maximal rate r that can be achieved corresponds to the aggregate upload bandwidth of all nodes divided by the number of peers ($B_s < r < B_f$). It is easy to see that a tree cannot be used to fulfill this goal because a slow node does not have enough upload bandwidth to serve even a single other peer at rate $r > B_s$.

A trivial approach is to form chains of peers, in which a combination of slow and fast peers team up and share their bandwidths at each level of the chain. Figure 1 shows such an architecture with 50% fast nodes and 50% slow nodes ($N_f = N_s = \frac{N}{2}$), and slow nodes having half of the upload bandwidth of fast nodes ($B_s = \frac{B_f}{2}$). The source is the topmost node and the numbers show the transmission rate on the corresponding link, as a fraction of B_f. Fast nodes are displayed in gray. The time units indicated in the figure do not explicitly take chunks into account: at $t = 1$, the second peer in the left chain has received the content at rate $\frac{3}{4}$; at $t = 3$, the first peer in the right chain has received the content via three links, each at rate $\frac{1}{4}$; etc. All time units should be divided by C when considering chunks. The download rate r is calculated as:

$$r = \frac{1}{N}(\frac{N}{2}B_f + \frac{N}{2}\frac{B_f}{2}) = \frac{3}{4}B_f$$

We can observe that an unused upload bandwidth of $\frac{3}{4}B_f$ remains because the source does not download any content. We shall ignore it in the rest of the paper.

If we generalize the upload bandwidth of the slow peers to a fraction of the upload bandwidth of the fast peers $B_s = \frac{B_f}{k}$ and compute the download rate r for a scenario where $N_f = N_s = \frac{N}{2}$, we obtain:

$$r = \frac{1}{N}(\frac{N}{2}B_f + \frac{N}{2}\frac{B_f}{k}) = \frac{k+1}{2k}B_f$$

We now relax the assumptions on the distribution of fast and slow nodes. If the number of fast peers is N_f, then the number of slow peers is $N_s = N - N_f$. Again the upload bandwidth of the slow peers is a fraction k of the upload bandwidth of the fast peers. The optimal download rate is then:

$$\begin{aligned} r &= \frac{1}{N}(N_f B_f + (N - N_f)\frac{B_f}{k}) \\ &= (\frac{N_f}{N} + \frac{1}{k}(1 - \frac{N_f}{N}))B_f \end{aligned} \quad (1)$$

If slow peers do not serve any content, i.e., $k \to \infty$, then Equation (1) becomes:

$$\lim_{k\to\infty}(\frac{N_f}{N} + \frac{1}{k}(1 - \frac{N_f}{N}))B_f = B_f\frac{N_f}{N}$$

In a scenario where $N_f = N_s$ this leads to a binary tree where the slow nodes are the leaves and the fast nodes are the inner nodes, each serving two other nodes at rate $r = \frac{B_f}{2}$ (as studied in [11]).

In the rest of the paper, unless explicitly mentioned, we consider equal populations of slow and fast peers ($N_s = N_f$).

4 A Study of Three Architectures

We now study and compare three different architectures that provide a uniform download rate to all peers.

4.1 Linear Chain Architecture

The first architecture considered in this paper forks several independent chains of peers that distribute content in parallel. The chains are constructed in three phases.

Phase 1 - Growing phase. The objective of the growing phase is to serve several peers (say m) in parallel starting from a single source. Obviously, such an expansion can only be achieved by fast peers, as they have more upload capacity than the target download rate r. Using this free capacity allows us to build the service capacity mr necessary to serve m peers in parallel.

Informally, the growing phase proceeds as follows. The first fast node (the source) starts a chain by serving one other fast peer with rate r. The remaining bandwidth $B_f - r$ will be used further down the chain. The second fast peer again serves another fast peer with rate r, which also leaves it with $B_f - r$ remaining bandwidth. This process continues until the sum of the remaining bandwidths of the first p fast nodes is sufficient to serve another peer, i.e., $p(B_f - r) \geq r$. Given that $B_s = \frac{1}{k}B_f$, p can be computed as:

$$p = \left\lceil \frac{k+1}{k-1} \right\rceil$$

In the formula above, depending on the value of k, some bandwidth may be lost in the integer conversion. This can be avoided by expanding to k nodes at once. The number of peers p_k necessary for this expansion can be computed by solving $p_k(B_f - r) = r(k-1)$, which gives:

$$p_k = k + 1$$

In the rest of the paper, we shall assume expansions to k chains using p_k peers (instead of 2 chains using p peers). Each fast peer can in turn fork another k chains with the help of $p_k - 1$ other fast peers. By repeating this process, the number of chains can be multiplied by k every iteration. Each expansion obviously requires p_k units of time. Examples with $k = 2$ ($r = \frac{3}{4}B_f$) and $k = 4$ ($r = \frac{5}{8}B_f$) are shown in Figures 1, 2, and 3. It is important to note that the peers are organized as a directed acyclic graph (DAG).

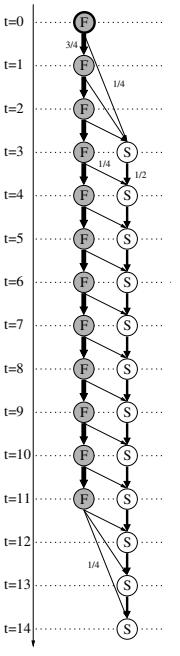


Fig. 1. Linear chains with one expansion step and $k = 2$ ($B_s = \frac{B_f}{2}$)

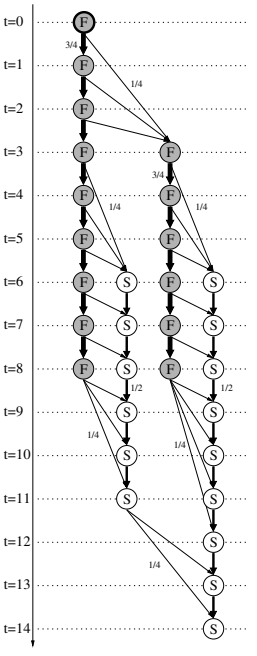


Fig. 2. Linear chains with two expansion steps and $k = 2$ ($B_s = \frac{B_f}{2}$)

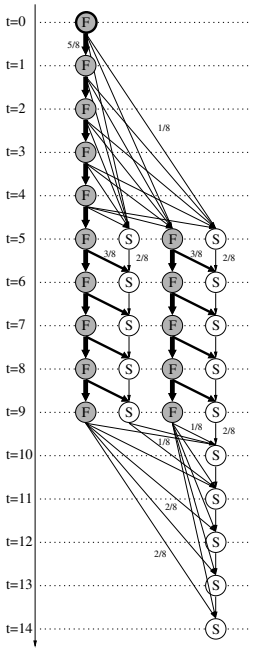


Fig. 3. Linear chains with one expansion step and $k = 4$ ($B_s = \frac{B_f}{4}$)

Phase 2 - Parallel phase. The parallel phase starts when the growing phase has finished its expansion to m peers. It constructs two sets of $\frac{m}{2}$ linear chains, composed respectively of fast and slow peers. Each chain of slow peers is combined with a chain of fast peers. A slow peer serves its successor at rate B_f/k. A fast peer serves its successor at rate r and the next slow peer in the companion chain at rate $B_f - r$. Thus, each peer is served at rate r. Phase 2 proceeds until all fast peers are being served (see Figures 1, 2, and 3).

Phase 3 - Shrinking phase. In the last phase, we are left with a set of slow peers to serve at rate r. As a slow peer cannot serve another peer by itself, the bandwidth of several peers must be combined, which leads to shrinking down the number of parallel chains. This phase is almost symmetrical to the growing phase, in that we can serve p_k slow peers from each set of k chains. We repeat this process until all slow peers have been served (see Figures 1, 2, and 3).

Analysis. We can easily notice that delays of $\delta_T = k$ are encountered during the growing phase. The case of the shrinking phase is more subtle, as δ_T grows larger if we keep it perfectly symmetric to the growing phase. By allowing some asymmetry, we can both bound the delays by the same value $\delta_T = k$ and reduce the total length of the shrinking phase.

We now compute the number of peers that can be served within a given time interval. After p_k steps, k peers can start again another chain. If we define s as the number of expansion steps, we can calculate the number of peers in the first phase N_1 to be:

$$N_1 = \sum_{i=0}^{s-1} k^i p_k = p_k \frac{k^s - 1}{k - 1}$$

The shrinking phase is built in a symmetric manner. Therefore the number of nodes N_3 in the third phase is the same as in the growing phase: $N_3 = N_1$. Given the constraint that $N_1 + N_3 \leq N$, the maximal value of s is:

$$s_{max} = log_k \left(N \frac{k-1}{2p_k} + 1 \right)$$

The number of nodes N_2 that can be served in phase 2 in a given time interval T is:

$$N_2 = k^s (T - 2sp_k + 1)$$

Indeed, there are k^s parallel nodes and phase 2 lasts for the given time interval minus the duration of the growing and shrinking phases. The number of peers served in a time interval T with s growing steps ($1 \leq s \leq \lfloor s_{max} \rfloor$) is then:

$$N(T, s, k) = 2p_k \frac{k^s - 1}{k - 1} + k^s (T - 2sp_k + 1)$$

We observe that the number of peers served in a given time interval grows with s, thus producing more efficient content distribution architectures (compare $N(14, 1, 2) = 24$ in Figure 1 and $N(14, 2, 2) = 30$ in Figure 3).

Solving the equation for T gives the number of units of time necessary to serve N peers:

$$T(N, s, k) = \frac{N(k-1) - 2p_k(k^s - 1)}{k^s (k-1)} + 2sp_k - 1 \tag{2}$$

Assuming that the content is split into chunks, the total download time for the complete file is then $1 + \frac{1}{C} T(N, s, k)$, i.e., the time necessary to transmit the whole file at rate r plus the propagation time of the chunks through the content distribution network. Using Equation (2) leads to:

$$T(N, s, k, C) = 1 + \frac{1}{C} \left(\frac{N(k-1) - 2p_k(k^s - 1)}{k^s (k-1)} + 2sp_k - 1 \right)$$

Figure 4 shows the time necessary to complete the download with the linear chain architecture for different values of k and C. We observe that performance improves with larger numbers of chunks, because all peers can be active most of the time. In contrast, with few chunks only a fraction of the peers will be uploading at any point in time, while the others have either already forwarded the entire file or not yet received a single chunk. Therefore, the value of k, which

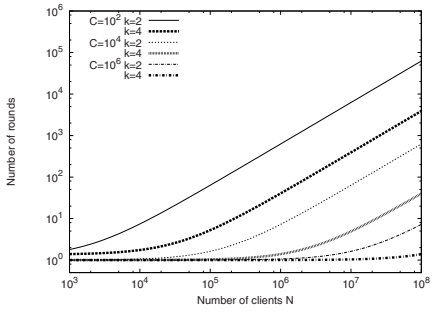


Fig. 4. Download time of the linear chain architecture for different values of k and C ($s = 4$)

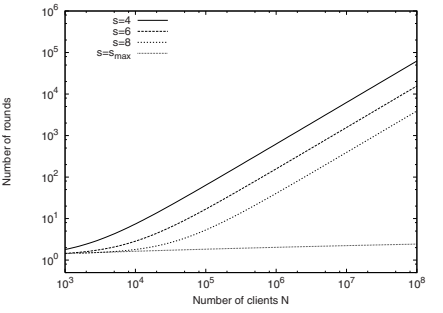


Fig. 5. Download time of the linear chain architecture for different values of s ($k = 2$, $C = 10^2$)

influences the depth of the content distribution architecture, has more impact on performance when the number of chunks is small. We notice indeed that the download times start degrading earlier with small values of k because they yield deeper architectures.

Figure 5 compares the download times for different values of s (the value s_{max} corresponds to the maximal number of expansion possible with the given peer population). As expected, performance improves with higher values of s because they produce flatter architectures. The optimal value s_{max} exhibits extremely good scalability.

4.2 Mesh Architecture

The linear chains architecture can be improved in several ways if we allow peers to be organized as a directed graph with cycles. We can reduce the duration of the growing phase and thus the length of the paths (and consequently the latency); we can simplify network management by only using connections with identical bandwidth capacities; and we can limit the size of buffers at each peer to a constant value.

The resulting mesh architecture is shown in Figure 6 (for $k = 2$ and one expansion step) and Figure 7 (for a general value of k and two expansion steps). A node does not only receive data from its parent, but also from its siblings. The source has $2k$ fast peers as children and sends data at rate $\frac{B_f}{2k}$ to each of them; the remaining bandwidth $\frac{B_f}{2}$ is provided by their siblings. The first-level fast peers together serve k^2 children with their remaining bandwidth of $\frac{B_f}{2}$; again, the remaining bandwidth $\frac{k-1}{2k}B_f$ is provided by the siblings. Second-level peers have enough bandwidth to completely serve k^2 children. Each third-level child can in turn expand to k^2 peers in three steps.

As in the previous architecture, one can build linear chains after the expansion phase before reducing the architecture to one peer. The shrinking phase is symmetric to the growing phase, as shown in Figure 6.

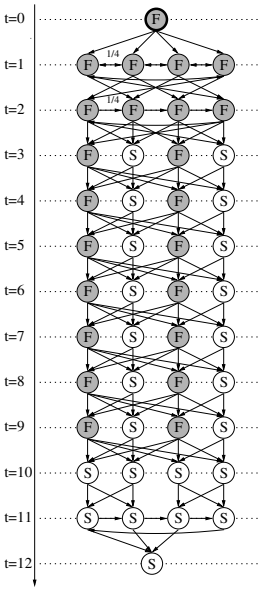


Fig. 6. Mesh with one expansion step and $k = 2$ ($B_s = \frac{B_f}{2}$)

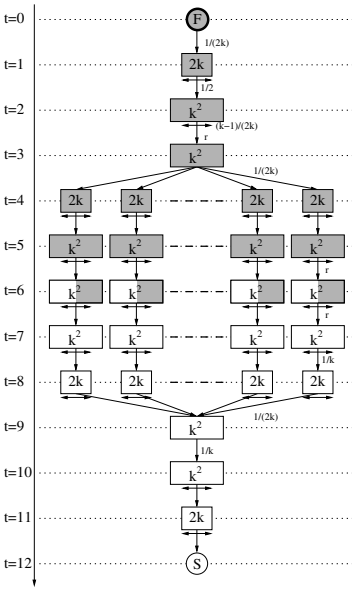


Fig. 7. Mesh with two expansion steps and any k ($B_s = \frac{B_f}{k}$)

Using only connections with identical rate $\frac{B_f}{2k}$ simplifies significantly the management of the architecture. The throughput is controlled by the source and peers only differ in their number of outgoing connections: the outdegree is always $2k$ for fast nodes and 2 for slow nodes. All peers have an indegree of $k + 1$.

Analysis. One can note in Figure 6 that the first level fast peers receive chunks from the source at $t = 1$ and from their sibling at $t = 2$; similarly, second level peers receive chunks at $t = 2$ and $t = 3$; on the third level, all chunks are received simultaneously at $t = 3$. A similar observation can be made with the shrinking phase and it follows that constant delays of $\delta_T = 1$ are encountered in this content distribution architecture.

For computing the number of nodes which can be served in time T we again analyze the three phases. As we have seen, a fast peer can expand to k^2 peers in three units of time with the help of $2k + k^2$ other fast peers. If we define s to be the number of expansion steps, then the number of peers served in the first phase is:

$$N_1 = 1 + (2k + 2k^2) \sum_{i=0}^{s-1} k^{2i} = 1 + 2k \frac{k^{2s} - 1}{k - 1}$$

The shrinking phase again is symmetric in the number of nodes so the number of nodes in the third phase N_3 is equal to N_1, thus $N_3 = N_1$. Given the constraint that $N_1 + N_3 \leq N$ we can compute the maximal value of s:

$$s_{max} = \frac{1}{2} log_k \left(\frac{(N-2)(k-1)}{4k} + 1 \right)$$

In phase 2, $k^2 k^{2(s-1)}$ parallel nodes can be served in the remaining time $T - 6s - 1$. In total the number of peers served within T units of time for a given number of s expansion steps $1 \leq s \leq \lfloor s_{max} \rfloor$ is then:

$$N(T, s, k) = 2 + 4k \frac{k^{2s} - 1}{k - 1} + k^{2s}(T - 6s - 1)$$

Solving the equation for T and introducing the number of chunks C gives:

$$T(N, s, k, C) = 1 + \frac{1}{C} \left(\frac{1}{k^{2s}} \left(N - 2 - 4k \frac{k^{2s} - 1}{k - 1} \right) + 6s + 1 \right)$$

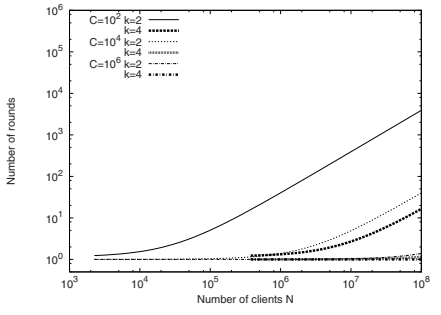


Fig. 8. Download time of the mesh architecture for different values of C ($k = 2$, $s = 4$)

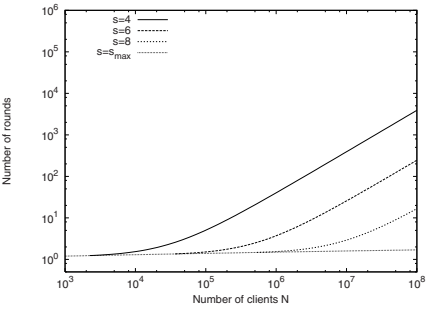


Fig. 9. Download time of the mesh architecture for different values of s ($k = 2$, $C = 10^2$)

Figure 8 shows the time necessary to complete the download with the use of the mesh architecture for different values of C and k. As expected, the download times follow the same general shape as for the linear chains architecture in Figure 4 but performance is significantly improved due to the faster expansion of the mesh architecture. We can observe in Figure 9 that a higher number of expansion steps s also produces flatter architectures and therefore reduces the download time. The maximal expansion for a given peer population s_{max} yields the best download times, which is almost constant, independent of the population size.

4.3 Parallel Trees

The third architecture studied in this paper consists in constructing multiple trees spanning all the nodes and sending a separate part of the content in parallel to each tree similarly to SplitStream [5] and PTree^k [11] (as $N_f = N_s$, we shall

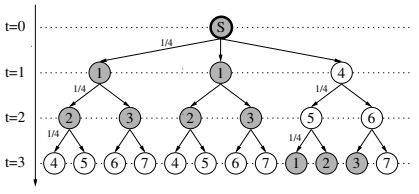


Fig. 10. Parallel trees with $N = 8$ and $k = 2$ ($B_s = \frac{B_f}{2}$)

use binary trees). If we construct $k+1$ trees that distribute content at rate $\frac{B_f}{2k}$, then every peer will receive data at the same uniform rate r.

We construct parallel trees by placing each fast peer (except the source) as interior node in k trees. Fast nodes will thus serve $2k$ other peers at rate $\frac{B_f}{2k}k$ (i.e., at aggregate rate B_f). The slow nodes are placed as interior nodes in a single tree and must thus serve two other nodes at rate $\frac{B_f}{2k}$ (i.e., at aggregate rate $\frac{B_f}{k}$). As the number of leaves in a complete binary tree is equal to the number of interior nodes plus one and the source is a fast node, the constraint $N_f = N_s$ is met. Figure 10 illustrates the parallel tree architecture (peers are numbered for clarity). Note that every peer except the source appears in all trees.

Analysis. We first need to determine the depth d of the trees. At each level i in the tree, we have 2^i nodes (the root is at level 0). Thus, the number of nodes in a binary tree of depth d is $\sum_{i=0}^{d} 2^i = 2^{d+1} - 1$. Considering the special role of the source, the $N-1$ remaining nodes can be placed in parallel trees of depth $d = \lfloor log_2(N-1) \rfloor$.

It follows from the construction of the trees that delays of $\delta_T = \lfloor log_2(N-1) \rfloor$ are encountered in this content distribution architecture. Delays grow with the number of peers, in contrast to the other architectures studied in this paper.

The number of nodes that can be served by the parallel tree architecture in a given time interval T can be computed as follows (the first term represents the source):

$$N(T) = 1 + \sum_{i=0}^{T-1} 2^i = 2^T$$

Solving this equation to T and introducing the number of chunks C leads to the time used to distribute a file to all nodes:

$$T(C, N) = 1 + \frac{1}{C} \lceil log_2 N \rceil$$

Figure 11 shows the time necessary to complete the download with the parallel tree architecture for two values of C (improvements become unnoticeable when C grows larger). As the download time is a function of the depth of the trees, which increases logarithmically with the number of peers, performance degrades only slowly with the population size.

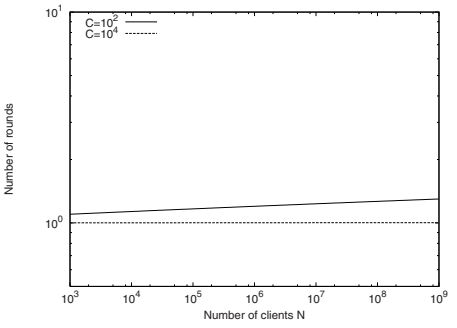


Fig. 11. Download time for the parallel trees architecture for different values of C

5 Comparative Analysis

In this section we compare the three architectures presented in this paper with the linear chain architecture analyzed in [13] (referred to as *Linear*). In contrast to our architectures, in *Linear* the peers have symmetric bandwidth capacities. The peers are organized in separate chains according to their bandwidth capacity and there is no cooperation between fast and slow nodes. Fast peers can therefore finish the download faster.

As we can see in Figure 12, this difference leads to a stepwise function with the fast nodes completing their download faster than the slow nodes ($N_f = N_s$). In contrast, the uniform architectures all scale well and yield an almost constant download rate independent of the population size. As expected, uniform linear chains are less efficient than the mesh and parallel tree architectures due to the longer paths.

In Figure 13 we can observe that with a smaller difference between fast and slow peers (lower value of k) the download time of *Linear* grows, whereas it

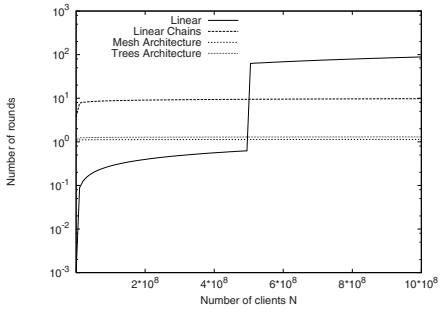


Fig. 12. Download time for different architectures with $k = 100$, $C = 100$ and $s = s_{max}$. *Linear* shows the completion times for a population of 10^9 peers with symmetric bandwidth.

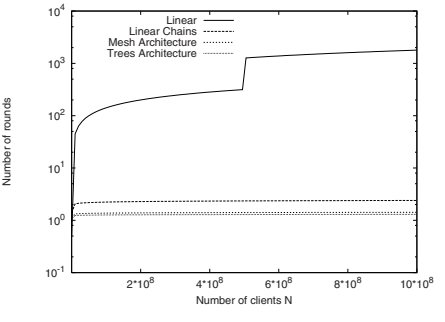


Fig. 13. Download time for different architectures with $k = 4$, $C = 100$ and $s = s_{max}$. *Linear* shows the completion times for a population of 10^9 peers with symmetric bandwidth.

decreases for the linear chains and the mesh architecture (remember that a unit of time is defined as a function of the uniform rate r). We can further see that the mesh architecture performs slightly better than parallel trees in Figure 12, unlike in Figure 13. This is due to the fact that the mesh architecture expands as a function of k^{2s} whereas the expansion of parallel trees does not depend on k. Thus the mesh will grow faster when k is large. Higher values of C do not produce interesting results as the difference between the various architectures quickly becomes unnoticeable.

6 Conclusion

Content distribution is an important problem for many distributed applications deployed in the Internet. Cooperative techniques based on peer-to-peer networks offer the technical capabilities to quickly and efficiently distribute large or critical content to huge populations of clients. When dealing with streaming or time-sensitive data, the content must be provided at a rate which is sufficient for its intended purpose (e.g., displaying a streaming movie).

In this paper, we have studied the problem of providing uniform download rates to a population of peers with asymmetric and heterogeneous bandwidth capacities. The architectures that best achieve this goal among those studied in the paper are the mesh and the parallel tree, but the latter requires peers to buffer data for a duration proportional to the depth of the trees. As the number of chunks grows, i.e., when the stream duration becomes very long, the differences between all the architectures become insignificant.

Although we only focused on analytical models for simple content distribution architectures, we believe that our analysis provides some important insights as how to set up peer-to-peer networks for distributing streaming data. It can also guide the design of cooperative applications that organize the nodes in a more dynamic manner than chains or trees. In particular, the system needs to build up upload capacity as fast as possible (which corresponds to maximizing the number of expansion steps) and the content should be partitioned into a large number of chunks (but not too many chunks as each one adds some coordination and connection overhead). By properly combining high and low capacity nodes, one can provide a high quality of service to every peer and even out their differences in a truly cooperative manner.

Acknowledgments. This work is supported in part by the Swiss National Foundation Grant 102819.

References

1. Chu, Y., Rao, S., Zhang, H.: A case for end system multicast. In: Proceedings of ACM Sigmetrics. (2000)
2. Hefeeda, M., Habib, A., Boyan, B., Xu, D., Bhargava, B.: PROMISE: peer-to-peer media streaming using CollectCast. Technical Report CS-TR 03-016, Purdue University (2003)

3. Jiang, X., Dong, Y., Xu, D., Bhargava, B.: Gnustream: A P2P media streaming system prototype. In: Proceedings of the International Conference on Multimedia and Expo (ICME),. (2003)
4. Chawathe, Y.: Scattercast: An adaptable broadcast distribution framework. Multimedia Systems **9** (2003) 104–118
5. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-bandwidth multicast in a cooperative environment. In: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). (2003)
6. Jannotti, J., Gifford, D., Johnson, K.L., Kaashoek, M.F., O'Toole, J.W.: Overcast: Reliable multicasting with an overlay network. In: Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI). (2000)
7. Pendarakis, D., Shi, S., Verma, D., Waldvogel, M.: Almi: An application level multicast infrastructure. In: Proceedings of USITS. (2001)
8. Padmanabhan, V., Wang, H., Chou, P.: Resilient peer-to-peer streaming. In: Proceedings of IEEE ICNP. (2003)
9. Rao, S., Padmanabhan, V., Seshan, S., Zhang, H.: The impact of heterogeneous bandwidth constraints on dht-based multicast protocols. In: Proceedings of the 4th International Workshop on P2P Systems (IPTPS). (2005)
10. Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: Scribe: a large-scale and decentralized application-level Multicast infrastructure. IEEE Journal on Selected Areas in Communications **20** (2003) 1489–1499
11. Biersack, E., Rodriguez, P., Felber, P.: Performance analysis of peer-to-peer networks for file distribution. In: Proceedings of the 5th International Workshop on Quality of future Internet Services (QofIS'04). (2004) 1–10
12. Yang, X., de Veciana, G.: Service capacity of peer-to-peer networks. In: Proceedings of IEEE INFOCOM. (2004)
13. Carra, D., Cigno, R.L., Biersack, E.: Introducing heterogeneity in performance analysis of p2p networks for file distribution. Technical Report DIT-04-113, University of Trento (2004)

JXTA Messaging: Analysis of Feature-Performance Tradeoffs and Implications for System Design*

Emir Halepovic[1], Ralph Deters[2], and Bernard Traversat[3]

[1] Department of Computer Science, University of Calgary,
Calgary AB T2N 1N4, Canada
emirh@cpsc.ucalgary.ca
[2] Department of Computer Science, University of Saskatchewan,
Saskatoon SK S7H 5L6, Canada
deters@cs.usask.ca
[3] Project JXTA, Sun Microsystems Inc,
Menlo Park CA 94025, USA
bernard.traversat@sun.com

Abstract. With the rise of Peer-to-Peer and Grid infrastructures, there is a renewed interest in messaging systems. Among the numerous messaging solutions for large loosely coupled distributed environments, the communication and messaging system of the JXTA peer-to-peer platform is the richest in features. This paper presents a comprehensive performance analysis of the JXTA messaging system and its application to the system design scenarios. Several application-layer messaging abstractions are analyzed and compared. The results reveal the limiting factors and behaviors with respect to workload, network distance, peer group size, and message relays. The tradeoffs between features and performance are observed and discussed in the form of recommendations to developers dealing with common system design cases.

1 Introduction

Peer-to-peer (P2P) systems are commonly characterized by decentralized control, high autonomy of participating nodes, shared hardware and software resources, potential to scale and ad-hoc connectivity. Most P2P applications have very specific approaches for dealing with peer and resource discovery, communication and resource sharing and tend to focus on a single purpose. In addition, the majority of file-sharing, collaboration, and computation P2P systems cannot interoperate.

The goal of Project JXTA is to formulate, develop and standardize the core P2P operations and protocols [13]. The JXTA platform defines protocols for peer discovery, messaging, identification, group organization, etc., which are necessary for all P2P applications, and provides the implementation in several programming languages. JXTA aims to leverage and build upon the traditional communication protocols and to provide universal components for building any type of P2P applications [10, 16].

* This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1097 – 1114, 2005.

JXTA provides several messaging abstractions with different features targeted to satisfy specific requirements. The questions arising are whether the features, such as reliability and security, introduce significant performance overhead that outweighs the benefits for a particular system, and how to optimize performance once the messaging components have been chosen.

This paper focuses on the messaging subsystem of the JXTA platform with the goal of answering the aforementioned questions. We present the methodology and analysis that can be used by developers during the system design, as well as measurements that show the tradeoffs between features and performance of JXTA components that can be directly applied to the most common P2P systems.

The presented work extends the earlier JXTA evaluations by analyzing the messaging performance over a WAN, scalability with multiple senders/receivers and overhead of indirect communication. New higher-level JXTA services are added to the evaluation and compared to the core JXTA pipes. We evaluate JXTA versions 1.0, 2.0 and 2.2 and their implementations for Java 2 Standard Edition (J2SE). The evaluation is based on benchmarking consistent with the proposed JXTA Performance Model [2], which outlines methods, components and metrics of interest.

The key results include the identification of performance penalties introduced by security, propagation and reliability features. Further, the scalability results show dependency on the payload size and the metric observed, with throughput scaling well for small payloads, and poorly for large payloads. The measured costs of indirect messaging reveal the cases when messaging over HTTP is faster than over TCP, as well as several other unintuitive results.

The most important implications of the findings are on the design of the messaging subsystem for new JXTA applications. The observed performance results reveal the ways to achieve optimal design for the specific messaging objectives, and we discuss these issues for reliability, security and message propagation in a JXTA system.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 explains the JXTA messaging architecture and components. Experimental methodology and environment are outlined in Section 4. Section 5 presents the analysis of performance results for direct messaging, Section 6 for message propagation and Section 7 for indirect relayed messaging. The discussion based on the results follows in Section 8. The conclusions and further work are outlined in Section 9.

2 Related Work

With the emergence of the Grid computing systems and multi-purpose P2P middleware, such as JXTA, the messaging layer becomes a critical component. A generic but efficient messaging layer allows for rich interaction between peers and provides a powerful basis for design of application-specific and complex communication protocols. Several proposed solutions indicate the importance of the communication layer and the significant role the messaging subsystem will have in future P2P and Grid systems. Message Oriented Middleware (MOM), such as the enterprise server-based JMS [5] is intended for small-scale deployments and it leverages Java J2EE. Other solutions, such as NaradaBroker [11] and PAWN [9] directly leverage or support JXTA.

Alternative messaging solutions were compared to early versions of JXTA and shown to be superior, such as the multi-ring based messaging [6] and TIBCO Rendezvous [15]. Due to the complexity of the JXTA messaging abstractions [12] and the provisions of security, reliability and propagation features, it is not surprising to find simpler and more efficient messaging solutions. However, these efforts and results additionally motivate the comprehensive study of the JXTA messaging subsystem and its evolution and improvements through major versions.

Performance measurements of JXTA are available in several forms. The targeted evaluations of the JXTA messaging components are based on the JXTA community Bench project [7]. The Bench project tracks progress through major and minor JXTA releases, and guide developers in optimizing the implementation. The results are obtained from a single controlled environment, an isolated LAN connecting several high-performance computers. We use a non-dedicated LAN, WAN and mid- to high-end personal computers as the more common P2P application environment.

More detailed evaluations show the JXTA pipe behavior constrained to LAN environment and one-to-one (1-1) communication [2, 4]. The results indicate the message round-trip time independence of the payload size up to 100 KB for non-secure pipes, consistent with findings in [12]. A change in the effect of the message composition in terms of number and size of XML elements is also noted between JXTA versions. While more complex messages took longer time to parse in JXTA 1.0, improvements in JXTA 2.0 included near-constant time to parse messages of different complexity for unicast and propagate pipes, and even decreasing parsing time with increasing number of elements for secure pipe for the same overall payload size. We expand the evaluation to the multiple sender and receiver scenarios in peer groups and newer JXTA version, as suggested in earlier works.

Additional important results show that JXTA performance can be tuned by swapping XML parsers and that hardware configuration and CPU power can significantly affect the performance of the JXTA message creation and processing [12]. It has also been reported that JXTA performs comparably on different operating systems, e.g. Linux vs. Windows, with the interesting trend of Linux performing worse for smaller messages [14]. Lower-level endpoint service has been evaluated with unicast pipe and JXTA socket on high-speed networks in 1-1 communication and compared to plain sockets [1]. It is shown that both unicast pipe and JXTA socket have comparable throughput to plain sockets that converges for large message sizes, but that the latency of JXTA is 5 to 20 times higher than that of plain sockets. Our study focuses on the comparison of the standard application-layer messaging services inside JXTA with the goal of finding optimal configuration for different system requirements, rather than measuring the direct effect of hardware or low-level implementation components.

The unintuitive results of the measurements obtained from the 1-1 communication on a LAN further motivate the in-depth evaluation expanded to the WAN, multiple sender and receiver scenarios, as well as addition of the new and feature-rich messaging services in JXTA, as included in this study.

All of the available results, including this study, are obtained by benchmarking, which is still the best method for evaluating the JXTA platform since the JXTA protocol specification [8] does not clearly specify any algorithms suitable for analysis or simulation. The performance of JXTA is dependent on the details of its implementation, and the Java-based reference implementation is updated first with the latest de-

sign decisions. Project JXTA features no public design documentation suitable for performance evaluation either. Therefore, benchmarking of its reference implementation is presently the best way to learn about the performance of JXTA.

3 JXTA Messaging Architecture

JXTA pipes are a fundamental abstraction used for inter-peer communication. JXTA peers pass messages through pipes, virtual channels that consist of input and output ends. Peers bind to one end of the pipe, and when both ends are bound, messages can be passed. Pipes are not tied to a physical location, IP address or port. Instead, a pipe has a unique identifier (ID), so peers retain their pipes as their network location changes. At runtime, a pipe end is resolved to the current endpoint IP address and port.

The JXTA core protocol specification [8] defines three kinds of pipes operating in two modes, referred to as core pipes. The two operation modes are unicast and propagate. Unicast and secure pipes serve for 1-1 communication, connecting two peers in unicast mode. Propagate pipes connect one sender peer to many receiving peers and they operate in propagate mode (1-M). Core pipes are asynchronous and unreliable. The reliability of JXTA pipes refers to the message-level flow control. Messages can be dropped from the overflowing queues at sender or receiver ends. Only JXTA sockets in the analyzed JXTA versions handle this kind of message loss, making it a reliable messaging component. JXTA platform uses TCP and HTTP as lower-level protocols and their reliability between endpoints is still in effect.

Unicast pipes can be combined to achieve many-to-one (M-1) communication. From the receiver's perspective, a single operation is required to open the input end of the pipe. Multiple senders can connect to the same pipe and open their output ends for communication. Senders "see" the pipe as a one-to-one connection, and handling of multiple connections at the receiver's side is completely transparent to the programmer. A secure pipe also operates in unicast mode, with the additional security provided by the Transport Layer Security (TLS) layer [8].

A propagate pipe is used for one-to-many communication; leveraging either IP multicast on the subnet or rendezvous peers [4] for message propagation. A sender commonly opens the output end of the pipe first and starts sending, whereas receivers connect to the propagate pipe by opening their input ends to receive any messages transmitted by the sender.

Two new and useful messaging abstractions are added to JXTA, the bi-directional pipe and the JXTA socket. Both are included in this performance evaluation as non-core JXTA services that offer additional features on top of the core specification. The goal of analyzing the non-core messaging services is to find if any tradeoffs exist between the features and performance, as compared to the core pipes.

The purpose of the bi-directional pipe is to provide two-way communication within a single messaging object. A bi-directional pipe features an API that is similar to a well-known Java socket API. Under the abstraction layer, the connection is opened using a unicast pipe for one direction, and then the reverse direction link is established using an internal endpoint service [8], which is normally not directly used at the application layer. The usual pipe resolution is bypassed for the reverse direction. All connection code is still hidden under the single method invocation.

Similar to the core pipes, the use of bi-directional pipes assumes that one peer initially creates an input endpoint (server), to which another peer (client) will connect. The difference in the API requires a programmer to implement a multi-threaded server, unlike for core pipes that have a built-in handling of multiple connections.

A JXTA socket is an optimized bi-directional pipe. It uses a standard JXTA pipe service for the initial connection and endpoint addressing for the reverse connection, similarly to bi-directional pipe. The JXTA socket provides reliability directly and has no limit on the transferred data size; it transmits in chunks of 16 KB by default. JXTA socket provides the same API as the standard Java socket classes, and hides the underlying pipe implementation. This includes the methods for obtaining the input and output streams as the primary communication API. The JXTA socket directly allows byte-level access to the transmitted data, which offers the highest flexibility to the developer in designing a communication protocol. Therefore, JXTA messages are not the intended unit of data transmission over JXTA sockets at the application layer.

JXTA messages are XML documents composed of ordered elements [8]. The elements are name-value pairs, and they can carry any type of payload. JXTA uses source-based routing and each message carries its routing information as a sequence of peers to traverse. The peers along the path may update this information.

A relay peer's main purpose is to cache routes and pass messages between peers that cannot establish a direct connection. As the usage of firewalls and NAT increases in the Internet, connectivity becomes more challenging and the role of relays becomes more important. Relays are potentially exposed to large amounts of traffic and their performance is important for the entire peer group.

The messaging architecture of JXTA is fairly complex, involving the XML parser and several layers of abstraction (services) that participate in the message construction, transmission and processing [12]. It is expected that the layers of abstraction and indirect messaging add significant overhead and affect the efficiency and performance of the whole messaging subsystem. Table 1 summarizes the main characteristics of the JXTA messaging components.

Table 1. Summary of the main features of the JXTA messaging components

Pipe	Ends	Reliable	Direction	Other
Unicast	1-1	No	1-way	Base pipe
Secure	1-1	No	1-way	Encrypted messaging
Propagate	1-M	No	1-way	Can use UDP for multicast
Bi-directional	1-1	No	2-way	Socket-like API
JXTA socket	1-1	Yes	2-way	Byte-level socket API

4 Methodology

This study is conducted by setting up a JXTA test-bed consisting of personal computers on and off campus of the University of Saskatchewan in Saskatoon, Canada. The benchmark applications start up as peers that connect into a JXTA peer group. A

series of experiments is then performed to evaluate the messaging components under a range of conditions. The important details of the experimental setup are outlined in this section and all evaluation parameters are summarized in Table 2.

The hardware environment is a pool of five computers equipped with 800 MHz CPU and 512 MB of RAM each, used for sending and relay peers. The receiver peers in a multiple sender scenarios run on two 2.5 GHz PCs with 1 GB of RAM. Most other studies used only state of the art machines, but we believe that inclusion of somewhat inferior hardware better reflects the typical user base of common P2P applications and allows for repeatable experiments.

Two network environments are used throughout this study, a LAN and a WAN. All LAN measurements are taken with JXTA peers running on the campus 100 Mbps LAN, inside a subnet. The ping tool measures the average transmission round-trip time (RTT) at less than 1 ms on this LAN. The high-bandwidth WAN environment includes peers inside the campus LAN and peers connected to the Internet through a 100 Mbps router and a cable modem. The ping RTT averages on a WAN were 34 to 39 ms at different times throughout the different benchmarking sessions.

The JXTA environment assumes the following settings and properties. The generic JXTA peer group (NetPeerGroup) is tested, but it is constrained to the test peers by disallowing external rendezvous or peer connections. The main underlying transport is TCP. The peer group size of up to 8 edge peers is used due to the complexity of deploying larger peer groups consisting of standalone independent peers. Nevertheless, we believe that the used group sizes appropriately reflect the majority of P2P communication scenarios, such as the limits on active incoming and outgoing connections of common file-sharing, chat and collaboration applications.

The analysis is based on two metrics commonly used in communication systems performance evaluation, RTT and throughput. Since the unit of data transfer in JXTA is a message, we use the message RTT as a bi-directional transfer metric, analogous to the packet RTT in TCP.

We define message RTT as the time elapsed between the instant a message is sent and the instant an acknowledgement message (ACK) is received by the sender. In this paper, RTT refers to the message RTT as defined above. The JXTA socket benchmarks use a pair of character strings in place of genuine JXTA messages, but they will still be referred to as “messages”. The first string contains the message ID, and the second stores the payload of desired size. The ACK is composed of the single string containing the message ID. The message load a pipe can sustain is measured by the message throughput. Message throughput refers to the maximum number of messages a pipe can reliably transmit in a unit of time.

Smooth traffic is used for throughput measurements, which assumes a uniform message-sending rate throughout the test run. Peers use one TCP or HTTP connection for the complete workload transfers over unicast, secure and propagate pipes and another connection for the ACK delivery over a unicast pipe. The non-core services use the reverse link of the same connection for ACK transfer. If any modifications to the general testing environment or diversions from the stated constraints are made, they are noted with the corresponding results in the following sections.

Table 2. Experimental parameters and configurations

Hardware	5 (800 MHz, 512 MB) PCs, 2 (2.5 GHZ, 1 GB) PCs
Software	MS Windows 2000 Professional, Java J2SE 1.4.1 JVM
Network	100 Mbps LAN, 4 Mbps/512 kbps WAN via cable modem
JXTA	J2SE-JXTA versions 1.0, 2.0, 2.2
Peer groups	2-8 edge peers, 1-2 relays, 1 rendezvous peer
Workload	1, 10, 100 KB, 1 MB message sizes
Sample size	10,000 messages (1,000 warm-up)
Metrics	Message RTT and Throughput

5 Direct Messaging

This section presents the direct messaging RTT and throughput results and analysis with respect to the pipe or service type, message size, network distance and the number of senders and receivers.

The RTT is measured for four message sizes, 1 KB, 10 KB, 100 KB and 1 MB, which cover most P2P application types, from simple chat applications to file sharing and content storage, with the assumption that larger files will be broken down into smaller pieces to enable faster and parallel transfers. The bi-directional pipe does not have a measurement for 1 MB message size due to the imposed size limit by the JXTA implementation. Furthermore, it is expected that the limit will be lowered to 64 KB for all pipes, except JXTA sockets, in the newer JXTA versions (2.3.x) to increase fairness in resource usage among peers, especially relays [1]. JXTA core pipes and services are evaluated in 1-1, 1-M and M-1 communication.

5.1 Message RTT in 1-1 Communication

The objective of this test is to understand how different messaging components behave over the range of message sizes, using a single payload element.

The evaluation of the core pipe performance on a LAN was conducted in [3] for JXTA version 1.0 and 2.0. Fig. 1 shows the same measurements collected for JXTA 2.2 and includes the non-core services. It shows that non-secure pipes, now including the bi-directional pipe, still exhibit a negligible increase in RTT for message sizes up to 100 KB and then linearly slow down for larger messages. This indicates the stronger effect of the underlying implementation, based on unicast pipe, rather than payload size on RTT (up to 100 KB). It is also noted that secure pipe and JXTA socket perform significantly better than unicast for smaller message sizes. They both have a sharper RTT increase at 10 KB size with secure pipe finally performing the worst of all for 1 MB messages.

The current evaluation of JXTA messaging on the WAN shows more results that are interesting. The average RTT is shown in Fig. 2. The RTT follows the same general patterns as on the LAN, with the expected shift due to the network distance and bandwidth limits. What is more interesting is that all types of pipes have much closer RTT than on the LAN, which converges as the message size increases. The unicast

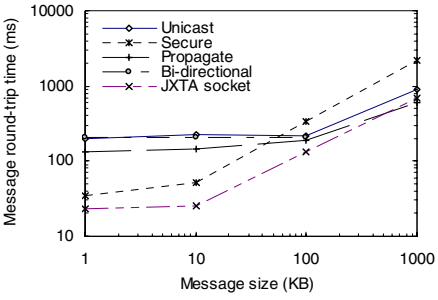


Fig. 1. Scaling of message RTT on LAN (log-log scale)

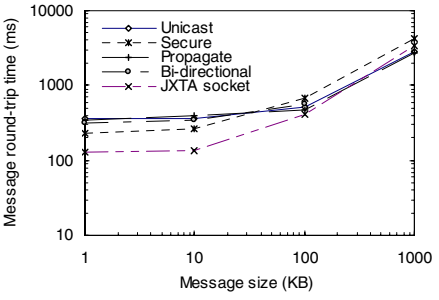


Fig. 2. Scaling of message RTT on WAN (log-log scale)

and propagate pipes have negligible difference in RTT due to the use of TCP, whereas UDP-based propagate pipe outperforms the unicast on the LAN by a factor of ten [3].

Overall, the observed results mean that for wide area deployments there is a smaller penalty in RTT performance with added features, such as security and transport reliability (bi-directional TCP, JXTA sockets). A small overhead for an additional feature is a desirable behavior for JXTA pipes. A pattern that emerges from the observed RTT is that the new non-core services show similarity to different core pipes.

The bi-directional pipe has the same pattern of behavior as unicast pipe, which is very favorable considering the added bi-directional feature. This is not surprising since the implementation is based on the unicast pipe. On a LAN, the bi-directional pipe yields results that are within 9%, and on a WAN within 15% of the unicast pipe. The RTT performance difference between the unicast and bi-directional pipe is therefore negligible for practical purposes.

On the other hand, JXTA socket behaves more like secure pipe, with shift in RTT due to the lack of security provisions. Higher sensitivity of the latter two pipes to the message size is due to the window-based flow control provisions, on TLS layer for secure pipe and on the service layer for JXTA socket. Finally, the newest JXTA socket service has the lowest RTT of all messaging components, up to an order of magnitude for small messages, with the added benefits of reliability and Java socket API.

5.2 Message Throughput in 1-1 Communication

It is already known that core pipes have limited message sending rates, which determines the 1-1 throughput as shown in [2] for LAN. This study expands the analysis to cover the effects of both network distance and message size on the sending rates of all messaging components.

Fig. 3 shows the throughput of the 1 and 10 KB smooth message streams on the LAN and WAN side-by-side, from which we can quantify the performance penalty of increased message size and network distance. It is also noticeable that the message size and network distance have different effects on different pipes.

Message size affects only the secure pipe on the LAN (note the same effect on RTT from 1 to 10 KB), whereas on the WAN it affects all except propagate pipe. In

fact, the propagate pipe has poor, but almost equal throughput regardless of the network distance or message size. We can also observe the relative penalty, expressed as a ratio of throughputs on the WAN and LAN for the same message size. For example, the relative penalty of WAN for JXTA socket is 2.29 for 1 KB message and 1.56 for 10 KB message. The bandwidth limit forces pipes to reduce the output rate over a WAN, which is most noticeable for unicast and bi-directional pipes for 10 KB messages, whose throughput is reduced by factors of 1.78 and 3.92, respectively.

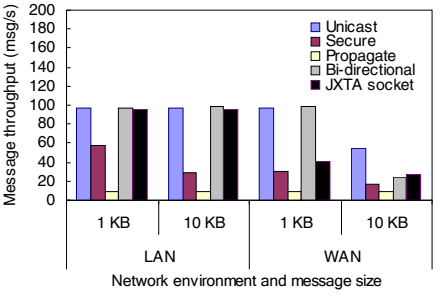


Fig. 3. Effect of message size and network distance on sending rates

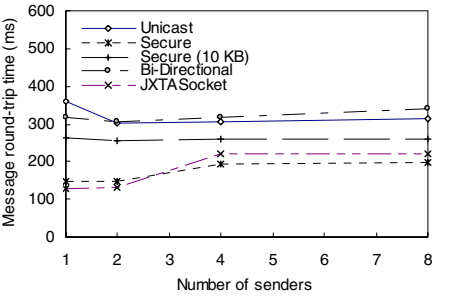


Fig. 4. Scaling of RTT with number of senders (1 KB, WAN)

The throughput observations on the LAN correspond to the RTT results. However, the throughput behavior of the secure pipe is an anomaly due to the implementation, which is to be updated in the future to allow the secure pipe to keep the same level of throughput for 10 KB messages as well. The propagate pipe has a limited throughput due to the rendezvous service implementation responsible for queuing and propagation of messages. Bandwidth appearing as a factor in a WAN test is logical and expected.

5.3 Message RTT in M-1 Communication

All JXTA pipes and sockets, except propagate pipe, are designed to allow simultaneous connections from multiple sending peers. The following results show how the pipes behave with up to 8 parallel senders.

Fig. 4 presents a comparison of the message RTT for pipes and JXTA socket transmitting 1 KB messages across a WAN. An additional line for secure pipe at 10 KB size is plotted to show the effect of the increased message size. All other pipes and JXTA socket record nearly the same RTT for 10 KB messages. Secure pipe is the only one noticeably affected by larger payload. The unicast and bi-directional pipes again follow the same pattern, and they both show lower latency for 2 than for 1 sender, as well as secure pipe for 10 KB message size. For unicast pipe, this may be an indication of thread management policy when parallel connections are active. For the other two pipes, the difference is too close to the measurement clock granularity of 10 ms to make any conclusion. Overall, the increasing number of senders has very little effect on the message RTT, as long as the bandwidth is not saturated (4 Mbps on the WAN).

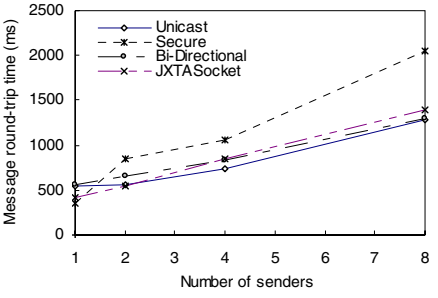


Fig. 5. Scaling of RTT with number of senders (100 KB, WAN)

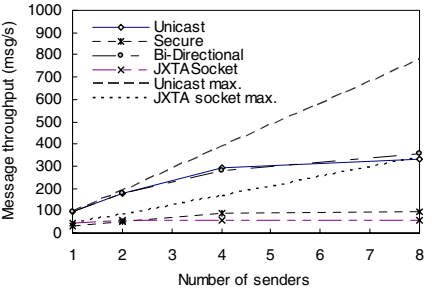


Fig. 6. Scaling of throughput with number of senders (1 KB)

Fig. 5 shows the RTT for 100 KB messages with an increasing number of senders across a WAN. The linear slopes are higher for all pipes and the similarity between unicast and bi-directional pipes is still visible. The effect of number of senders is pronounced for this message size. Here we see that JXTA socket is getting slower than unicast and bi-directional pipes with increasing number of senders.

5.4 Message Throughput in M-1 Communication

The throughput with multiple senders is investigated next for smooth streams of 1 and 10 KB messages across the WAN. The message throughput for 1 KB messages is shown in Fig. 6. The upper and lower dotted lines represent the maximum throughputs calculated by multiplying the single-sender rate with the number of senders for unicast pipe and JXTA socket, respectively. All messaging components show similar performance patterns on both LAN and WAN. Familiar pattern similarities are visible for unicast and bi-directional pipes, as well for secure and JXTA socket.

The behavior patterns do not appear for 10 KB message transfers (Fig. 7), where the secure pipe surprisingly shows the best throughput scaling with number of senders. The bi-directional pipe achieves less than half the throughput of the unicast pipe for 10 KB messages in 1-1 setup, but all other configurations yield more comparable performance. However, the reliability of the bi-directional pipe across the WAN connection is extremely low, reflected by high message loss (message queue overflows) if the sending rate is higher than the maximal sustainable at the receiver.

JXTA socket, on the other hand, gets increasingly slower when bandwidth is near saturation, to the point of shutting down the cable modem at 8 senders attempting 1 MB messages. Surprisingly, other pipes in most throughput test configurations outperform the newest and most optimized JXTA socket.

5.5 Summary of Direct Messaging Findings

The most important characteristic of the JXTA messaging components, in respect to message size, is a very slow increase in the RTT with the increasing size of message. The implication of this behavior is that it is more beneficial to send larger messages, considering the constant control overhead inside the messages. This would be

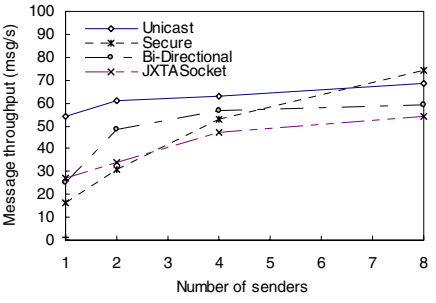


Fig. 7. Scaling of throughput with number of senders (10 KB)

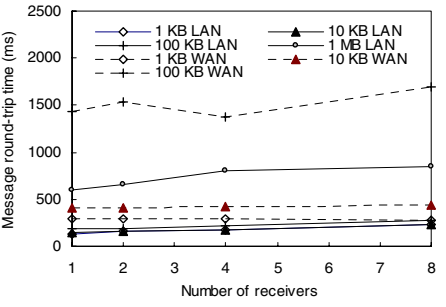


Fig. 8. Scaling of RTT with number of receivers on LAN and WAN

an opposing goal to the fairness towards the network, peer group and especially relays that queue messages irrespective of their size. In the newer JXTA versions, the reduction of message size limits favors the overall peer group and network.

The RTT scales well with the increasing number of senders for 1 and 10 KB messages and visibly worse for 100 KB messages, but still in overall slow linear fashion.

For all JXTA messaging components, the achieved throughput gradually departs from the ideal, dropping to near half of the maximum when 8 senders are connected to the unicast and bi-directional pipes, and 4 senders to the secure pipe and JXTA socket. This issue is recognized and expected to be rectified in the future platform release.

All pipes have lower sending than receiving capacity, which has an interesting implication for file-sharing applications. Peers can download more than they can upload, making them net downloaders by implementation. Application-layer controls would be necessary to balance the uploaded and downloaded volumes of data.

6 Message Propagation

The propagate pipe is already evaluated in a direct 1-1 communication, but its true purpose is to pass messages from one sender to multiple receivers. This is accomplished via a rendezvous peer or by using multicast inside a subnet; therefore, the messaging can be direct or indirect.

Several combinations of transport protocols can be used between the sender, receivers and a rendezvous peer. If the whole peer group is on the LAN, multicast can be used for direct propagation. In case that multicast is not an option, peers communicate using TCP or HTTP, such that the sender passes a message to the rendezvous peer, which propagates it to all connected receivers. A combination of the two is also possible, where the sender has a TCP or HTTP connection to the rendezvous, which uses multicast to propagate messages to the receivers. This is required if the sender itself is unable to use multicast for any reason, e.g. when it is outside the subnet.

Fig. 8 shows the RTT for different message sizes and numbers of receivers in the indirect propagation configuration with sender-rendezvous link using TCP and rendezvous-receiver links using UDP multicast. The performance and scaling properties

in this configuration are overall better than in all-TCP or all-UDP configurations (not shown). The all-TCP configuration yields slower RTT in all combinations of message sizes and number of receivers. The all-UDP (multicast) setup, on the other hand, performs better for 1 and 10 KB messages, but noticeably slower for larger 100 KB and 1 MB messages, due to the fragmentation into multiple UDP packets for messages over 16 KB, including control overhead. The values for increasing number of senders exhibit a slow linear increase, except for 1 MB message size that rises faster to 2 and 4 receivers but then stays almost constant for 8 receivers.

A similar characterization can be given to the WAN results (Fig. 8). The actual RTT is higher due to the increased network distance between the sender and the rendezvous. Larger relative difference is also noted between the values for different message sizes, but very little difference when using increasing numbers of receivers, due to the UDP-based propagation link. High RTT values and jumpiness for 100 KB messages are due to the saturation of the uplink bandwidth of 512 kbps that was used for the direction of payload transmissions. The lower bandwidth is the reason for dropping the 1 MB message size from this test.

The message throughput of the propagate pipe on both the LAN and WAN with multiple receivers (2, 4 and 8) is almost the same as with a single receiver over a direct connection (Fig. 3). The mean throughput of a smooth stream for 1, 2, 4 and 8 receivers shows a very high stability, even across all protocol configurations. The mean throughput for 1 KB messages is 9.06 msg/s with the standard deviation of 0.09 msg/s. Overall, the propagate pipe has poor, but stable throughput on LAN and WAN configurations. However, it is still questionable how well it will scale at higher loads.

7 Indirect Messaging Through Relays

Relays are necessary to connect the pipe ends between peers that are not directly accessible to each other. Relays potentially introduce an overhead by adding the processing cost and extending the pipe length. The RTT test measures the cost of the relay for a two-way communication. The relay throughput test measures the receiving rate of messages given some sending rate and the message path through the relay. Lower receiving rates can quantify the overhead a relay imposes on a one-way pipe throughput. The messages in this test contain one payload element 1 KB in size and edge peers communicate in a 1-1 fashion. Four peer and relay configurations are tested: (1) direct TCP without a relay, (2) single relay using TCP, (3) single relay using HTTP and (4) two relays using at least two HTTP connections.

Since HTTP is a higher-level protocol that uses TCP, it should introduce additional overhead. For JXTA 2.x configuration 4 uses two HTTP connections (between the sender and its relay and between two relays) and one TCP connection (between the receiver and its relay). The addition of TCP in configuration 4 is to prevent the edge peers from optimizing the path by converging to the single relay, as provided by the newer implementation.

The test-bed for this relay evaluation study is on a WAN and includes the JXTA version 2.2 and new non-core messaging components: the bi-directional pipe and JXTA socket. The shown results are representative of the impact that relays have on messaging performance. The results for other JXTA versions and network environ-

ments follow the same patterns, noting that JXTA 1.0 relays used only HTTP and there was no path optimization in configuration 4. The results for JXTA 1.0 and 2.0 on a LAN are discussed in more detail in [4] (core pipes only).

7.1 Effect of Relays on Message RTT

In the direct TCP configuration, two peers connect without any rendezvous or relay. This scenario is shown as a baseline for easier comparison with relayed configurations.

In all of the relayed configurations, the performance trends of the core pipes are consistent, but more emphasized on a WAN. The most intriguing is the effect of the relay and transport on the RTT of the unicast, propagate and bi-directional pipes (Fig. 9). Their latency is lowest through an HTTP relay than in any other non-multicast configuration. This is quite an unexpected but consistent behavior and the reason is found in an implementation bug within TCP transport module that persisted throughout the JXTA versions. This behavior is reportedly fixed in the next JXTA release (2.3). Secure pipe and JXTA socket become uniformly slower through the configurations with added relays, all JXTA versions and both LAN and WAN configurations, as expected.

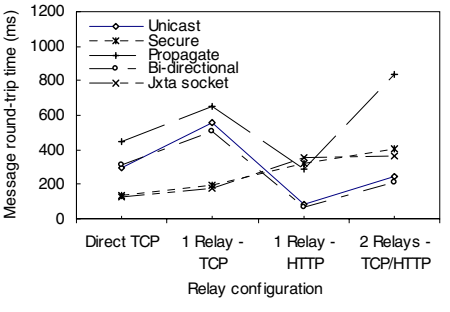


Fig. 9. Effects of relay configurations on pipe RTT

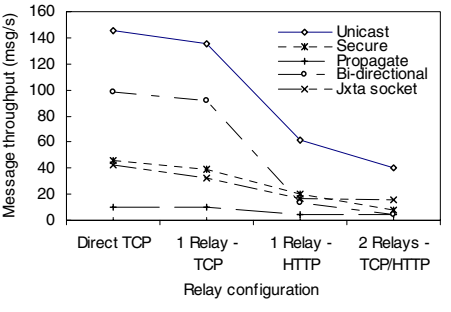


Fig. 10. Effects of relay configurations on pipe throughput

In the two-relay configuration, the relays pass messages between each other on behalf of edge peers. The additional relay has the strongest effect on the propagate pipe causing the RTT to reach the level that will likely be noticeable to the user.

Looking at the evolution through versions [4], an interesting result is that the RTT performance of the relayed secure pipe over TCP improved between JXTA versions 1.0 and 2.x, relative to the unicast and propagate pipe. It is also surprising to see that the secure pipe has lower RTT than the propagate pipe in the two-relay configuration, regardless of the overhead of security provisions.

For JXTA 2.2 the bi-directional pipe and JXTA socket are added to the analysis and they are both susceptible to the effects of relays. The bi-directional pipe continues to exhibit the same patterns and similar RTT to the unicast pipe and JXTA socket behaves like the secure pipe.

7.2 Effect of Relays on Message Throughput

Fig. 10 shows the achieved relay throughput for JXTA 2.2 on a WAN. The throughput values follow the expected pattern due to the addition of relays and HTTP. The issues with the TCP transport module do not affect the one-way throughput. All pipes, including JXTA socket, show the effect of the transport protocol and number of relays.

The propagate pipe in JXTA 2.2 appears to be the least affected, as it has the most stable, but poor throughput under different relay and network setups (close to 10 msg/s). The unicast and bi-directional pipes follow the similar patterns of behavior and the throughput penalty is very high for both. Using an HTTP vs. TCP relay reduces the throughput of the unicast pipe by the penalty factor of 2.20 and of the bi-directional pipe by 6.58. The penalty factor of an additional relay is 1.53 for the unicast pipe and 2.81 for bi-directional pipe vs. 1 HTTP relay. These very high penalties call for improvements in the relay implementations. The JXTA socket still follows the secure pipe very closely, both being affected by relays, but to the smaller degree than unicast and bi-directional pipes.

During the throughput tests of JXTA 1.0, relay peers consistently failed due to overloading. The repeated runs resulted in failures at almost the same message counts indicating a buffering issue. A similar problem occurred during the two-relay WAN test for both 2.x versions (secure, propagate, bi-directional) and single HTTP relay for version 2.2 (secure, propagate). Several test runs of over 1,000 successful message transmissions were required to collect 10,000 samples. The low relay throughput values are traced to the default queue sizes, which can be tuned to produce better results.

8 Implications for System Design

The major lessons learned from the performance analysis are summarized in this section and presented using sample decision paths that may be used by a developer. We concentrate on the performance on a WAN as a more common P2P environment.

The first general recommendation is that the decision-making be prioritized by the effect that the component under consideration has on the whole system. For example, the limitations of the relay peers are likely to have stronger consequences rather than pipe limitations, hence they should be considered first.

It is also recommended that the system designers use benchmarking in the early system analysis and design phases, with parameters that best suit their prospective system. The numerous tradeoffs between features and performance make it worthwhile to carefully pick the appropriate solution. The substantial improvements in JXTA 2.x make the version 1.0 obsolete, and the use of the current version is strongly suggested.

8.1 General Design Case

Within the context of messaging, the choice of the components may start with regard to the relay. If the system or its component require a relay, it is better to look at the constraints early to avoid the potential problems. For JXTA 2.x, it is best to avoid HTTP relay configurations, when using secure, propagate or bi-directional pipes, due to the problems with pipe resolution and transmission reliability. Especially if one-way communication is prevalent, the TCP relay is a better choice because of higher

throughput and reliability. The unicast and bi-directional pipes distinctively perform with highest throughput and stability over a TCP relay. Overall, the TCP relay offers the best support for unidirectional data transfer, such as for file-sharing applications or any case where larger amount of data is transmitted in one direction. On the other hand, HTTP relay is better for bi-directional exchanges, such as instant messaging and collaboration cases, due to lower RTT than TCP relay.

An additional consideration is that the lack of flow control will cause message drops at the overloaded relay, so the safest approach is to deploy a relay on the dedicated machine. The high cost of an additional relay on the message path proves that it was the right decision to implement an optimization algorithm to converge to a single relay accessible to both communicating peers.

Deciding on the messaging component or service is the next step and can be made by looking at major performance advantages and disadvantages of each pipe. In the general deployment case, we look at the unicast and bi-directional pipes. The unicast pipe is suitable for environments of high message loads and bursty traffic. Its overall performance scales well with the message size and number of senders, but its RTT performance degraded since the version 2.0 for small message sizes.

The bi-directional pipe is better suited for peer groups where two-way messaging is prevalent. Its API is also an advantage over the unicast pipe for a typical Java programmer. However, it may be better to stick with unicast pipe in WAN environments of very high message load, where the bi-directional pipe suffers from high message loss and has lower throughput through relays. From the programming perspective, the single messaging object for bi-directional communication and the API similar to Java sockets are traded off with the need to implement a custom multi-threaded server.

An alternative in the general deployment case is JXTA socket, with lowest latency and best scalability for two-way small message exchanges (discussed further). This analysis is continued with specific design cases where reliability, security and efficient message propagation are the main development objectives.

8.2 Reliable Messaging Case

The only messaging component with explicit built-in reliability is the JXTA socket for version 2.2. This messaging component is the best choice for systems that require highest possible reliability. The configurations that maximize JXTA socket performance include messaging under 1 MB per element of transfer, and TCP relays for highest throughput and lowest RTT.

JXTA socket is a clear choice for two-way communication with message sizes up to 100 KB. The API and reliability are the main advantages, but they come at the high price of very low one-way throughput, compared to other pipes. Parallel sender scenarios should also be avoided by having a receiver open multiple input pipes for load balancing. However, although the sustainable load is lower than for unicast and bi-directional pipes, most of the systems would not put a single peer under the higher pressure that it can sustain.

8.3 Secure Messaging Case

JXTA-based systems that require high security, in particular secure messaging, need to be designed and deployed to maximize performance of secure pipes. The bench-

marking results show that secure pipes perform better in JXTA 2.x versions than in 1.0. The RTT of secure pipes is excellent for message sizes of up to 10 KB, even on the WAN. Only the RTT over WAN with message sizes of 100 KB and more tends to scale slightly worse than other pipes with more senders. For JXTA 1.0, complex messages with many elements should be avoided, but in JXTA 2.0, more small elements per message yields better RTT results. If the system requires the use of a relay, the best way to increase performance and reliability is to use a single relay with TCP for JXTA 2.2, HTTP for 2.0 and enable both TCP and HTTP for JXTA 1.0.

Secure pipe is therefore suitable for applications that require secure small message exchange, such as instant messaging and collaboration in secure groups, or for secure exchange of data and results in a distributed computing system.

8.4 Efficient Propagation Case

The propagate pipe has a special feature in transparently allowing messaging from one sender to many receivers. This is an efficient way to propagate data, which is likely to be a primary reason for its selection. Although the performance with respect to the RTT and throughput lags after other pipes, this may be only a secondary factor in the selection process, depending on the application. The results also show that the efficiency and value of the propagation increase with the addition of receivers, especially if the multicast is available. There is an additional benefit when rendezvous peers propagate messages. Receivers do not need to connect directly to the sender, whose only contact is with the rendezvous. This reduces the processing load on the sender, allowing it to run in a constrained environment, such as on a sensor that sends the readings to many interested receivers.

The propagate pipes should be used for under 10 KB message size, usually up to the local subnet multicast limit. The default setting in JXTA is 16 KB and should not be increased arbitrarily. Since the intent is to use propagation for unidirectional data transfer, the main objective is to optimize throughput. A single TCP relay is recommended for indirect relayed communication. Most data dissemination applications can exploit advantages of the propagate pipes, such as content delivery, file sharing, data replication or 1-M group-based instant messaging or chat. However, the throughput limits are still too low for real-time video streaming or similar uses.

9 Conclusion and Further Work

This paper discusses the performance issues of the JXTA platform messaging subsystem. The performance results are collected in a variety of configurations, which are chosen to cover common environments that P2P applications operate in. The ranges of JXTA-specific parameters are chosen with an attempt to show the increasing cost or latency with higher workload, such as peer group and message sizes. Most of the parameter values produce logical and expected results and behaviors. However, a considerable number of unexpected and unintuitive results are obtained as well. This shows that JXTA is still a maturing P2P platform and that there are many opportunities to optimize a JXTA-based system and improve the platform implementation.

As new JXTA versions are released frequently, running a large suite of benchmarks for each version is not feasible, so system designers are advised to incorporate bench-

marking in the early project stages. The requirements of the system would normally dictate the choice of the messaging service, e.g. with respect to its security and reliability provisions, but the performance penalty for additional features may not be acceptable. The presented results and discussion indicate the tradeoffs and it is foreseeable that some systems would not operate under such compromise. However, JXTA offers ample room for implementation of custom provisions such as reliability, as well as tunable buffer and message sizes that can be used to improve performance [1].

The improvements in the JXTA socket and bi-directional pipe implementations would clearly distinguish these two services as the primary messaging options for most application scenarios. JXTA socket in particular needs better throughput for large messages, high load and multiple sender environments, whereas bi-directional pipe needs higher reliability.

During this study, several issues came up that could not be circumvented by changing configurations. The Project community has recognized all discovered issues as being caused by implementation problems and earlier design decisions. Improvements in the secure pipe, propagation service, message queuing and relay implementations are already in progress for the next version, which will bring even better JXTA platform.

There are three main benefits of the presented performance results: learning the limitations of the JXTA platform, formulating the simulation parameters based on empirical results and using the methodology and findings to optimize the design of the messaging subsystem for a new application. This paper includes the discussion of the design optimizations for messaging reliability, security and propagation.

Areas of future work relevant to messaging include primarily scalability evaluation of large peer groups in direct and relayed communication. The next points of interest are the behavior in sub-groups and the performance impact of HTTP in direct messaging. The public JXTA network has grown to size that can be used for traffic tracing and probing in search of the traffic and peer community characteristics. The JXTA community has initiated a number of projects to provide alternate language implementations, most importantly C/C++, which now follows the updates of the J2SE implementation. Further evaluation would show whether different implementations are more appropriate for specific parts of a heterogeneous JXTA P2P system.

References

1. Antoniu, G., et al.: Performance Evaluation of JXTA Communication Layers. *Proc. of GP2PC'05,* Cardiff, UK (2005)
2. Halepovic, E., et al.: The Costs of Using JXTA. *Proc. of P2P '03,* Linköping, Sweden (2003) 160-167
3. Halepovic, E., et al.: JXTA Performance Study. *Proc. of PACRIM '03,* Victoria, BC, Canada (2003) 149-154
4. Halepovic, E., et al.: The JXTA Performance Model and Evaluation. Future Generation Computer Systems, Elsevier, Vol. 21, No. 3. (2005) 377-390
5. Java Message Service (JMS). Sun Microsystems, Inc. http://java.sun.com/products/jms/
6. Junginger, M., et al.: The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks. *Proc. of P2P '02,* Linköping, Sweden (2002)
7. JXTA Bench Project. http://bench.jxta.org/

8. JXTA v2.0 Protocols Specification. http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html, 2003
9. Matossian, V., et al.: Enabling Peer-to-Peer Interactions for Scientific Applications on the Grid. *Proc. of Euro-Par '03,* Klagenfurt, Austria (2003) 1240-1247
10. MyJXTA2 Enterprise Edition. http://myjxta2.jxta.org/servlets/ProjectHome, 2002
11. Pallickara, S., et al.: NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. *Proc. of Middleware '03,* Rio de Janeiro, Brazil (2003) 41-61
12. Parker, D.C., et al.: Building Near Real-Time Peer-to-Peer Applications with JXTA. *Proc. of GP2PC '04,* Chicago, USA (2004)
13. Project JXTA Home Page. http://www.jxta.org/
14. Seigneur, J.-M., et al.: P2P with JXTA-Java pipes. *Proc. of PPPJ '03,* Kilkenny City, Ireland (2003)
15. Tran, P., et al.: JXTA and TIBCO Rendezvous – An Architectural and Performance Comparison. http://www.smartspaces.csiro.au/links.htm, 2003
16. Verbeke, J., et al.: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. *Proc. of GRID'02,* Baltimore, MD, USA (2002) 1-12

An Aspect-Oriented Communication Middleware System

Marco Tulio de Oliveira Valente, Fabio Tirelo,
Diana Campos Leao, and Rodrigo Palhares Silva

Department of Computer Science,
Catholic University of Minas Gerais, Brazil
{mtov, ftirelo}@pucminas.br

Abstract. This paper describes a Java-based communication middleware, called AspectJRMI, that applies aspect-oriented programming concepts to achieve the following requirements: (1) modular implementation of its features, including those with a crosscutting behavior; (2) high degree of configurability and adaptability; (3) performance similar to conventional object-oriented communication middleware systems, such as CORBA and Java RMI. In AspectJRMI, users may explicitly select the features provided by the middleware infrastructure, according to their needs. Most of these features have a crosscutting behavior, including interceptors, oneway calls, asynchronous calls, value-result parameter passing, and collocation optimizations. In this case, they are implemented as aspects. The design of AspectJRMI follows a set of principles, called horizontal decomposition, to achieve pluggability of aspects to the core middleware implementation. This paper presents the programming interface and the implementation of AspectJRMI. It also presents experimental results of its performance.

1 Introduction

Middleware systems, such as CORBA [14] and Java RMI [24], encapsulate several details inherent to distributed programming, including communication protocols, data marshalling and unmarshalling, heterogeneity, service lookup, synchronization, and failure handling. Although such systems have been proposed to make distributed programming more simple and natural, they have evolved over the years to become complex, monolithic, and heavyweight [7, 26, 28]. On the other hand, a wide range of applications use only a reduced subset of middleware features. In this case, the monolithic architecture of middleware contributes to increase the complexity, size, and the resource requirements of such systems, without providing proportional benefit.

This paper describes a Java-based communication middleware, called AspectJRMI, that applies aspect-oriented programming concepts to achieve the following requirements:

– Modular and open implementation of features, including those having a crosscutting behavior. In AspectJRMI, users can change and extend the main components of the platform.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1115–1132, 2005.

- High degree of configurability and pluggability. At compile time, users may explicitly select the features provided by the middleware infrastructure, according to their needs.
- Performance similar to conventional object-oriented communication middleware systems, such as CORBA and Java RMI.

In order to achieve such requirements, the design of AspectJRMI follows a set of principles, called horizontal decomposition [28], that advocates the synergistic combination of objects and aspects in order to modularize middleware concerns. The components of AspectJRMI are divided in two categories: mandatory and non-mandatory. Following the guidelines proposed by horizontal decomposition method, mandatory (or core) components are those related to the main middleware functionality, i.e., components that support the implementation of transparent remote method invocations (using call-by-value, at-most-once, and synchronous semantics). As examples of mandatory components, we can mention channels, stubs, skeletons, and remote references. Moreover, in AspectJRMI users can extend or adapt the default behavior of mandatory components. For example, they can define channels that use different transport protocols or add extra functions to channels (such as encryption or compression of messages). Mandatory components were implemented using traditional techniques in object-oriented programming (such as design patterns and vertical decomposition). Particularly, the core of the system was implemented using components defined in Arcademis [17], a Java-based framework that supports the implementation of middleware architectures.

Non-mandatory components implement features that are not required in every distributed application. Most of these features have a crosscutting behavior, such as interceptors, oneway calls, asynchronous calls, value-result parameter passing, and collocation optimizations [26, 27]. For this reason, they are implemented as aspects using AspectJ [11]. An aspect compiler is used to weave the core and optional components. Moreover, the static nature of the weaving process in AspectJ contributes to keep the performance and memory footprint overhead of AspectJRMI at acceptable values.

The rest of the paper is structured as follows. Section 2 presents an overview of the main principles advocated by horizontal decomposition. Section 3 describes the core of AspectJRMI. First, the section describes the overall architecture of Arcademis and then describes the main components of the AspectJRMI core. Section 4 presents the aspects that can be woven to the core in order to support the following crosscutting features: oneway calls, asynchronous calls, call by value-result, service combinators, remote reference decorators and collocation optimizations. Section 5 describes an experimental evaluation of the size and performance overhead of AspectJRMI. Section 5 describes related work and Section 6 concludes the paper.

2 Horizontal Decomposition

Horizontal decomposition is a set of guidelines and principles that support the implementation of middleware systems with high degrees of modularity and

adaptability [28]. The method proposes a solution to the *feature convolution* phenomenon in traditional middleware systems, i.e., the fact that many middleware features can not be easily plugged in and plugged out of the platform, since they crosscut the implementation of other features. Horizontal decomposition advocates the use of traditional modularization techniques, such as vertical decomposition, to implement a minimal but well-modularized core middleware system. Basically, this core should provide support to synchronous and statically defined remote invocations, using call-by-value parameter passing. Horizontal decomposition also advocates the use of aspect-oriented programming to superimpose orthogonal features in this core. A feature is considered orthogonal if both its semantics and implementation do not fit in a single component. Traditional weaving process is used to compose the core and the orthogonal features that are requested in a particular application. The method supports fine-grained customizations; particularly, features not used by an application do not impact the generated middleware platform.

3 AspectJRMI Core

The core of the system is derived from components defined in Arcademis [17], a Java-based framework that supports the implementation of customizable middleware architectures.

3.1 Arcademis Architecture

Arcademis predefines the overall architecture of middleware platforms, as described in Figure 1. In object-oriented middleware platforms, clients traditionally use intermediate components to invoke methods on remote objects. Two of these components are the stub, which exists on the client side of a distributed

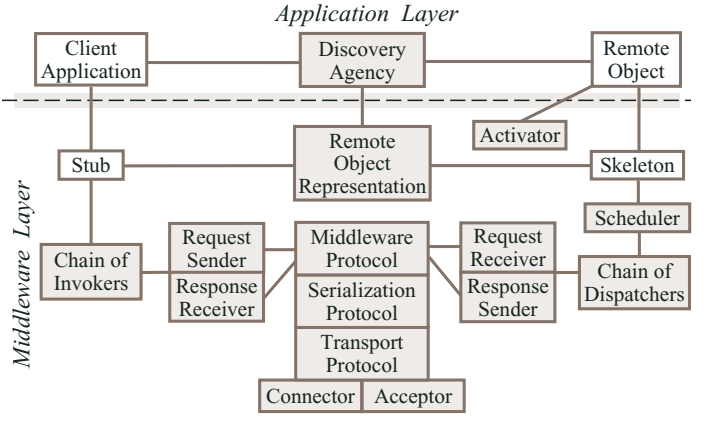


Fig. 1. Arcademis Architecture

application, and the skeleton, which is located on the server side. The stub acts as a local proxy for the remote object, and its function is to forward to the server remote calls made by the client. The skeleton represents the invoking client to the remote object, acting as an adapter.

Besides stubs and skeletons, Arcademis defines several other components. The `invoker` is responsible for emitting remote calls, whereas its server counterpart, the `dispatcher`, is in charge of receiving and passing them to the skeleton. The `Scheduler` is used whenever necessary to sort remote calls according to their priorities. The communication layer in Arcademis is represented by a set of components that constitute the transport protocol, the serialization protocol and the middleware protocol. Connections are established by two components: the `Connector` and the `Acceptor`. Request senders and receivers provide means to assure the reliability level the middleware provides to distributed applications. A lookup service allows clients to discover and access remote objects. Finally, the `Activator` defines how an object is made ready for receiving remote calls.

3.2 AspectJRMI Core Components

The core of AspectJRMI is an instance of Arcademis that provides a basic remote method invocation service. The components in the core implement interfaces and abstract classes defined in Arcademis and reuse concrete classes from this system. Including Arcademis components, the core is composed by 4 interfaces, 11 abstract classes, 89 concrete classes and almost 9000 lines of Java code. The core also includes a stub/skeleton compiler.

Communication Components: The core uses TCP/IP for data transmission. It also supports a middleware protocol with four different types of messages: *call*, *return*, *ping* and *ack*. The *call* message describes a remote invocation, including its arguments and identifiers. The *return* message holds the results of remote calls. Messages *ping* and *ack* are used in order to verify if servers or clients are alive.

In the core, acceptor and connector components provide a synchronous service, meaning that the client thread remains blocked during the execution of remote calls. On the server side, a connector creates a new thread for each incoming connection. Service handler components (request sender/receiver and response receiver/sender) implement an at-most-once invocation semantics.

Client and Server Configurations: We have implemented two versions of the core of AspectJRMI. The first version has only client functionality, i.e., it has only components in charge of dispatching remote method invocations, such as connector, invoker, request sender and stub. In the current implementation of the system the client core has 37 KB (size of all the .class files). The second version supports client and server functionality, i.e., it has components in charge of sending and receiving remote method invocations. Besides the client components, this version includes the following elements: acceptor, activator, connection server, dispatcher, skeleton and request response. The client/server core has 68 KB.

Reconfiguration of Core Components: Following the Arcademis architecture, the core of AspectJRMI has a singleton component called ORB. The ORB contains a set of factories that eases changing the implementation of a component without interfering in other modules of the system. In order to extend the behavior of a component, users should only associate a new factory to the ORB. The ORB has factories for the following components: channels, service handlers, activators, connectors, messages, end-points, notifiers, acceptors, streams, schedulers, connection servers, remote object identifiers and dispatchers.

4 Crosscutting Features

Besides offering a synchronous remote method invocation service, AspectJRMI also provides modular implementations of the following crosscutting features: oneway calls, asynchronous calls, call by value-result, service combinators, remote reference decorators and collocation optimization.

4.1 Oneway Calls

In oneway calls, control returns to the client as soon as the middleware layer receives the call. Thus, client and remote method executions are asynchronous. Oneway calls neither return values nor throw remote exceptions. In AspectJRMI, the abstract aspect `OneWayAspect` defines oneway calls. This aspect has the abstract pointcut `onewayCalls` that defines the calls dispatched using oneway semantics.

```
abstract aspect OneWayAspect {
  protected abstract pointcut onewayCalls();
}
```

Example: Aspect `MyOneWayAspect` defines that calls to `void Hello.sayHello()` should be dispatched using oneway semantics.

```
aspect MyOneWayAspect extends OneWayAspect {
  protected pointcut onewayCalls(): call(void Hello.sayHello());
}
```

4.2 Asynchronous Calls

In order to use asynchronous calls, the programmer has to define which methods are intended to be asynchronously called by means of intertype declarations of AspectJ. AspectJRMI provides the aspect `AsynchronousCallsAspect` which encapsulates almost all implementation details of this feature. This aspect relies on pointcut `asynchronousCalls()` to specify asynchronous calls, which are those whose names start with `async_`, return `Future`, and are not performed within `Skeleton` classes.

```
public aspect AsynchronousCallsAspect {
 public pointcut asynchronousCalls():
   call(Future *.async_*(..)) && !(within(*_Skeleton));
}
```

Method `getResult` in class `Future` returns the result of an asynchronous remote call. If this call has not finished, `getResult` waits its conclusion. If this call fails, `getResult` throws an exception of type `RemoteException`.

Example: Consider that the methods in interface `Hello` may be called asynchronously.

```
interface Hello extends Remote {
 String sayHello() throws ArcademisException;
 String sayHello(String name) throws ArcademisException;
}
```

For this purpose, it is only necessary to define an aspect `AsyncHello` that introduces in interface `Hello` an asynchronous version for each of its methods. The asynchronous version must return `Future`, have a prefix `async_` in its name and a body that just returns `null`.

```
aspect AsyncHello {
  public Future Hello.async_sayHello() { return null; }
  public Future Hello.async_sayHello(String name) { return null; }
}
```

The following code fragment asynchronously calls method `async_sayHello` and later uses method `getResult` for synchronizing and getting its result.

```
Future ftr = server.async_sayHello("Bob");
...
String res = (String) ftr.getResult();
```

4.3 Call by Value-Result

Call by value-result is specified by defining formal parameters as having type `Holder`. This class plays the role of a wrapper for the argument passed by value-result. This argument must implement the interface `Marshable`, which denotes serializable values in Arcademis. Class `Holder` has the following interface:

```
class Holder {
  public Holder(Marshable o);
  public Marshable getValue();
  public void setValue(Marshable o);
}
```

On calling a remote method that has a parameter of type `Holder`, a client must: (i) create an instance h of an object of type `Holder` enclosing the actual parameter value; (ii) use h as an actual parameter; and (iii) on return of the method, extract the result from holder h.

In order to use call by value-result, the programmer does not need to define any aspect. The implementation of AspectJRMI uses the following internal aspect to intercept calls having parameters of type `Holder`:

```
pointcut callbyValueResult(): call(* *(..,Holder,..));
```

Example: The following code fragment defines a method with two value-result parameters, as well as a client using this method.

```
void foo(Holder h1, Holder h2) {
  Date d = (Date) h1.getValue();
  d.setDate(12, d.getDay() + 1, 2004);
  h2.setValue(new Date(2, 28, 2005));
}
........
Holder h1 = new Holder(new Date(2,17,2005));
Holder h2 = new Holder(new Date());
foo(h1,h2);
((Date) h1.getValue()).print(); // prints December 18th, 2004
((Date) h2.getValue()).print(); // prints February 28th, 2005
```

In addition, AspectJRMI supports call by result, in which parameters are used to return values from methods. Call by result is specified in a way similar to call by value-result, using type `ResultHolder`.

Restriction: Call by value-result and call by result are incompatible with asynchronous and oneway calls. If this restriction is not followed, a run time error is raised.

4.4 Service Combinators

Service combinators were originally proposed to handle failures and to customize programs that need to retrieve web pages [4]. In AspectJRMI, we adapt this mechanism in order to express that remote invocations should be dispatched sequentially (to provide fault tolerance), concurrently (to decrease response time) or non-deterministically (to provide load distribution).

Let C_1 and C_2 be two remote method calls. AspectJRMI allows the composition of these calls by means of the following service combinators:

- $C_1 > C_2$ (alternative execution): C_1 is invoked; if its invocation fails then C_2 is invoked; if the invocation of C_2 also fails then the combined call fails. Thus, the combinator $>$ provides fault tolerance.

- C_1 ? C_2 (non-deterministic choice): Either C_1 or C_2 is non-deterministically chosen to be invoked. If the selected call fails, then the other one is invoked. The combinator fails when both C_1 and C_2 fail. Thus, service combinator ? provides load balancing.
- C_1 | C_2 (concurrent execution): Both C_1 and C_2 are concurrently started. The combinator returns the result of the first call that succeeds first; the other one is ignored. The combinator fails when both calls fail. Thus, service combinator | optimizes the response time of an idempotent remote call by concurrently invoking it in two servers.

Service combinators are associated with remote references using the following classes:

```
class SimpleRemoteRef extends RemoteRef {
  public SimpleRemoteRef(Remote endpoint);
}
class StructuredRemoteRef extends RemoteRef {
  public StructuredRemoteRef(char op, RemoteRef r1, RemoteRef r2);
}
```

The abstract class `RemoteRef` represents a remote reference in the system. `SimpleRemoteRef` denotes a standard remote reference, with no service combinator, whereas `StructuredRemoteRef` denotes a remote reference associated with a service combinator. Users can also provide their own subclasses of `RemoteRef` in order to define new combinators.

The abstract aspect `RemoteAspect` associates service combinators with remote references:

```
abstract aspect RemoteAspect {
  protected abstract pointcut RemoteCalls();
  protected abstract RemoteRef getRemoteRef();
}
```

Aspects extending `RemoteAspect` must define the calls with an associated service combinator (pointcut `RemoteCalls`) and the `RemoteRef` used in the invocation of such calls (method `getRemoteRef`).

Example: Suppose the following client of interface `Hello` (Section 4.2).

```
class HelloClient {
  Hello server = RemoteRef.InitRemoteRef();
  String s1 = server.sayHello();
  String s2 = server.sayHello("Bob");
}
```

Suppose we want to use the following tactics in invocations of services of type `Hello`: each call must be first dispatched to the server `helloSrv` in node

skank ; on failure, it must be dispatched to the server helloSrv in node patofu. Moreover, this tactic must be associated with calls to Hello methods inside class HelloClient. The aspect HelloClientAspect implements the proposed invocation tactics:

```
 1: aspect HelloClientAspect extends RemoteAspect {
 2:   private RemoteRef ref;
 3:   protected pointcut RemoteCalls(): within(HelloClient)
 4:                                     && call(* Hello.*(..));
 5:   public HelloClientAspect() {
 6:     String s1 = "skank.inf.pucminas.br/helloSrv";
 7:     String s2 = "patofu.inf.pucminas.br/helloSrv";
 8:     ref = new StructuredRemoteRef('>',
 9:                 new SimpleRemoteRef(RmeNaming.lookup(s1)),
10:                 new SimpleRemoteRef(RmeNaming.lookup(s2)));
11:   }
12:   protected RemoteRef getRemoteRef() {
13:     return ref;
14:   }
15: }
```

Line 3 specifies that the invocation tactics supported by this aspect must be associated with calls of methods of type Hello occurring inside class HelloClient. Lines 8-10 create a structured remote reference using service combinator > for failure recovery.

Restrictions: In order to be properly combined, C_1 and C_2 should represent calls to methods sharing a common name (probably having different target objects). Moreover, the static type checking rules of Java prevents combining of synchronous and asynchronous calls.

4.5 Remote Reference Decorators

Remote reference decorators (also known as interceptors in CORBA) are used to insert additional behavior in the invocation path of remote calls. Decorators use class composition to create a chain of tasks to be executed in the invocation flow of a remote call. A decorator is defined by means of class RemoteRefDecorator:

```
abstract class RemoteRefDecorator extends RemoteRef {
  public RemoteRefDecorator(RemoteRef ref);
}
```

The constructor of this class has a parameter ref which denotes the remote reference to be decorated. A remote reference decorator should extend the class RemoteRefDecorator. AspectJRMI provides the following standard decorators: Cache (that implements a cache with the results of idempotent remote calls); Log (that provides a log service for remote calls); and Timer (that specifies a timeout for the execution of a remote call before throwing an exception).

Example: The following fragment of code associates a `Log` decorator with both components of the structured remote reference of the previous example.

```
 8: ref = new StructuredRemoteRef('>',
 9:          new Log(new SimpleRemoteRef(RmeNaming.lookup(s1))),
10:          new Log(new SimpleRemoteRef(RmeNaming.lookup(s2))));
```

4.6 Collocation Optimizations

In distributed object-oriented systems, there are situations where servers and clients are collocated in the same address space. In such cases, there is no need to dispatch remote calls using the middleware infrastructure. Instead, remote invocations may be directly forwarded to the server object. This strategy is usually called direct collocation optimization[21].

In AspectJRMI, direct collocation does restrict or change the semantics of other concerns associated to an invocations subjected to optimization. Particularly, oneway calls and asynchronous calls can be forwarded to a local object. Moreover, service combinators, remote reference decorators, and call by value-result preserve their semantics in calls subjected to optimizations.

In order to activate collocation optimizations, we need to weave an aspect named `CollocationAspect` with the component application. This aspect has the following interface:

```
aspect CollocationAspect {
  pointcut remoteObjectInit(RemoteObject r):
     initialization(RemoteObject+.new(..)) && this(r);
  pointcut remoteObjectAsResult():
     call(RemoteObject+ *(..));
  pointcut remoteObjectAsParam():
     call(public object Stream.readObject() && within(*_Skeleton);
}
```

Pointcut `remoteObjectInit` captures the initialization of remote objects. An advice associated to this pointcut inserts the identifier of this object in an internal collocation table. Pointcut `remoteObjectAsResult` captures all method invocations that return a `RemoteObject`. An around advice associated with this aspect checks whether or not the identifier of this remote object is in the collocation table. If it is, a local reference is returned instead of a remote reference. In the lexical scope of skeletons, pointcut `remoteObjectAsParam` captures the deserialization of objects that are passed as parameters in remote calls. Similar to the previous advice, an around advice checks whether or not the deserialized object is local. If it is, a local reference for it is returned.

5 Experimental Results

This section presents results obtained from experiments performed with our implementation of AspectJRMI. The experiments were used to evaluate the size and performance overhead of AspectJRMI.

5.1 Size Overhead

We use the following tools in the experiments: `javac` (JDK 1.4 version), `ajc` (version 1.5.0) and `abc` (version 1.0.2). The `ajc` tool is the default aspect compiler for AspectJ and `abc` is an aspect compiler that incorporates a series of optimizations [1].

Table 1 summarizes the size of the AspectJRMI framework. As reported in Section 3.2, the client version of the core has 37 KB and the client/server version has 68 KB. The internal classes and aspects of AspectJRMI have 50 KB. There are also 41 KB from the AspectJ run-time (package `aspectjrt.jar`). Thus, the total size of the system is 128 KB (client only) and 159 KB (client/server), which we consider a competitive value. For example, full implementations of CORBA, such as the JacORB system [9], have approximately 10 MB. Implementations of CORBA for mobile and embedded devices, such as ORBit2 [16], have at least 2 MB. On the other hand, some implementations of CORBA are smaller than the ones mentioned. For example, the UIC-CORBA system has 48.5 KB in the Windows CE platform and around 100 KB in Windows 2000 [19]. However, UIC-CORBA supports only a basic remote method invocation service.

Table 1. Size (in KB) of AspectJRMI components

Component	Client	Server
Core	37	68
AspectJRMI	50	50
AspectJ	41	41
Total	128	159

Besides the size of the internal components of the framework, there is also the cost of the weaving process. In order to evaluate such cost, we have implemented the following programs:

- P1: A client that calls, using a oneway semantics, a remote method passing as argument a string with 16 chars and an array with 16 integers.
- P2: A client that calls, using an asynchronous semantics, a remote method that receives as argument two integers and returns a string.
- P3: A client that calls a remote method passing as argument two arrays of integers, with size 128 and 32. The call uses as target a remote reference with an associated `?` service combinator (non-deterministic choice).
- P4: A client that calls a remote method passing as argument a string with 32 chars and two integers. The invocation uses call by value-result.

We have used the `ajc` and `abc` weavers to compile such applications. Furthermore, in order to provide a lower bound for comparison we have changed programs P1 and P2 to use a standard synchronous semantics. Program P3 was changed in order to remove the `?` service combinator. Program P4 was changed

Table 2. Size (in KB) of the experiments (for one call)

	One call	Weaving	ajc	abc	javac	abc-javac
P1	Oneway	Client	4.99	2.46	0.84	1.62
P2	Asynchronous	Client	11.00	5.76	2.84	2.92
P3	Combinator ?	Client	5.76	3.21	0.81	2.40
P4	Value-result	Client/Server	8.67	5.53	1.92	3.61

Table 3. Size (in KB) of the experiments (for 100 calls)

	100 calls	Weaving	ajc	abc	javac	(abc-javac)/100
P1	Oneway	Client	47.9	11.5	2.07	0.09
P2	Asynchronous	Client	11.0	5.76	2.84	0.03
P3	Combinator ?	Client	48.9	12.2	2.05	0.10
P4	Value-Result	Client/Server	8.67	5.53	1.92	0.03

to use call by value. The modified and simplified programs were compiled using the standard `javac` compiler.

Table 2 summarizes the results. Column weaving describes the components of the application that were instrumented by the weaver compiler (client or client/server). Columns `ajc` and `abc` show the size of these components after the weaving (using the `ajc` and `abc` compilers). The next column shows the size of these components in the modified programs when compiled using the `javac` compiler. Finally, column `abc-javac` presents the difference in size of the code generated by the two compilers.

The results of the first experiment show that the `ajc` compiler introduces a considerable size overhead. Certainly, the reason is that this compiler does not support many optimizations that are possible in aspect-oriented languages [2]. On the other hand, the results using the `abc` are much more acceptable. When compared to the `javac` programs, the overhead ranges from 1.62 KB (for program P1) to 3.61 KB (for program P4). This overhead is for one remote call that adds a new feature to both programs (oneway semantics in the case of program P1 and a service combinator in program P3).

In the second group of experiments, we have changed programs P1 to P4 to make 100 remote calls sequentially (one call after the other, without the use of loops). Table 3 presents the results. Considering only the `abc` compiler, the overhead was significantly reduced, ranging from 30 bytes per remote call (programs P2 and P4) to 100 bytes per remote call (program P3). We believe this overhead is fully acceptable.

5.2 Performance Overhead

Oneway Calls, Asynchronous Calls and Call by Value-Result: In order to evaluate the performance of such features we have reused programs P1, P2, and P4

from the previous section. We run such programs (client and server processes) on a Pentium 4 machine, with 2.00 GHZ, 512 KB RAM, Microsoft Windows Service Pack 4, JDK 5.0 and `abc` aspect compiler (version 1.0.2). Each remote invocation was executed 5000 times. In program P2, we measured the time to dispatch the asynchronous calls, store the `Future` values in an array and retrieve all results from this array (using the `getResult` method). To establish a comparison, we ported programs P1, P2, and P4 to JacORB, preserving their semantics. Program P3 was excluded from the experiment since in JacORB there is no support to service combinators. Table 4 presents the results in calls/second. As showed in this table, the performance of AspectJRMI is very close to JacORB performance.

Table 4. Throughput (in calls/sec)

		AspectJRMI (A)	**JacORB (B)**	**A / B**
P1	Oneway	3595	3426	1.04
P2	Asynchronous	2028	2159	0.94
P4	Value-Result	2192	2153	1.02

Collocation Optimizations: A second experiment was conducted in order to measure the performance gains of collocation optimizations. The following program was used in this experiment:

```
1: s.f1(b1);        // remote call (b1 is a local object)
2: B b2= s.f2()     // remote call that returns a reference to b1
3: b2.g();          // optimization: b2.g() ==> b1.g()
```

The same program has been compiled and executed with and without optimizations. We execute each remote call (lines 1 to 3) 5000 times, with client and server processes in the same machine. All methods have an empty body. Table 5 presents the results in calls/msec. When the collocation aspect was woven to the application we observed a small reduction in the throughput of remote calls not subjected to optimization (lines 1 and 2). This was due to the need to check if the arguments or the results are local objects. On the other hand, calls subjected to optimization (line 3) achieved a substantial performance gain. This result was expected since all the middleware overhead was fully eliminated, including marshalling, unmarshalling, and TCP/IP communication. Improvements in the same order of magnitude were already reported for direct collocation optimization in CORBA [13, 21].

6 Related Work

Customizable and Adaptive Middleware: Several software engineering techniques have been applied in the construction of customizable, open and adaptive middleware platforms. Computational reflection, for example, is the central concept

Table 5. Calls/msec with collocation optimization enabled and disabled

Line	Call	Disabled (A)	Enabled (B)	B/A
1	s.f1(b)	3.36	3.31	0.98
2	s.f2()	3.51	3.50	0.99
3	b2.g()	3.40	833.33	245.09

of systems such as openORB [3], openCOM [5], UIC [19], and dynamicTAO [12]. However, reflective middleware systems often provide low level APIs and introduce non-marginal performance and memory overheads. Systems such as Quarterware [22] and Arcademis [17] rely on the concept of frameworks. These systems provide semi-complete platforms that can be extended and personalized by middleware users. Other systems, such as TAO [20], relies on design patterns. However, frameworks and design patterns do not provide modularized implementation for crosscutting features, such as interceptors, oneway calls, asynchronous calls, value-result parameter passing, and collocation optimizations. Therefore, it is usually difficult to remove or add such features in the middleware platform. Particularly, even if a crosscutting feature is not requested in a particular application, the middleware carries code to support its implementation. This increments the size of the system and may impact its performance.

AOP Refactorization of ORBacus: Using aspect mining techniques, Zhang and Jacobsen have quantified the crosscutting nature of several features of CORBA based middleware [26, 27]. They have measured the scattering degree of features such as portable interceptors, dynamic programming invocations, collocation optimizations, and asynchronous calls. They have also showed that such features can be modularized using aspect-oriented programming. From this experience, they proposed the horizontal decomposition method [28]. They have assessed the effectiveness of their method by re-implementing as aspects crosscutting features of the original implementation of ORBacus [15]. As expected, their refactorization preserves the CORBA programming interface.

AspectJRMI design was not constrained by a predefined programming interface. Similar to Java RMI [24], AspectJRMI does not extend the Java language nor require a particular interface definition language. Instead, the system leverages the pointcut mechanism of AspectJ to define features such as oneway calls, service combinators, and decorators. Intertype declarations are used to define asynchronous calls. This contributes to increase the degree of obliviousness provided by the middleware API.

Just-in-time Middleware and Abacus: Following horizontal decomposition guidelines, the Just-in-time Middleware (JiM) paradigm [25] advocates that middleware implementation should be pos-postulated, i.e., middleware components should be selected and assembled after the user application is specified. Abacus is a prototype implementation of JiM principles. The system relies on an aspect-aware compiler, called Arachne, to generate just-in-time middleware configurations. Arachne collects middleware functionalities from IDL declarations

and from a user interface. The synthesis process also depends on dependencies, constraints and convolution descriptions. Dependencies define that the implementation of a certain feature is composed by other ones. Constraints specify that some features must be included or excluded from the system depending on external conditions. Convolutions descriptions specify that a feature crosscuts the implementation of other features. On the other hand, AspectJRMI shows that a standard AOP language and weaver can be used to associate pospostulated features to a minimal core middleware system. From the list of aspects available in Abacus, AspectJRMI does not provide support only to server data types (since the system relies on the Java type system) and some CORBA advanced features (such as interface repository and dynamic invocation interface). However, we believe that AspectJRMI can be extended to include such missing aspects.

Other Applications of AOP to Middleware: JBossAOP [10] uses Java 1.5 annotations to support the implementation of several concerns, including oneway calls, asynchronous calls, transactions and persistence. Basically, in JBossAOP annotations provide syntactical hooks to insert aspectual code. However, one can argue that in this case annotations introduce code scattering, since they can be required in several elements of the system. Preferably, annotations should be used to express functional behavior of the annotated element, for example, to define that an operation is idempotent.

Alice [8] is a middleware that proposes the combination of aspects and annotations to implement container services, such as authentication and sessions. AspectJ2EE [6] proposes the use of AOP to implement open, flexible and extensible container middleware. Soares, Laureano and Borba [23] have proposed a set of guidelines to implement distribution, persistence and transactions as aspects. The previous systems, however, consider the underlying communication middleware as a monolithic block.

FACET [18] is an implementation of a CORBA event channel that relies on AOP to provide a customizable event system. Similar to AspectJRMI, the system has a core and a set of selectable features. Each feature adds a new functionality to the core or to other feature. Examples of features include pulling events, dispatching strategies, event payload types, event correlation and filtering, and event profile. FACET also includes a test framework that automatically validates combinations of features. Thus, FACET design follows many of the horizontal decomposition principles.

7 Conclusion

In this paper we described an aspect-oriented communication middleware system with high degree of modularization, configurability and customizability. The design of the system has followed the horizontal decomposition principles, and thus AspectJRMI has a set of mandatory components (middleware core) designed using traditional vertical decomposition techniques. Furthermore, AspectJRMI uses aspects to encapsulate optional features so that users can select

only those features that are needed in a particular distributed application. An aspect compiler is used to weave aspects and mandatory components at compile time.

The core of the system is derived from components defined in Arcademis, a Java-based framework that supports the implementation of customizable middleware architectures. Well-known design patterns, such as singletons, factories, strategies, decorators and façades, are used to foster a non-monolithic and flexible middleware core. Users can change and extend almost all internal components of the core, including channels, invokers, service handlers, streams, remote references, dispatchers, and activators. Despite its open architecture, the core of AspectJRMI has just 37 KB (client features only) or 68 KB (client and server features). For example, we were able to successfully run the core in the CLDC configuration of the J2ME platform. This configuration targets resource constrained devices, such as cell phones and low-end PDAs.

The core provides a primitive remote invocation service (synchronous, using call by value and at-most-once semantics). Whenever crosscutting functionalities are required, they can be introduced using aspects. AspectJRMI provides aspects for the following features: oneway calls, asynchronous calls, service combinators, remote reference decorators, value-result and result parameter passing, and direct collocation optimizations. All these aspects when packed have around 50 KB. When using the `abc` AspectJ compiler the cost of weaving aspects to the base application ranges from 30 to 100 bytes per remote call. This makes AspectJRMI a competitive solution when compared to other middleware implementations. For example, minimal CORBA implementations that are equivalent to AspectJRMI core have around 50 to 100 KB [19]. On the other hand, full implementations of CORBA have at least 2 MB [16]. Thus, AspectJRMI shows that aspect-oriented programming is an effective solution to provide middleware systems with size and functionalities between these bounds. Our experiments have also shown that our static weaving approach has not significantly impacted middleware performance. We were able to obtain performance results equivalent to mature CORBA implementations.

Differently from CORBA, AspectJRMI is a solution to Java-based distributed applications. For this reason, the proposed middleware does not require specific interface definition languages and type systems. The middleware also does not require extensions to the Java object model or execution environment. The same approach is followed by Java RMI, which provides just a basic invocation service, supported by a monolithic kernel. In AspectJRMI, aspect-oriented abstractions, such as pointcuts and intertype declarations, are used to add extra features to an open and non-monolithic kernel. The only requirement is that users should be familiar with an aspect-oriented language (AspectJ, in the case).

As future work, we intend to support other non-functional requirements, besides distribution. We plan to investigate support to aspects such as security, load balancing, fault tolerance and persistence.

References

1. P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, J. Lhotak, O. Lhotak, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. abc: An extensible AspectJ compiler. In *4th International Conference on Aspect-Oriented Software Development*, pages 87–98. ACM Press, 2005.
2. P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, J. Lhotak, O. Lhotak, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. Optimising AspectJ. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 117–128. ACM Press, 2005.
3. G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. M. Costa, H. A. Duran-Limon, T. Fitzpatrick, L. Johnston, R. S. Moreira, N. Parlavantzas, and K. B. Saikoski. The design and implementation of Open ORB 2. *IEEE Distributed Systems Online*, 2(6), 2001.
4. L. Cardelli and R. Davies. Service combinators for web computing. *IEEE Transactions on Software Engineering*, 25(3):309–316, 1999.
5. M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas. An efficient component model for the construction of adaptive middleware. In *Middleware: IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218 of *LNCS*, pages 160–178. Springer-Verlag, 2001.
6. T. Cohen and J. Gil. AspectJ2EE = AOP + J2EE. In *18th European Conference Object-Oriented Programming*, volume 3086 of *LNCS*, pages 219–243. Springer-Verlag, 2004.
7. A. Colyer and A. Clement. Large-scale AOSD for middleware. In *3rd International Conference on Aspect-Oriented Software Development*, pages 56–65. ACM Press, 2004.
8. M. Eichberg and M. Mezini. Alice: Modularization of middleware using aspect-oriented programming. In *4th International Workshop on Software Engineering and Middleware*, volume 3437 of *LNCS*, pages 47–63. Springer-Verlag, 2005.
9. JacORB. http://www.jacorb.org.
10. JBossAOP. http://www.jboss.org/developers/projects/jboss/aop.
11. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *15th European Conference on Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–355. Springer Verlag, 2001.
12. F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhes, and R. Campbell. Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In *IFIP/ACM International Conference on Distributed Systems Platforms*, volume 1795 of *LNCS*, pages 121–143. Springer-Verlag, 2000.
13. A. S. Krishna, D. C. Schmidt, K. Raman, and R. Klefstad. Enhancing real-time CORBA predictability and performance. In *International Symposium on Distributed Objects and Applications*, volume 2888 of *LNCS*, pages 1092–1109, 2003.
14. Object Management Group. The common object request broker: Architecture and specification revision 3.0.2, Dec. 2002.
15. Orbacus. http://www.orbacus.com.
16. ORBit2. http://orbit-resource.sourceforge.net.
17. F. M. Pereira, M. T. Valente, R. Bigonha, and M. Bigonha. Arcademis: A framework for object oriented communication middleware development. *Software Practice and Experience*, 2005. to appear.
18. R. Pratap. Efficient customizable middleware. Master's thesis, Department of Computer Science and Engineering, Washington University, 2003.

19. M. Román, F. Kon, and R. Campbell. Reflective middleware: From your desk to your hand. *Distributed Systems Online*, 2(5), July 2001.
20. D. C. Schmidt and C. Cleeland. Applying patterns to develop extensible and maintainable ORB middleware. *IEEE Communications*, 37(4):54 – 63, 1999.
21. D. C. Schmidt, N. Wang, and S. Vinoski. Object interconnections collocation optimizations for CORBA. *SIGS C++ Report*, 10(9), 1999.
22. A. Singhai, A. Sane, and R. H. Campbell. Quarterware for middleware. In *18th International Conference on Distributed Computing Systems (ICDCS)*, pages 192–201. IEEE Computer Society, 1998.
23. S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with AspectJ. In *17th ACM Conference on Object-Oriented programming systems, languages, and applications*, pages 174–190. ACM Press, 2002.
24. A. Wollrath, R. Riggs, and J. Waldo. A distributed object model for the Java system. In *2nd Conference on Object-Oriented Technologies & Systems*, pages 219–232. USENIX, 1996.
25. C. Zhang, D. Gao, and H.-A. Jacobsen. Towards just-in-time middleware architectures. In *4th International Conference on Aspect-Oriented Software Development*, pages 63–74. ACM Press, 2005.
26. C. Zhang and H.-A. Jacobsen. Quantifying aspects in middleware platforms. In *2nd International Conference on Aspect-Oriented Software Development*, pages 130–139. ACM Press, 2003.
27. C. Zhang and H.-A. Jacobsen. Refactoring middleware with aspects. *IEEE Transactions Parallel and Distributed Systems*, 14(11):1058–1073, 2003.
28. C. Zhang and H.-A. Jacobsen. Resolving feature convolution in middleware systems. In *19th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 188–205. ACM Press, 2004.

Using AOP to Customize a Reflective Middleware

Nélio Cacho and Thaís Batista

Federal University of Rio Grande do Norte,
Informatics Department (DIMAp),
Campus Universitário – Lagoa Nova – 59.072-970 - Natal – RN- Brazil
cacho@consiste.dimap.ufrn.br, thais@ufrnet.br

Abstract. In this paper we present Aspect Open-Orb, an aspect-oriented reflective middleware platform that is synthesized according to the application requirements. It is based on the Open-ORB component model and follows the idea that the middleware functionalities must be driven by the application code and needs. The synthesis process is supported by a reflective middleware implementation and by an aspect-oriented infrastructure composed of AspectLua, a Lua extension that handles AOP. We also present a comparison between the performance of Open-ORB and Aspect Open-Orb.

1 Introduction

Middleware platforms have been widely used as an underlying infrastructure to the development of distributed applications. They provide distribution and heterogeneity transparency and a set of common services such as names, security and fault tolerance. Typically, they are structured as a monolithic architecture that includes a lot of features in order to satisfy a wide range of applications. Traditional middleware such as CORBA[1], COM+[2] and Java RMI[3] are examples of monolithic architectures. They were designed and implemented to provide a wide range of functionalities to different kind of application domains. This broad range of functionalities has increased the popularity of such platforms but, on the other hand, it has increased the size and complexity of the middleware.

Customization mechanisms can be used to overcome this problem by composing a middleware platform according to the requirements of the applications. Among the techniques commonly used to customize middleware platforms, two are becoming very popular: *aspect-oriented programming (AOP)* and *computational reflection*. AOP [15] applies the principle of *separation of concerns* [4] in order to simplify the complexity of large systems. Using this approach to compose middleware platforms, the middleware core contains only the basic functionalities. Other functionalities that implement specific requirements of the applications are inserted in the middleware by the weaver process, when they are required. Computational reflection [18] is the ability of a system to inspect and to manipulate its internal implementation. It divides a system into the meta-level and the base-level. The middleware core is represented by base-objects and new functionality is inserted by meta-objects. Meta-object protocols [20] provide an interface to support the base/meta objects communication.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1133 – 1150, 2005.

In this paper we combine these two approaches in order to customize a middleware platform. We combine a reflective middleware, Open-ORB [11], and a dynamic weaving process in order to dynamically customize an aspect-oriented middleware platform: *Aspect Open-Orb*. This middleware is customized according to the application requirements.

Aspect Open-Orb was implemented using a dynamically typed language with reflective facilities, the Lua [5] language, and an extension of this language, AspectLua [6], that handles AOP. Lua was chosen because it is interpreted and dynamically typed and due to its reflective facilities that provide flexibility for dynamic adaptation [7, 8]. In addition, the availability of AspectLua and LOpenOrb[9], a Lua implementation of the reflective middleware Open-ORB, has facilitated the implementation of Aspect Open-Orb.

This paper is structured as follows. Section 2 presents background on middleware customization. Section 3 presents Aspect Open-Orb including its architecture and an example that illustrates its customization. Section 4 discusses about the performance evaluation of this new architecture. Section 5 contains the related work. Section 6 presents the final remarks.

2 Middleware Customization

The customization of a middleware platform consists of composing it according to the functionalities required by the applications that run on top of the platform. This is a dynamic process that consists of adding new functionalities as well as of removing unnecessary functionalities. We distinguish *decreased* and *increased customization*. *Decreased customization* is a strategy applied to remove functionalities of the middleware in order to customize it. In this case the original middleware provides a broad range of functionalities that are removed in order to tailor the middleware to a given application or environment. On the other hand, *increased customization* is applied when the customization is grounded via the insertion of new functionalities.

Decreased customization is commonly used in traditional middleware platforms such as CORBA, COM+ and Java RMI. The original implementation of such middleware provides a set of functionalities to multiple application domains. In order to reduce the size of the middleware or to tailor it to a given application, it is necessary to remove unessential functionalities.

Increased Customization is commonly used by the *next generation middleware platforms* [10, 11, 12]. These middleware platforms were designed to overcome the limitations of monolithic architectures. Their goal is to offer a small core and to use computational reflection to insert new functionalities. Increased customization is also used by the traditional middleware when new functionalities are needed.

We also distinguish customization according to the moment that it is implemented: *pre-postulated* and *Just-in-time* [32]. *Pre-postulated customization* tailors the middleware before knowing its applications. This process tries to identify the general requirements of possible future applications and defines the middleware configuration that will be used by the applications. On the other hand, *Just-in-time customization* occurs at runtime by identifying the requirements of the running application and customizing the middleware according to the application needs.

The different customization types (*decreased/increased* and *pre-postulated/Just-in-time*) are implemented using a diverse number of techniques such as composition filters [13], hyper-spaces [14], AOP and computational reflection. These techniques share the idea of separating the middleware basic functionalities and the additional functionalities such as non-functional aspects. However, the *crosscutting concerns* [15] represent a puzzling problem in this context. They are elements that are spread around the basic functionalities but that in many cases represent non-functional requirements that must not compose the main middleware core. In order to address this problem, it is necessary to apply a *separation of concerns* [16] strategy that deals with the modularization of crosscutting concerns and separate them from the middleware core.

The next subsections give detail about the use of AOP and computational reflection in the customization of a middleware platform.

2.1 Customization Using Reflective Middleware

The *next generation middleware* [11, 17] exploits computational reflection [18] to customize the middleware architecture. Reflection is used to monitor the middleware internal (re)configuration [19]. The middleware is divided in two levels: *base-level* and *meta-level*. The base-level represents the core. The meta-level contains the building blocks responsible for supporting reflection. These levels are connected to allow that modifications at the meta-level to be reflected into corresponding modifications at the base-level. Thus, modifications at the core should be reflected at the meta-level. The elements of the base-level and of the meta-level are respectively represented by base-level objects and meta-level objects.

Fig. 1 shows that the meta-level is orthogonal to the middleware and to the application. This separation allows the customization of the middleware via the extension of the meta-level. A *causal connection* associates the base-objects with the meta-objects and guarantees that changes to the meta-level are reflected into corresponding changes to the base-level and vice-versa.

In order to illustrate the use of reflection in the customization of a middleware, we have implemented a case study that introduces security communication in a standard binding. The middleware system we used is LOpenOrb [9], a Lua implementation of the Open-ORB component model [21]. Fig. 2 illustrates the elements that compose the LOpenOrb architecture. They are organized in a layered style where the upper layers depend on the lower layers and each element provides an API.

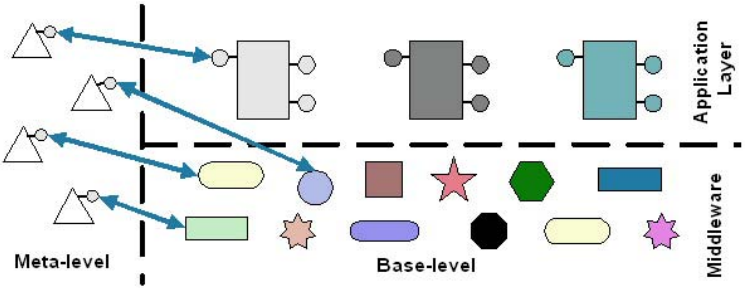


Fig. 1. Reflective Model

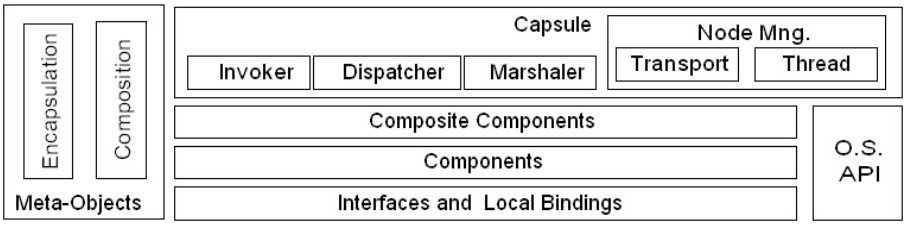


Fig. 2. LOpenORB Architecture

```
1 require "LOpenOrb"
2 LOpenOrb.init(arg)
3 local_bank = LOpenOrb.IRef({},
                {},{"deposit"})
4 remote_bank = LOpenOrb.localcapsule:
                        getRemoteInterfaceByFile("./bank_int.ref")

5 LOpenOrb.localBind({local_bank},{remote_bank})
6 local_bank:deposit(50)
```

Fig. 3. Using LOpenOrb

The functionality of some methods provided by the LOpenOrb API is illustrated in Fig. 3. This figure shows the steps to invoke the *deposit* method implemented in a remote server. The two first lines load and start LOpenOrb. On the next two lines *local_bank* and *remote_bank* interfaces are defined. Interfaces are access points of components. Each interface can export and/or import methods. Exported methods correspond to the provided services. Imported methods are required services. In Fig. 3 the *local_bank* interface imports the *deposit* method while the *remote_bank* interface use the local container (local *Capsule* in Open-ORB terminology) to get the remote reference of the *bank* interface. This reference is stored in the *bank_int.ref*. file. After obtaining the reference, a local *binding* is defined between the two interfaces. The role of the local *binding* is to associate compatible interfaces. Finally, the example illustrates the remote invocation of the *deposit* method.

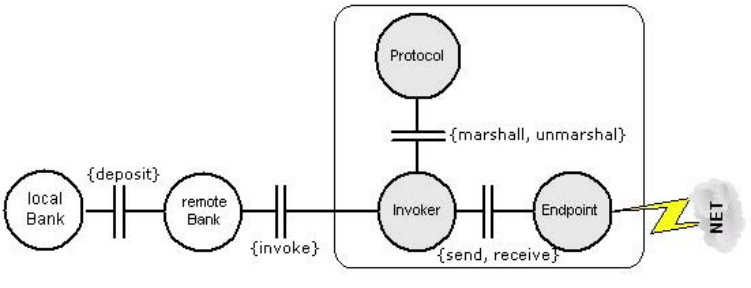


Fig. 4. Invocations details in the LOpenOrb

The internal invocation of the *deposit* method is illustrated in Fig. 4. The *local_Bank* interface invokes the *deposit* method of the *remote_Bank* interface that acts as a proxy that forwards this invocation to the *invoke* method provided by the *Invoker* element. This element marshals the parameters, via the *Protocol* component, and sends the data to the remote interface by using the *send* method offered by the *Endpoint* component. The *receive* method is used to receive replies. The results are sent to the *Protocol* component to unmarshal the data. Then the results of the *deposit* invocation are sent to the remote and local interfaces.

```
1 local cryptProtocol = {}
2 function cryptProtocol:encrypt(stream) ... end
3 function cryptProtocol:decrypt(stream) ... end
4 cryptProtocol.intProtocol = LOpenOrb.IRef({},{},
                                    {"marshall","unmarshal"})
5 function cryptProtocol:marshall( ...)
6      local result = self.intProtocol:marshall(arg)
7      return self:encrypt(result)
8 end
9  function cryptProtocol:unmarshal(...)
10    local result = self:decrypt(arg)
11    return self.intProtocol:unmarshal(result)
12 end
13 cryptProtocol.intCryptProtocol = LOpenOrb.IRef(cryptProtocol,
                 {"marshall","unmarshal"},{})
14 local metaobj = LOpenOrb.metamodel:composition(local_bank)
15 local graph = metaobj:inspect()
16 metaobj:break("use_protocol", "provide_protocol")
17 metaobj:addIIF("use_crypt_protocol", cryptProtocol.intProtocol)
18 metaobj:addIIF("prov_crypt_protocol", cryptProtocol.intCryptProtocol)
19 metaobj:bind("provide_protocol", "use_crypt_protocol")
20 metaobj:bind("use_protocol", "prov_crypt_protocol")
```

Fig. 5. LOpenOrb customization

An example of a customization that includes a cryptograph protocol to encrypt the messages sent to the remote interface is illustrated in Fig. 5. Initially there is the definition of the methods to encrypt and decrypt the messages. On line 4 the *intProtocol* interface is created to use the *marshal* and *unmarshal* methods provided by the original *Protocol* component. Lines 6 and 11 illustrate the use of these two methods. The result of the marshal process is encrypted (line 7). The result of the decrypt process (shown on line 10) is unmarshaled and sent back to the *Invoker* component. In order to provide the *marshal* and *unmarshal* methods with support for cryptograph, the *intCryptProtocol* interface exports these two methods. After the definitions, the insertion of this new functionality in the middleware starts. Initially it is necessary to obtain the meta-object that represents the link between the *local_bank* interface and the other interfaces. In order to make the meta-level easier to use, Open-ORB provides four different meta-models: *encapsulation, composition, environment* and *resource*. Each meta-model is accessed via a meta-object that supports specific aspects of the base-level elements. In this customization, the *composition* meta-object is used. It is responsible for representing the links between interfaces and components. On line 14

the meta-object is obtained and on the next line the *inspect* method is invoked. It returns all components and interfaces connected to the *local_bank* interface. The goal of this method is to provide transparency to the programmer. He/She can include new functionalities without the need of knowing the source code of the middleware. Based on the component graph, the programmer can identify the insertion point and determine the proper procedure, as illustrated on Line 16 where links are broken (between the *use_protocol* interface of the *Invoker* component and the *provide_protocol* of the *Protocol component). use_crypt_protocol* and *prov_crypt_protocol* interfaces are inserted in the component graph to allow the establishment of a link with the *provide_protocol* and *use_protocol* interfaces. With these new links the customization process ends and the customized middleware supports cryptography.

The customization illustrated by this example can be classified as increased and pre-postulated customization, since it inserts new functionalities specified by a configuration file. It does not implement an automatic recognition process. Although the new functionality is dynamically inserted, this issue was not identified by the middleware platform. It was determined by the user by invoking the configuration file.

LOpenOrb, as well as other reflective middleware [30], supports *decreased customization*. For instance, the *metaobj:remove* method is used to remove a component. A limitation of this customization type is that it is not applied to all basic elements of the middleware. It is not applied to the elements that support reflection.

2.2 Customization Using AOP

Aspect-Oriented Programming (AOP) implements *separation of concerns* by decoupling concerns related to components from those related to aspects that crosscut components. The composition of the two concerns is implemented by a weaving process.

The advantage of this approach is to separate the non-functional requirements and to make their manipulation easier and with no impact to the basic code. The two types of code are not tangled and spread. Thus, AOP supports aspect isolation, composition and reuse. The two codes are combined via a weaving process that, depending on the implementation, can take place at compile time or at runtime. In AspectJ [22] and AspectC++ [23] this process occurs at compile time and the aspect are defined via new statements that are included in the language syntax. The weaver is a compiler that receives the system source code and the aspect code and generates an integrated version of the system.

Another strategy includes the insertion of aspects at runtime. AspectLua [5], AOPy [24] and AspectR [25] are languages that follow this approach. They handle AOP using the orignal syntax of the language - Lua, Python [26] and Ruby[27], respectively. No new elements are needed.

Although there is no consensus about the terminology of the elements that make part of AOP, we refer in this work the terminology used in AspectJ because it is the most traditional aspect-oriented language. *Aspects* are the elements designed to encapsulate crosscutting concerns and take them out of the functional code. *Join Points* are well-defined points in the execution of a program. An *advice* defines code that runs at join points. It can run at the moment a joint point is reached and before the method begins running (before), at the moment control returns (after) and during the execution of the joint point (around).

In the context of customizing middleware platforms, AOP is used to implement both increased and decrease customization. For these two processes, it is necessary to know the structure and functioning of the architecture. This knowledge is used to define the *join points* and the *advice*. In case of decreased customization, the study of the target architecture goes beyond the limits of the join points and advice definitions. The definition of these elements is preceded by *aspect mining* [31] and *re-factoring* [29] of the architecture.

According to [31], *aspect mining* consists of capturing and analyzing tangled code in legacy implementations. Thus, *aspect mining* tries to identify the *crosscutting concerns* and to define the separation between the middleware core and the aspect part. In general the separation between the basic functionalities and the aspects is done by a re-factoring process that defines three groups of files: basic functionalies, aspects and weaving. In order to compose the information of the two first files the weaving file defines a set of rules to perform the composition.

In order to exemplify the use of AOP to implement decreased customization we apply *aspect mining* and *re-factoring* strategies in LOpenOrb. The *aspect mining* process identified different aspects:

- *Invocation Types*: in addition to the traditional invocation/return, LOpenOrb also implements an *oneway* invocation that uses a *best-effort* strategy and avoids blocking the client application. The asynchronous invocation is also a non-blocking invocation that returns an object that can be used to capture the values returned by the invoked method.
- *Bindings Types*: three types of bindings are provided [28]: *Operational, Stream* and *Signal*. Each binding has specific features and are commonly used by specific applications that need to send and to control remote invocations, data streams and signals, respectively. Each one implements specific protocols to manage buffer size, synchronism, etc.
- *Reflection*: although the reflection model promotes the *separation of concerns* through the use of base and meta objects, it also introduces *crosscutting* concerns that act on the implementation of the causal connection.

The identification of aspects is followed by a re-factoring process that separates the basic functionalities of LOpenOrb from the elements identified as *crosscutting concerns*. Based on these elements, we demonstrate the re-factoring process via the removal of the code related to the reflective mechanisms illustrated in Fig. 6. In this figure, interface A invokes the *deposit* method of interface B via the *execute_method* of the local binding represented by the *BindCtrl* class. The goal of *execute_method* is to forward each invocation to the target interface. In addition, this method also contains code responsible for maintaining the causal connection.

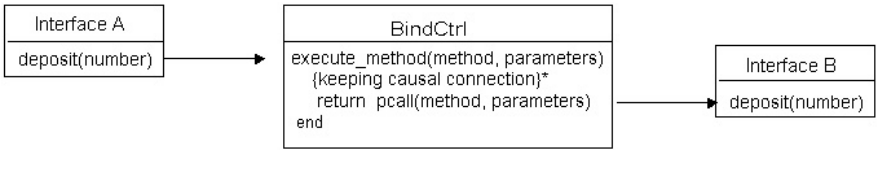


Fig. 6. *Crosscutting concerns* in BindCtrl

To perform the re-factoring we separate the code related to reflection. Later, it is composed using AspectLua [6], an extension of the Lua language for definition of aspects, pointcuts and advice. The AspectLua elements are defined in a weaving file using the Lua features and no special commands are needed. AspectLua defines an *Aspect* object that handles all aspect issues. To use AspectLua, it is necessary to create an instance of the *Aspect* class by invoking the *new* function. After creating a new instance, it is necessary to define a Lua table containing the aspect elements (name, pointcuts, and advice) divided in the follow parameters:

- The first parameter of the *aspect* method is the aspect name;
- The second parameter is a Lua table that defines the pointcut elements: its name, its designator and the functions or variables that must be intercepted. The designator defines the pointcut type. AspectLua supports the following types: *call* for function calls; *callone* for those aspects that must be executed only once; *introduction* for introducing functions in tables (objects in Lua); and *get* and *set* applied upon variables. The *list* field defines functions or variables that will be intercepted. It is not necessary that the elements to be intercepted have been already declared. This list can use wildcards. For instance, *Bank.** means that the aspect should be applied for all methods of the *Bank* class;
- Finally, the third parameter is a Lua table that defines the advice elements: the type (after, before, and around) and the action to be taken when reaching the pointcut.

An example of this process is illustrated in Fig. 7, where the separation of the resulting re-factoring is divided in: (1) *primary program* file, containing the basic functionalities of the ORB; (2) *aspect implementation* file that defines the reflective functionality; (3) *weaving* file that defines how the aspect regarding reflection is inserted in the *primary program*. With this separation, the aspect relative to reflection is maintained out of the middleware basic code and also out of the weaving file code. When needed, they are dynamically inserted. This is an example of decreased customization, where a tangled code is separated and composed via AOP.

Primary Program File

```
BindCtrl ={}
                              . . .
function BindCtrl:execute_method(method, parameters)
   return call(method, parameters)
end
```

Aspect Implementation File

```
function keeping_causal_connection(arg) . . .  end
```

Weave File

```
asp = Aspect:new()
asp:aspect( {name = 'reflect'},
   {pointcutname = 'pre', designator = 'call',
   list = {'BindCtrl.execute_method'}},{type = 'before',
   action = keeping_causal_connection} )
```

Fig. 7. Removing a *crosscutting concern: reflection*

```
1  function Ad_encrypt(...)
2      local result = proceed(arg)
3      return encrypt(result)
4  end
5  function Ad_decrypt(stream)
6    local result = decrypt(arg)
7    return proceed(result)
8  end
9  asp = Aspect:new()
10 asp:aspect( {name = 'cryptaspect'},
   {pointcutname = 'crypt', designator = 'call',
   list ={'Protocol.marshall'}},{type = 'around', action = Ad_encrypt})
11 asp:aspect( {name = 'decryptaspect'},
   {pointcutname = 'decrypt', designator = 'call',
   list={'Protocol.unmarshal'}},{type = 'around', action = Ad_decrypt})
```

Fig. 8. Decreased AOP customization

The use of AOP in the customization of a middleware platform includes the insertion of services not originally provided by the standard implementation. To demonstrate this issue, we use AOP to introduce the cryptograph service represented in Fig. 5. In a different way of the customization illustrated in the previous section, where the meta-object protocol is the mechanism used and where the functionality was inserted in an interface, in this one, the user must know the components and their classes in order to insert the *Ad_encrypt* and *Ad_decrypt* as *advice.* The join points are represented by the *marshall* and *unmarshal* methods of *Protocol* class.

Fig. 8 shows the definition of these two functions and their join points. These functions act in a similar way of those defined in Fig. 5. The difference is in the way they invoke the marshall and unmarshal functions. In this case they are invoked by the proceed function. This function is provided by AspectLua. On lines 10 and 11, the aspects are defined to act in the join points Protocol.marshall and Protocol.unmarshal. In this example of customization, Fig. 8 mixes the code of the aspect file (line 1 to 8) and of the weaving file (lines 9 to 11). In this work we prefer to use the approach illustrated in Fig. 7, where the aspect code and the weaving code are separated in different files. We consider that this separation facilitates the implementation of an automatic weaving process (described in the next section).

3 Combining AOP and Reflection to Middleware Customization

According to the examples illustrated in the previous section, computational reflection is an efficient and simple way of inserting new functionalities in a reflective middleware. It uses a meta-object protocol to abstract away the implementation details. Thus, it is necessary only to know components and interfaces. On the other hand, decreased customization is limited to some elements of the application itself. The other approach, using AOP, requires a broader knowledge of the structures and the application functionality. However, it supports the decreased customization in a broader sense and as a consequence, it allows the removal of more *crosscutting concerns*. The two approaches share the limitation of requiring the specification of the middleware functionality. They

do not support the runtime identification of the application needs and the dynamic customization of the middleware according to the application requirements.

In order to address this lack of customizability support we use AOP to make it possible to customize the reflective middleware LOpenOrb. The goal is to avoid resource wasting and to improve dynamic adaptation. This approach targets two common problems of middleware platforms. The first one is related to the complexity of providing customized middleware implementations by separating basic code and croscutting concerns. The second one is related to the dynamic evolution of middleware platforms. The insertion of new functionalities must be controlled in order to avoid tangled code.

In this work we consider that the application needs can be represented in two ways: (1) in the application code, such as asynchronous invocations; (2) in configuration files, such as security policies. To handle the needs expressed in the application code we use AOP that supports fine-grained identification, insertion and removal of crosscutting concerns. Aspects that are not in the application code can be dynamically inserted using the meta-object protocol. The aspect-oriented middleware we propose, named *Aspect Open-Orb,* allows the customization of the middleware according to the application requirements. This infra-structure is based on the idea that the middleware functionalities are defined by the application code. Fig. 9 illustrates the architecture of Aspect Open-Orb infrastructure. *LOpenOrb Aspects Dependencies* defines the dependencies between aspects and types of method invocations. Each application is composed by its base code (core) and its aspects (Security, Fault Tolerance, etc). At runtime, AspectLua loads LOpenOrb and weaves files according to the application needs. These dynamically loaded elements compose Aspect Open-Orb. Thus, the Aspect Open-Orb internal architecture is dynamically composed according to the application needs.

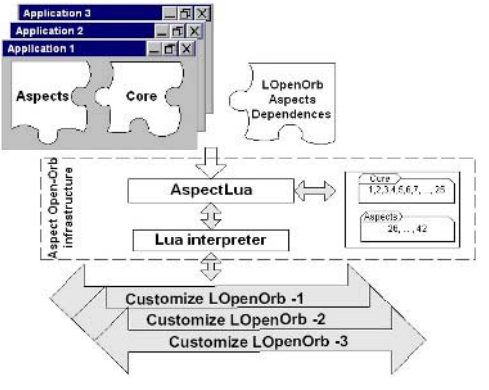


Fig. 9. Aspect Open-Orb infrastructure

The *LOpenOrb Aspects Dependencies* were defined by an *aspect mining* process applied to LOpenOrb. This process has identified the *crosscutting concerns* and the methods provided by the LOpenOrb API used to activate each *crosscutting concern*.

Table 1. Dependence relationships between invocation types and LOpenOrb aspects

Designator	Aspects					
	Reflection	One-way	Asynchronous	Binding	Acceptors	Invokers
LOpenORB.metamodel.composition	X					
LOpenORB.remoteBind				X		
LOpenORB.localcapsule.sendMethod			X			
LOpenORB.localcapsule.announceMethod		X				
LOpenORB.localcapsule.getRemoteInterface						X
LOpenORB.metamodel.encapsulation	X					
LOpenORB.localcapsule.server					X	
LOpenORB.streamBind			X	X		

Table 1 defines some dependencies. For instance, the invocation of *LOpenORB.localcapsule.sendMethod* is associated with the aspect responsible for asynchronous invocations.

After that, the re-factoring process has defined the different files and assigned a number to the aspects shown in Table 1. Note that we consider that different methods can activate a given aspect and that some aspects have common dependencies.

The dependencies relationship between the invoked methods and the loaded files is illustrated in Fig. 10. For each method or set of methods is specified a *join point* that when reached, an advice is invoked to handle it. This advice queries a dependency table that lists all files to be loaded for the execution of the join point. The files are numbered and stored in two folders: the first one contains the basic code of LOpenOrb; the second one contains the weaving files responsible for the insertion of aspects. The purpose of numbering the files is to define which files must be loaded by the loading file process. We avoid the use of long strings for this purpose.

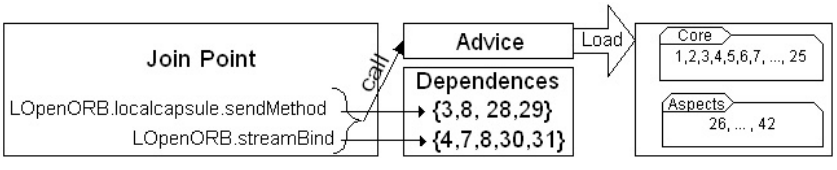


Fig. 10. LOpenOrb Aspects Dependences

```
1 asp = Aspect:new()
2
3 function myInit(arg)
4  LOpenOrb.hostname = arg[2]
5  LOpenOrb.port = arg[3]
6 end
7 asp:aspect({name = 'skipInit'} , {name = 'skip',
        designator='call', list = {'LOpenOrb.init'}},
       {type ='around', action = myInit})
8
9 tblfiles = {}
10 tblfiles[1] = "\core\lopenorb_interface_and_bidinglocal.lua"
11 tblfiles[2] = "\core\lopenorb_component.lua"
12 tblfiles[3] = "\core\lopenorb_composiste.lua"
13  . . .
14
15 tblconf = {}
16 table.insert(tblconf,{key = "LOpenOrb.*IRef", dep = {1} }
17 table.insert(tblconf,{key = "LOpenOrb.localBind*", dep = {1} }
18  . . .
19 table.insert(tblconf,{key = "LOpenOrb.streamBind",
                         dep = {4,7,8,30,31,32,39,40} }
20 table.insert(tblconf,{key = "LOpenOrb.localcapsule.sendMethod",
                         dep=  {3,8,28,29,30,31,45,46} }
21 table.insert(tblconf,{key = "LOpenORB.metamodel.composition",
                         dep = {1,2,3,4,23,24,32,33} }
22 . . .
23
24 function loadfiles( ... )
25   local funcname = table.remove(arg,table.getn(arg))
26   loca contload = 0
27   for k,idxfile in ipairs(searchKey(funcname)) do
28                    local namefile = rawget(tblfiles,idxfile)
29                    if (namefile ~= nil)then
30                    dofile(namefile)
31                    deletefile(idxfile)
32                    contload = contload + 1
33                    end
34   end
35
36   if contload > 0 then
37                  metafunc = LuaMOP:getInstance(funcname)
38                  func = metafunc:getFunction()
39                  return func(unpack(arg))
40   end
41 end
42
43 for k,conf in ipairs(tblconf) do
44  asp:aspect({name = 'ConfORB'} ,
           {name = 'loadfiles', designator = 'callone',
           list = {conf.key}},
           {type ='around', action = loadfiles})
45 end
```

Fig. 11. Middleware aspects configuration

Fig. 10 is implemented by using two sets of aspects. The first set is responsible for intercepting the invocation of the *LOpenORB.init* method. The second one supports the elements defined in Table 1. To illustrate this issue, Fig. 11 shows the code that implements the middleware customization. Lines 3 to 6 contain the definition of *myInit* function. This function, via the aspect defined on line 7, acts on *LOpenORB.init.* All invocations to the *init* function are redirected to the *myInit* function. This avoids the loading of all LOpenOrb library, which is done by the *init* method. As the library is not loaded, the API of the elements is not available.

To maintain the availability of the LOpenOrb API it is necessary to define a second set of aspects that act as *anticipated join points* to the invocations to the LOpenOrb API. Anticipated join points are interception points for elements that have not yet been declared in the application program. The use of anticipated join points makes it possible to intercept an invocation to an undeclared method and to apply to it a specific action, such as, lazy loading a code. Table 1 shows the elements that are defined as anticipated join points. This definition starts on lines 9 to 12 (Fig. 11), which contain the code that creates the list of files - *tblfiles* – indexed by one unique number ID. The next lines insert in *tblconf* some of elements defined in Table 1. This insertion includes two fields: the *Key* field, which contains the aspect name, and the *dep* field, which stores the files needed to invoke the method. Lines 43 to 45 shows the definition of each element in the *tblconf* list as an aspect including an anticipated join point. Each aspect invokes only once (*callone*) the *loadfiles* method when the pointcut defined by *tblconf.Key* is reached. The goal of the *loadfiles* function is to load, according to the function name obtained in Line 25, a loop that returns in the *idxfile* variable each layer needed to support the invocation of the method.

The *dofile* method loads the code of each file. To maintain the loaded methods under control, the *deletefile* function is invoked after loading a method. This function removes the file name from the *tblfiles* list. The *contload* variable is a counter that indicates the number of files already loaded. This is important to avoid a double invocation of the desired method in the case of the following pointcuts: *LOpenOrb.localcapsule.getRemoteInterface* and *LOpenOrb. getRemoteInterfaceByFile*. LuaMOP [33] is used to obtain the desired method that, at this moment, is already defined. Finally, the desired method is invoked on Line 39 and receives as a parameter the arguments of the original invocation (arg).

In order to use Aspect Open-Orb it is necessary to load the file described in Fig. 11 and after that, to load the application code. Fig. 3 shows a simple example of an application code. At runtime, the invocation of the *LOpenOrb.init* method is replaced by the invocation of the *MyInit* method. This method defines the initialization parameters but does not load the LOpenOrb implementation files. According to this situation, the next invocations in Fig. 3 would return an error of missing method. However, as we are using anticipated join points to handle invocations to methods provided by the API, invocations to *LOpenOrb.IRef* or to any other method are forwarded to the proper *advice* that loads the implementation and invokes the target method.

4 Performance Evaluation

In this section we evaluate the performance of Aspect Open-Orb comparing to LOpenOrb in order to identify the impacts of using a dynamic weaving process in the

customization of a middleware specified by a re-factoring process. We performed the experiments on a PC Duron 1.6MHz with 256MB of RAM, and running Linux-Mandrake 9.2.

Table 2 compares the execution time of LOpenOrb and Aspect Open-Orb. For each test type three results are shown: the first one uses the original LOpenOrb; the second and third use Aspect Open-Orb without reflection and with reflection, respectively. As reflection can be seen as a widely spread aspect in Aspect Open-Orb to handle the causal connection, it can be used as a parameter to verify the efficiency of the re-factoring process and how the dynamic weaving of AspectLua can affect the performance of the middleware platform. The efficiency of the re-factoring process is represented in the third row where the comparison with LOpenOrb, the re-factored solution of the Aspect Open-Orb has a superior performance. Thus, as in Aspect Open-Orb, just the code responsible for the functionality is executed, the invocation time is inferior to LOpenOrb, where it is necessary to manage the tangled code.

Table 2. Comparing execution time of LOpenOrb and Aspect Open-Orb

Tests type	LOpenOrb	Aspect Open-Orb	Aspect Open-Orb with reflection
Creating local binding	25.03 µs	23.55 µs	26.20 µs
Execution through local binding	90.12µs	75.43 µs	92.81 µs
Creating Op. Binding	24.49ms	20.17 ms	24.81ms
Execution through Op. Binding	1.312ms	1.157 ms	1.319ms

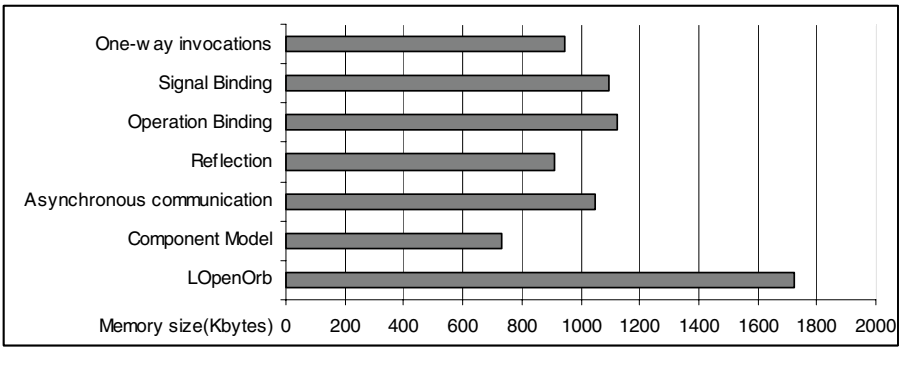


Fig. 12. Memory size usage

Regarding the impact of using AspectLua, the time is slightly superior to those of LOpenOrb. We conclude that the use of a dynamic weaving approach does not impact the performance.

Another key factor of the middleware customization is the reduction of memory usage for some configurations. Fig 12 shows, for six types of different configurations,

the memory usage. The minimal configuration of Aspect Open-Orb corresponds to the Component Model. In this configuration the component model (Interfaces, local binding, Component) is the only element that composes the middleware. No remote communication is handled. Based on this initial configuration other configurations are defined: one-way invocation, Signal Binding, Operational Binding, Reflection and Asynchronous communication. Comparing these configurations with the memory obtained by the use of LOpenOrb, it is clear the reduction of memory obtained by the customization of Aspect Open-Orb.

5 Related Work

In section 2 we discussed examples involving LOpenOrb and AspectLua, how the different types of customization can be applied using AOP or computational reflection. In this section we comment different works that support middleware customization.

DynamicTAO [30] and Open-ORB [11] are the most popular examples of middleware platforms that use computational reflection. DynamicTAO is a reflective middleware based on CORBA that supports dynamic reconfiguration. Each Orb instance contains a component configurator named TAOConfigurator. This component contains hooks that implement the ORB customization. The Open-ORB architecture, as discussed in section 2, defines four distinct meta-object protocols that reify specific aspects of a middleware platform. The customization of these two middleware is pre-postulated and does not affect the internal elements of the middleware such as the TAOConfigurator and the elements that maintain the causal connection in Open-ORB. Thus, the customization is determined by configuration files that specify the insertion/removal of hooks or invocations to the MOP, as illustrated in Fig. 5.

Component Frameworks [35, 36] are used to build customized middleware architectures that abstract away, via the use of components, the basic functionalities and allow the specialization and extension according to application requirements. The main drawback of Component Frameworks is to restrict the customization to the introduction/removal of pluggable aspects specified by configuration files.

AOP is used in some works to support increased customization as discussed in, [33, 34]. In these works AOP is used to introduce event, transaction and fault tolerance services that implement non-functional requirements specific to some applications. On the other hand, decreased customization is supported in [29,31] that use aspect mining and re-factoring techniques to identify and separate the basic functionalities from the non-functional aspects. The combination of these two parts is statically implemented using AspectJ.

Another approach is described in [32]. It applies just-in-time customization. Initially, the acquisition process analyses the application source code and the user preferences in order to obtain the application requirements. Then, the result of the acquisition phase is verified by using dependencies rules and composition constraints. After that the middleware is customized using a compiled approach. This static customization is different from our dynamic approach that allows the runtime customization of the middleware platform according to the application requirements.

6 Final Remarks

In this work we presented Aspect Open-Orb, a highly customized middleware platform that supports just-in-time customization as well as increased and decreased customization.

In order to support the high degree of customization we combined two approaches commonly used separately: AOP and computational reflection. Most of the customized middleware platforms use one of these techniques. We discussed how customization is implemented using each technique separately. Then, we presented our approach to combine the two techniques to customize Aspect Open-Orb.

We also presented the performance evaluation of Open-ORB and of Aspect Open-Orb. The comparisons of their performance evaluation lead us to conclude that the two paradigms work well together without major performance degradation.

We discussed about related work including researches that exploit AOP to customize middleware platforms. While these researches rely on static approaches, our dynamic approach supports the runtime customization of Aspect Open-Orb according to the application requirements.

References

1. OMG Common Object Request Broker Architecture: Core Specification Technical Report Revision 3.0.3. (2004)
2. Morgenthal, J. P.: Microsoft COM+ Will Challenge Application Server Market. (1999). Available at: <http://www.microsoft.com/com/wpaper/complus-appserv.asp>.
3. Wollrath, A., Riggs, R. and Waldo, J.: A distributed object model for the Java system. In: 2nd Conference on Object-Oriented Technologies & Systems (COOTS). USENIX Association, (1996). p.219–232.
4. Parnas, D. L.: On the criteria to be used in decomposing systems into modules. Commun. ACM, v. 15, n. 12, (1972), p.1053–1058.,.
5. Ierusalimsky, R., Figueiredo, L. H., and Celes, W.: Lua – an extensible extension language. Software: Practice and Experience, 26(6), (1996), p.635-652..
6. Cacho, N. and Batista, T. and Fernandes, F. A.: AspectLua – A Dynamic AOP Approach. To appear in a Special Issue of the Journal of Universal Computer Science (J.UCS), 2005.
7. Batista, T. V.; Cerqueira, R.; Rodriguez, N.: Enabling reflection and reconfiguration in CORBA. In: In Workshop Proceedings of the International Middleware Conference. (2003). p. 125–129.
8. Batista, T. V.; Rodriguez, N.: Dynamic reconfiguration of component-based applications. In: Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE). (2000), p. 32-39.
9. Cacho, N, Batista, T.: Adaptação Dinâmica no Open-Orb: detalhes de implementação Proceedings of the 23th Brazilian Symposium on Computer Networks (SBRC'2005), (v.1), SBC, Fortaleza, CE, Brazil, May (2005), p. 495-508.
10. Agha, G. A.: Adaptive middleware. Commun. ACM, ACM Press, v. 45, n. 6, , (2002), p. 31–32
11. Blair, G. et al.: An architecture for next generation middleware. In: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing. London: Springer-Verlag, (1998).

12. Tripathi, A.: Challenges designing next-generation middleware systems. Commun. ACM, ACM Press, v. 45, n. 6, (2002), p. 39–42.
13. Bergmans, L. and Aksit, M.: Aspects & crosscutting in layered middleware systems. In: International Middleware Conference, Workshop Proceedings. New York, USA, (2000).
14. Ossher, H. and Tarr, P.: "Multi-dimensional separation of concerns and the hyperspace approach," in Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, Kluwer Academic Publishers, April (2000).
15. Kiczales, G. et al.: Aspect-oriented programming. In: Proceedings European Conference on Object-Oriented Programming. Berlin, Heidelberg, and New York: Springer-Verlag, (1997). v. 1241, p. 220–242.
16. Dijkstra, E. A: Discipline of Programming. Prentice-Hall, (1976).
17. Kon, F. et al. The case for reflective middleware. Commun. ACM, ACM Press, v. 45, n. 6, (2002), p. 33–38.
18. Smith, B. C.: Procedural Reflection in Programming Languages. These (Phd) — Massachusetts Institute of Technology, (1982).
19. Roman, M., F. Kon, and R.H. Campbell.: Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online Journal, 2(5), (2001).
20. Kiczales, G.; Rivières, J. and Bobrow, D.: The Art of the Metaobject Protocol. MIT Press, (1991).
21. Andersen, A.; Blair, G. S.; Eliassen, F.: A reflective component-based middleware with quality of service management. In: PROMS 2000, Protocols for Multimedia Systems. Cracow, Poland, (2000).
22. Kiczales, G., Hilsdale, E., Hugunin, J. et al.: An Overview of AspectJ. In: Lecture Notes in Computer Science (LNCS), Vol. 2072, (2001), p. 327-355.
23. Gal, A.; Schroder-Preikschat, W.; Spinczyk, O.: AspectC++: Language Proposal and Prototype Implementation. University of Magdeburg, (2001).
24. Dechow, D. R.: Advanced separation of concerns for dynamic, lightweight languages. In: Generative Programming and Component Engineering. (2003).
25. Bryant, A.; Feldt, R.: AspectR - Simple aspect-oriented programming in Ruby. 2002. Available at: <http://aspectr.sourceforge.net/>.
26. Drake, F. L.: Python Reference Manual. 2003. Available at: <http://www.python.org/doc/current/ref/ref.html>.
27. Thomas, D.; Fowler, C.; Hunt, A.: Programming Ruby: A Pragmatic Programmer's Guide. 2000. Available at: <http://www.rubycentral.com/book/>.
28. Fitzpatrick, T. et al.: Supporting adaptive multimedia applications through open bindings. In: Proceedings of the International Conference on Configurable Distributed Systems. [S.l.]: IEEE Computer Society, (1998). p. 128.
29. Zhang, C. and Jacobsen, H.: Re-factoring Middleware with Aspects. IEEE Transactions on Parallel and Distributed Systems, Vol. 14, (2003), p. 1243-1262.
30. Kon, F. et al.: Monitoring, security, and dynamic configuration with the dynamic Tao reflective orb. In: IFIP/ACM International Conference on Distributed systems platforms. Springer-Verlag New York, Inc., (2000), p. 121–143.
31. Zhang, C. and Jacobsen, H. A.: Quantifying aspects in middleware platforms. In: Proceedings of the 2nd international conference on Aspect-oriented software development. [S.l.]: ACM Press, (2003), p. 130–139.
32. Zhang, C.; Gao, D. and Jacobsen, H. A.: Towards just-in-time middleware architectures. In: Proceedings of the 4th international conference on Aspect-oriented software development. ACM Press, (2005). p. 63–74.

33. Fernandes, F.; Batista, T. and Cacho, N.: Exploring Reflection to Dynamically Aspectizing CORBA-based Applications. In: Proceedings of the 3nd Workshop on Reflective and Adaptative Middleware. ACM Press, (2004). p. 220-225 .
34. Herrero, J.L., Sánchez, F., Toro, M. Fault tolerance AOP approach. In: Workshop on Aspect-Oriented Programming and Separation of Concerns, Lancaster, (2001).
35. Singhai A., Sane A. and Campbell R. Quarterware for middleware. In Proceedings of the 18th IEEE International Conference on Distributed Computing Systems (ICDCS), May, (1998).
36. Truyen E. et al, Aspects for Run-Time Component Integration. Workshop on Aspects and Dimensions of Concern at ECOOP'2000, Cannes, France, June (2000).

ODBASE 2005 PC Co-Chairs' Message

Welcome to the Fifth International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2005). This year's ODBASE conference is being held in Agia Napa, Cyprus, from October 31 till November 4, 2005.

The ODBASE conferences provide a forum for exchanging the latest research results on ontologies, data semantics, and other areas of computing related to the Semantic Web. We encourage participation of both researchers and practitioners in order to facilitate exchange of ideas and results on semantic issues in Web information systems. Towards this goal, we accepted both research and experience papers.

This high-quality program would not have been possible without the authors who chose ODBASE as a venue for their publications. Out of 114 submitted papers, we selected 25 full papers, 7 short papers, and 8 posters. To round up this excellent program, V.S. Subrahmanian has agreed to be our keynote speaker.

We are grateful for the dedicated work of the 65 top experts in the field who served on the program committee. Special thanks goes to the external referees who volunteered their time to provide additional reviews. Finally, we are indebted to Kwong Yuen Lai who was immensely helpful in facilitating the review process and making sure that everything stayed on track.

August 2005

Michael Kifer,
University at Stony Brook
Stefano Spaccapietra,
Swiss Federal Institute of Technology at Lausanne
(ODBASE 2005 Program Committee Co-Chairs)

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, p. 1151, 2005.

Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences

Yuan An[1], Alex Borgida[2], and John Mylopoulos[1]

[1] University of Toronto, Canada
{yuana,jm}@cs.toronto.edu
[2] Rutgers University, USA
borgida@cs.rutgers.edu

Abstract. There are many problems requiring a semantic account of a database schema. At its best, such an account consists of mapping formulas between the schema and a formal conceptual model or ontology (CM) of the domain. This paper describes the underlying principles, algorithms, and a prototype of a tool which infers such semantic mappings when given *simple correspondences* from table columns in a relational schema and datatype properties of classes in an ontology. Although the algorithm presented is necessarily heuristic, we offer formal results stating that the answers returned are "correct" for relational schemas designed according to standard Entity-Relationship techniques. We also report on experience in using the tool with public domain schemas and ontologies.

1 Introduction and Motivation

A number of important database problems have been shown to have improved solutions by using a conceptual model or an ontology (CM) to provide he *precise semantics* of the database schema. These include federated databases, data warehousing [1], and information integration through mediated schemas [7]. (See survey [15].) Since much information on the web is generated from databases (the "deep web"), the recent call for a Semantic Web, which requires a connection between web content and ontologies, provides additional motivation for the problem of associating semantics with data (e.g., [6]). In almost all of these cases semantics of the data is captured by some kind of *semantic mapping* between the database schema and the CM. Although sometimes the mapping is just a *simple* association from terms to terms, in other cases what is required is a *complex* formula, often expressed in logic or a query language.

For example, in both the Information Manifold data integration system presented in [7] and the study of data integration in data warehousing presented in [1], Horn formulas in the form $T(\overline{X})$:- $\Phi(\overline{X}, \overline{Y})$ are used to connect a relational data source to a CM described by some Description Logic, where $T(\overline{X})$ is a single predicate representing a table in the relational data source and $\Phi(\overline{X}, \overline{Y})$ is a conjunctive formula over the predicates representing the concepts and relationships in the CM. In the literature, such a formalism is called local-as-view (LAV).

So far, it has been assumed that *humans* specify the mapping formulas – a difficult, time-consuming and error-prone task. In this paper, we propose a tool that assists users

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1152–1169, 2005.

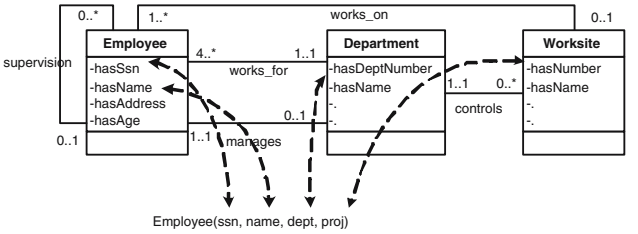


Fig. 1. Relational table, Ontology, and Correspondences

in specifying LAV mapping formulas between relational databases and ontologies. Intuitively, it is much easier for users to draw the *simple correspondences* from the columns of the tables in the database to datatype properties of classes in the ontology – manually or through some existing schema matching tools (e.g., [3, 13]) – than to compose the logic formulas. Given the set of correspondences and following the LAV formalism, the tool is expected to reason about the database schema and the ontology, and to generate a ranked list of candidate Horn formulas for each table in the relational database. Ideally, one of the formulas is the right one capturing the user's intention underlying the specified correspondences. The following example illustrates the input/out behavior of the tool we seek.

Example 1. An ontology contains concepts (classes), attributes of concepts (datatype properties of classes), and relationships between concepts (object properties of classes). Graphically, we use the UML notations to represent the above information. Given the ontology in Figure 1 and a relational table *Employee(ssn, name, dept, proj)* with key ssn, a user could draw the simple correspondences as the arrowed dash-lines shown in Figure 1. Using prefixes $\mathcal{T}$ and $\mathcal{O}$ to distinguish predicates in the relational schema and the ontology, we represent the correspondences as follows:

$\mathcal{T} : Employee.ssn \leftrightsquigarrow \mathcal{O} : Employee.hasSsn$
$\mathcal{T} : Employee.name \leftrightsquigarrow \mathcal{O} : Employee.hasName$
$\mathcal{T} : Employee.dept \leftrightsquigarrow \mathcal{O} : Department.hasDeptNumber$
$\mathcal{T} : Employee.proj \leftrightsquigarrow \mathcal{O} : Worksite.hasNumber$

Given the above input, we may expect the tool generate a mapping formula of the form

$\mathcal{T}$:Employee(ssn, name, dept, proj) :-
$\mathcal{O}$:Employee(x_1), $\mathcal{O}$:hasSsn(x_1,ssn), $\mathcal{O}$:hasName(x_1,name), $\mathcal{O}$:Department(x_2),
$\mathcal{O}$:works_for(x_1,x_2), $\mathcal{O}$:hasDeptNumber(x_2,dept), $\mathcal{O}$:Worksite(x_3), $\mathcal{O}$:works_on(x_1,x_3),
$\mathcal{O}$:hasNumber(x_3,proj). □

An intuitive and naive solution (inspired by early work of Quillian in [12]) gives rise to finding the minimum spanning trees or Steiner trees[1] among the classes that have datatype properties corresponding to table columns and encoding the trees into logic formulas. However, the problem is that a spanning/Steiner tree may not match

[1] A Steiner tree for set M of nodes in graph G is a minimum spanning tree of M that contains nodes of G which are not in M.

the semantics of the given table due to their constraints. For example, consider the relational table $Project(\underline{name}, supervisor)$, with $name$ as its key and corresponding to $\mathcal{O}$:$Worksite.hasName$, plus $supervisor$ corresponding to $\mathcal{O}$:$Employee.hasSsn$ in Figure 1. The minimum spanning tree consisting of $Worksite$, $Employee$, and the edge `works_on` does not match the semantics of table $Project$ because there are multiple $Employee$s working on a $Worksite$. In this paper, we turn to a database design process to uncover the connections between the constraints in relational schemas and ontologies. In contrast to the graph theoretic results which show that there might be too many minimum spanning/Steiner trees between a fixed set of nodes (for example, there are already 5 minimum spanning trees among *Employee, Department*, and *Worksite* in the very simple graph in Figure 1, considering each edge has the same weight,) we propose to generate a limited number of "reasonable" trees and formulas.

Our approach is directly inspired by the Clio project [10, 11], which developed a successful tool that infers mappings from one set of relational tables and/or XML documents to another, given just a set of correspondences between their respective attributes. Without going into further details at this point, we summarize the contributions which we feel are being made here:

- The paper identifies a new version of the data mapping problem: that of *inferring* complex formulas expressing the semantic mapping between relational database schemas and ontologies from simple correspondences.
- We propose an algorithm to find a "reasonable" tree connection in the ontology graph. The algorithm is enhanced to take into account information about the schema (key and foreign key structure), the ontology (cardinality restrictions), and standard database schema design guidelines.
- To gain theoretical confidence, we describe formal results which state that if the schema was designed from a CM using techniques well-known in the Entity Relationship literature (which provide a natural semantic mapping for each table), then the tool will report essentially all and only the appropriate semantics. This shows that our heuristics are not just shots in the dark: in the case when the ontology has no extraneous material, and when a table's schema has not been denormalized, the algorithm will produce good results.
- To test the effectiveness and usefulness of the algorithm in practice, we implemented the algorithm in a prototype tool and applied it to a variety of database schemas and ontologies. Our experience has shown that the user effort in specifying complex mappings by using the tool is significantly less than that by manually writing formulas from scratch.

The rest of the paper is structured as follows. Section 2 discusses related work, and Section 3 presents the necessary background and notation. Section 4 describes an intuitive progression of ideas underlying our approach, while Section 5 provides the mapping inference algorithm. In Section 6 we report on the prototype implementation of these ideas and experience with the prototype. Finally, Section 7 concludes and discusses future work.

2 Related Work

As mentioned earlier, the Clio tool [10, 11] discovers formal queries describing how target schemas can be populated with data from source schemas. The present work could be viewed as extending this to the case when the source schema is a relational database, while the target is a ontology. For example, in Example 1, if one viewed the ontology as a relational schema made of unary tables, e.g., $Employee(x_1)$, $Department(x_2)$, binary tables, e.g., $hasSsn(x'_1, ssn)$, $hasDeptNumber(x'_2, dept)$, $works\_for(x''_1, x''_2)$, and foreign key constraints, e.g., x'_1 and x''_1 referencing x_1, x'_2 and x''_2 referencing x_2, where x_i, x'_i, x''_i $(i = 1, 2)$ are object identifiers available in the ontology, one could in fact try to apply directly the Clio algorithm to it, pushing it beyond its intended application domain. The desired mapping formula from Example 1 would not be produced for several reasons: (i) Clio [11] does not make a so-called logical relation connecting $hasSsn(x'_1, ssn)$ and $hasDeptNumber(x'_2, dept)$, since the chase algorithm of Clio only follows foreign key references *out* of tables. Specifically, there would be three separate logical relations, i.e., $Employee(x_1) \bowtie_{x_1=x'_1} hasSsn(x'_1, ssn)$, $Department(x_2) \bowtie_{x_2=x'_2} hasDeptNumber(x'_2, dept)$, and $works\_for(x''_1, x''_2) \bowtie_{x''_1=x_1} Employee(x_1) \bowtie_{x''_2=x_2} Department(x_2)$. (ii) The fact that *ssn* is a key in the table $\mathcal{T}$:$Employee$, leads us to prefer (see Section 4) a many-to-one relationship, such as *works_for*, over some many-to-many relationship which could have been part of the ontology (e.g., $\mathcal{O}$*:previouslyWorkedFor*); Clio does not differentiate the two. So the work to be presented here analyzes the key structure of the tables and the semantics of relationships (cardinality, IsA) to eliminate *unreasonable* options that arise in mapping to ontologies.

The problem of *data reverse engineering* is to extract a CM, for example, an ER diagram, from a database schema. Sophisticated algorithms and approaches to this have appeared in the literature over the years (e.g., [8, 5]). The major difference between data reverse engineering and our work is that we are given an existing ontology, and want to interpret a legacy relational schema in terms of it, whereas data reverse engineering aims to construct a new ontology.

Schema matching (e.g., [3, 13]) identifies semantic relations between schema elements based on their names, data types, constraints, and schema structures. The primary goal is to find the one-to-one simple correspondences which are part of the input for our mapping inference algorithms.

3 Formal Preliminaries

For an ontology, we do not restrict ourselves to any particular ontology language in this paper. Instead, we use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL, and description logics. In the sequel, we use CM to denote an ontology prescribed by the generic CML. Specifically, the language allows the representation of *classes/concepts* (unary predicates over individuals), *object properties/relationships* (binary predicates relating individuals), and *datatype properties/attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this paper. Concepts are organized in the familiar **is-a** hierarchy. Object properties, and their inverses (which are always present), are subject to constraints such

as specification of domain and range, plus the familiar cardinality constraints, which here allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships). We shall represent a given CM using a directed and labeled *ontology graph*, which has concept nodes labeled with concept names C, and edges labeled with object properties p; for each such p, there is an edge for the inverse relationship, referred to as p^-. For each attribute f of concept C, we create a separate attribute node denoted as $N_{f,C}$, whose label is f, and with edge labeled f from node C to $N_{f,C}$.[2] For the sake of simplicity, we sometimes use UML notations, as in Figure 1, to represent the ontology graph. Note that in such a diagram, instead drawing separate attribute nodes, we place the attributes inside the rectangle nodes. Readers should not be confused by this compact representation.

If p is a relationship between concepts C and D (or object property having domain C and range D), we propose to write in text as `C ---p--- D` (If the relationship p is functional, we write `C ---p->-- D`.) For expressive CMLs such as OWL, we may also connect C to D by p if we find an existential restriction stating that each instance of C is related to *some* or *all* instance of D by p.

For relational databases, we assume the reader is familiar with standard notions as presented in [14], for example. We will use the notation $T[\underline{K}, Y]$ to represent a relational table T with columns KY, and key K. If necessary, we will refer to the individual columns in Y using $Y[1], Y[2], \ldots$, and use XY as concatenation. Our notational convention is that single column names are either indexed or appear in lower-case. Given a table such as T above, we use the notation key(T), nonkey(T) and columns(T) to refer to K, Y and KY respectively. (Note that we use the terms "table" and "column" when talking about relational schemas, reserving "relation(ship)" and "attribute" for aspects of the CM.) A foreign key (fk) in T is a set of columns F that *references* table T', and imposes a constraint that the projection of T on F is a subset of the projection of T' on $\mathsf{key}(T')$.

In this paper, a *correspondence* $T.c \rightsquigarrow D.f$ will relate column c of table T to attribute f of concept D. Since our algorithms deal with ontology graphs, formally a correspondence L will be a mathematical relation $L(T, c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three.

Finally, we use Horn-clauses in the form $T(X)$:- $\Phi(X, Y)$, as described in Introduction, to represent *semantic mappings*, where T is a table with columns X (which become arguments to its predicate), and Φ is a conjunctive formula over predicates representing the CM, with Y existentially quantified as usual.

4 Principles of Mapping Inference

We begin with the set of *concept nodes*, M, such that for each node in M some of the attribute nodes connected to it are corresponded by some of the columns of a table, and M contains all of the nodes singled out by all of the correspondences from the columns of the table. We assume that the correspondences have been specified by users. To seek LAV mapping, it is sufficient to only focus on the connections among nodes in M

[2] Unless ambiguity arises, we will use "node C", when we mean "concept node labeled C".

by stripping off the attribute nodes[3]. Note that attribute nodes, which we can attach them back at any time, are important when encoding trees into formulas for proving the formal results. The primary principle of our mapping inference algorithm is to look for *shortest* "reasonable" trees connecting nodes in M. In the sequel, we will call such a tree *semantic tree*.

As mentioned before, the naive solution of finding min-spanning trees or Steiner trees does not give us good results. The semantic tree we seek is not only shortest but "reasonable". Although the "reasonableness" is vague at this moment, we will lay out some principles according to the semantics carried by the relational schemas and ontologies; and we will show that our principles have a solid foundation that the "reasonableness" can be formally proved in a very strict but useful setting.

Consider the case when $T[\underline{c}, b]$ is a table with key c, corresponding to an attribute f on concept C, and b is a foreign key corresponding to an attribute e on concept B. Then for each value of c (and hence instance of C), T associates at most one value of b (instance of B). Hence the semantic mapping for T should be some formula that acts as a function from its first to its second argument. The semantic trees for such formulas look like functional edges, and hence should be preferred. For example, given table $Dep[\underline{dept}, ssn, \ldots]$, and correspondences which link the two named columns to $hasDeptNumber$ and $hasSsn$ in Figure 1, respectively, the proper semantic tree uses $\texttt{manages}^-$ (i.e., `hasManager`) rather than $\texttt{works\_for}^-$ (i.e., `hasWorkers`).

Conversely, for table $T'[\underline{c, b}]$, an edge that is functional from C to B, or from B to C, is likely not to reflect a proper semantics since it would mean that the key chosen for T' is actually a super-key – an unlikely error. (In our example, consider a table $T[\underline{ssn, dept}, \ldots]$, where both named columns are foreign keys.) To deal with such problems, an algorithm should work in two stages: first connecting the concepts corresponding to key columns into somehow a *skeleton tree*, then connecting the rest nodes corresponding to other columns to the skeleton by, preferably, functional edges.

Most importantly, we must deal with the assumption that the relational schema and the CM were developed independently, which implies that not all parts of the CM are reflected in the database schema and vice versa. This complicates things, since in building the semantic tree we may need to go through additional nodes, which end up not being corresponded by any columns in the relational schema. For example, Consider again the $Project(\underline{name}, supervisor)$ table and its correspondences mentioned in Introduction. Instead of the edge `works_on`, we prefer the *functional path* $\texttt{controls}^-.\texttt{manages}^-$ (i.e., `controlledBy` followed by `hasManager`), passing through node *Department*. Similar situations arise when the CM contains detailed *aggregation* hierarchies (e.g., *city* part-of *township* part-of *county* part-of *state*), which are abstracted in the database (e.g., a table with columns for *city* and *state* only).

We have chosen to flesh out the above principles in a systematic manner by considering the behavior of our proposed algorithm on relational schemas designed from Entity Relationship diagrams — a topic widely covered in even undergraduate database courses [14]. (We call this *er2rel schema design*.) One benefit of this approach will be to allow us to prove that our algorithm, though heuristic in general, is in some sense

[3] In the sequel, we will say "a concept corresponded by some columns of a table" without mentioning its attributes.

"correct" for a certain class of schemas. Of course, in practice such schemas may be "denormalized" in order to improve efficiency, and, as we mentioned, only parts of the CM are realized in the database. We emphasize that our algorithm uses the general principles enunciated above even in such cases, with relatively good results in practice.

To reduce the complexity of the algorithms which is inherently a tree enumeration, and the size of the answer set, we modify the graph by collapsing multiple edges between nodes E and F, labeled $p_1, p_2, \ldots$ say, into a single edge labeled $'p_1; p_2; \ldots'$ The idea is that it will be up to the user to choose between the alternative labels after the final results have been presented by the tool, though the system may offer suggestions, based on additional information, such as heuristics concerning the identifiers labeling tables and columns, and their relationship to property names.

5 Mapping Inference Algorithms

As stated before, the algorithm is based on the relational database design methodology from ER models. We will introduce the details of the algorithm in a gradual manner, by repeatedly adding features of an ER model that appear as part of the CM. We assume that the reader is familiar with basics of ER modeling and database design [14], though we summarize the ideas.

5.1 An Initial Subset of ER Notions

We start with a subset of ER that contains the notions such as *entity set* E (called just "entity" here), with attributes referred as $\mathsf{attribs}(E)$, and *binary relationship set*. In order to facilitate the statement of correspondences and theorems, we assume in this section that attributes in the CM have globally unique names. (Our implemented tool does not make this assumption.) An entity is represented as a concept/class in our CM. A binary relationship set corresponds to two relationships in our CM, one for each direction, though only one is mapped to a table. Such a relationship will be called *many-many* if neither it nor its inverse is functional. A *strong entity* S has some attributes that act as identifier. We shall refer to these using $\mathsf{unique}(S)$ when describing the rules of schema design. A *weak entity* W has instead $\mathsf{localUnique}(W)$ attributes, plus a functional total binary relationship p (denoted as $\mathsf{idRel}(W)$) to an identifying owner entity (denoted as $\mathsf{idOwn}(W)$).

Note that information about general identification cannot be represented in even highly expressive languages such as OWL. So functions like unique are only used while describing the er2rel mapping, and are not assumed to be available during semantic inference. The er2rel design methodology (we follow mostly [8, 14]) is defined by two components: To begin with, Table 1 specifies a mapping $\tau(O)$ returning a relational table schema for every CM component O, where O is either a concept/entity or a binary relationship. In this subsection, we assume that no pair of concepts is related by more than one relationship, and that there are no so-called "recursive" relationships relating an entity to itself. (We deal with these in Section 5.3.)

In addition to the schema (columns, key, fk's), Table 1 also associates with a relational table $T[V]$ a number of additional notions:

Table 1. er2rel Design Mapping

ER Model object O	Relational Table τ(O)	
Strong Entity S	*columns:*	X
	primary key:	K
Let X=attribs(S)	*fk's:*	none
Let K=unique(S)	*anchor:*	S
	semantics:	$T(X)$:- $S(y)$,hasAttribs(y, X).
	identifier:	identify$_S(y, K)$:- $S(y)$,hasAttribs(y, K).
Weak Entity W	*columns:*	ZX
let	*primary key:*	UX
$E =$ idOwn(W)	*fk's:*	X
$P =$ idrel(W)	*anchor:*	W
Z=attribs(W)	*semantics:*	$T(X, U, V)$:- $W(y)$, hasAttribs(y, Z), $E(w)$,$P(y, w)$, identify$_E(w, X)$.
$X =$ key($\tau(E)$)	*identifier:*	identify$_W(y, UX)$:- $W(y)$,$E(w)$, $P(y, w)$, hasAttribs(y, U), identify$_E(w, X)$.
$U =$localUnique(W)		
$V = Z - U$		
Functional Relationship F	*columns:*	X_1X_2
	primary key:	X_1
E_1 --F-> E_2	*fk's:*	X_i references $\tau(E_i)$,
let $X_i =$ key($\tau(E_i)$)	*anchor:*	E_1
for $i = 1, 2$	*semantics:*	$T(X_1, X_2)$:- $E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $F(y_1, y_2)$, $E_2(y_2)$, identify$_{E_2}(y2, X_2)$.
Many-many Relationship M	*columns:*	X_1X_2
	primary key:	X_1X_2
E_1 --M-- E_2	*fk's:*	X_i references $\tau(E_i)$,
let $X_i =$ key($\tau(E_i)$)	*semantics:*	$T(X_1, X_2)$:- $E_1(y_1)$,identify$_{E_1}(y_1, X_1)$, $M(y_1, y_2)$,$E_2(y_2)$, identify$_{E_2}(y2, X_2)$.
for $i = 1, 2$		

– an *anchor*, which is the central object in the CM from which T is derived, and which is useful in explaining our algorithm (it will be the root of the semantic tree);
– a formula for the semantic mapping for the table, expressed as a Horn formula with head $T(V)$ (this is what our algorithm should be recovering); in the body of the Horn formula, the function hasAttribs(x, Y) returns conjuncts $attr_j(x, Y[j])$ for the individual columns Y[1], Y[2],... in Y, where $attr_j$ is the attribute name corresponded by column $Y[j]$.
– the formula for a predicate identify$_C(x, Y)$, showing how object x in (strong or weak) entity C can be identified by values in Y[4].

Note that τ is defined recursively, and will only terminate if there are no "cycles" in the CM (see [8] for definition of cycles in ER).

The er2rel methodology also suggests that the schema generated using τ can be modified by (repeatedly) *merging* into the table T_0 of an entity E the table T_1 of some functional relationship involving the same entity E (which has a foreign key reference to T_0). If the semantics of T_0 is $T_0(K, V)$:- $\phi(K, V)$, and of T_1 is $T_1(K, W)$

[4] This is needed in addition to hasAttribs, because weak entities have identifying values spread over several concepts.

:- $\psi(K, W)$, then the semantics of table T=merge(T_0,T_1) is, to a first approximation, $T(K, V, W)$:- $\phi(K, V), \psi(K, W)$. And the anchor of T is the entity E.

Please note that one conceptual model may result in several different relational schemas, since there are choices in which direction a one-to-one relationship is encoded (which entity acts as a key), and how tables are merged. Note also that the resulting schema is in Boyce-Codd Normal Form, if we assume that the only functional dependencies are those that can be deduced from the ER schema (as expressed in FOL).

Now we turn to the algorithm for finding the semantic trees between nodes in the set M singled out by the correspondences from columns of a table. As mentioned in the previous section, because the keys of a table functionally determine the rest of the columns, the algorithm for finding the semantic trees works in several steps:

1. Determine a skeleton tree connecting the concepts corresponding to key columns; also determine, if possible, a unique anchor for this tree.
2. Link the concepts corresponding to non-key columns using shortest functional paths to the skeleton anchor.
3. Link any unaccounted-for concepts corresponding to some other columns by arbitrary shortest paths to the tree.

More specifically, the main function, getTree(T,L), will infer the semantics of table T, given correspondence L, by returning an semantic tree S. Encoding S into formula yields the conjunctive formula defining the semantics of table T.

Function getTree(T,L)
input: table T, correspondences L for columns(T)
output: set of semantic trees [5]
steps:

1. Let L_k be the subset of L containing correspondences from key(T);
   compute (S', Anc')=getSkeleton(T,L_k).
2. If onc(nonkey(T))[6]- onc(key(T)) is empty, then return (S', Anc'). */*if all columns correspond to the same set of concepts as the key does, then return the skeleton tree.*/*
3. For each foreign key F_i in nonkey(T) referencing $T_i(K_i)$:
   let $L^i_k = \{T_i.K_i \rightsquigarrow L(T, F_i)\}$, and compute (Ss''_i, Anc''_i)= getSkeleton(T_i,L^i_k). */*recall that the function $L(T, F_i)$ is derived from a correspondence $L(T, F_i, D, f, N_{f,D})$ such that it gives a concept D and its attribute f ($N_{f,D}$ is the attribute node in the ontology graph.)*/*
   find π_i=shortest functional path from Anc' to Anc''_i; let $S =$ combine[7]$(S', \pi_i, \{Ss''_i\})$.
4. For each column c in nonkey(T) that is not part of an fk, let $N =$ onc(c); find π=shortest functional path from Anc' to N; update $S :=$ combine(S, π).
5. In all cases above asking for functional paths, use a shortest path if a functional one does not exist.
6. Return S.

[5] To make the description simpler, at times we will not explicitly account for the possibility of multiple answers. Every function is extended to set arguments by element-wise application of the function to set members.

[6] onc(X) is the function which gets the set M of concepts corresponded by the columns X.

[7] Function combine merges edges of trees into a larger tree.

The function getTree(T, L) makes calls to function getSkeleton on T and other tables referenced by fks in T, in order to get a set of (skeleton tree, anchor)-pairs, which have the property that in the case of er2rel designs, if the anchor returned is concept C, then the encoding of the skeleton tree is the formula for identify$_C$.

Function getSkeleton(T,L)
input: table T, correspondences L for key(T)
output: a set of (skeleton tree, anchor) pairs
steps:
Suppose key(T) contains fks F_1,...,F_n referencing tables $T_1(K_1)$,..,$T_n(K_n)$;

1. If $n \leq 1$ and onc(key(T)) is just a singleton set $\{C\}$, then return $(C, \{C\})$.[8] /**Likely a strong entity: the base case.*/*
2. Else, let L_i=$\{T_i.K_i \rightsquigarrow L(T, F_i)\}$/**translate corresp's thru fk reference*/*;
   compute (Ss_i, Anc_i) = getSkeleton(T_i, L_i).
   (a) If key(T) $= F_1$, then return (Ss_1, Anc_1). /**functional relationship of weak entities.*/*
   (b) If key(T)=F_1A, where columns A are not in any foreign key of T then /**possibly a weak entity*/*
      i. if $Anc_1 = \{N_1\}$ and onc(A) $= \{N\}$ such that there is a total functional path π from N to N_1, then return (combine(π, Ss_1), $\{N\}$). /**N is a weak entity.*/*
   (c) Else supposing key(T) has additional non-fk columns $A[1], \ldots A[m]$, ($m \geq 0$); let Ns=$\{Anc_i\} \cup \{$onc$(A[j]), j = 1, .., m\}$, and find skeleton tree Ss' connecting the nodes in Ns, where any pair of nodes in Ns is connected by a many-many path; return (combine(Ss', $\{Ss_j\}$), Ns). /**dealing with the many-to-many binary relationships; also the default action for unaccounted-for tables, e.g., cannot find an identifying relation from a weak entity to the supposed owner entity. No unique anchor exists.*/*

In order for getSkeleton to terminate, it is necessary that there be no cycles in fk references in the schema. Such cycles (which may have been added to represent additional integrity constraints, such as the the fact that an association is total) can be eliminated from a schema by replacing the tables involved with their outer join over the key. getSkeleton deals with strong entities and their functional relationships in step (1), with weak entities in step (2.b.i), and so far, with functional relationships of weak entities in (2.a). In addition to being a catch-all, step (2.c) deals with tables representing many-many relationships (which in this section have key $K = F_1F_2$), by finding anchors for the ends of the relationship, and then connecting them with paths that are not functional, even when every edge is reversed.

To get the logic formula from a tree based on correspondence L, we provide the procedure encodeTree(S, L) below, which basically assigns variables to nodes, and connects them using edge labels as predicates.

Function encodeTree(S,L)
input: subtree S of ontology graph, correspondences L from table columns to attributes of concept nodes in S.
output: variable name generated for root of S, and conjunctive formula for the tree.
steps: Suppose N is the root of S. Let $\Psi = \{\}$.

[8] Both here and elsewhere, when a concept C is added to a tree, so are edges and nodes for C's attributes that appear in L.

1. if N is an attribute node with label f, find d such that $L(\_, d, \_, f, N) = true$, return$(d, true)$. */*for leaves of the tree, which are attribute nodes, return the corresponding column name as the variable and an empty formula.*/*
2. if N is a concept node with label C, then introduce new variable x; add conjunct $C(x)$ to Ψ;
   for each edge p_i from N to N_i */*recursively get the entire formula.*/*
      let S_i be the subtree rooted at N_i,
      let $(v_i, \phi_i(Z_i))$=encodeTree(S_i, L),
      add conjuncts $p_i(x, v_i) \wedge \phi_i(Z_i)$ to Ψ;
return (x, Ψ).

To specify the properties of the algorithm, we now suppose that the correspondences L be the identity mappings from attribute names to table columns. The interesting property of getSkeleton is that if $T = \tau(C)$ according to the er2rel rules in Table 1, where C corresponds to a (strong or weak) entity, then getSkeleton returns (S, Anc), where $Anc = C$ as anchor, and encodeTree(S, L) is logically equivalent to identify$_C$. Similar property exists for $T = \tau(p)$, where p is a functional relationship originating from concept C, in which case its key looks just like an entity key. We now state the desirable properties more formally. Since the precise statement of theorems (and algorithms) is quite lengthy and requires a lot of minute details for which we do not have room here, we express the results as "approximately phrased" propositions. First, getTree finds the desired semantic mapping, in the sense that

Proposition 1. *Let table T be part of a relational schema obtained by er2rel derivation from conceptual model $\mathcal{E}$. Then some tree S returned by* getTree(T, L) *has the property that the formula returned by* encodeTree(S, L) *is logically equivalent to the semantics assigned to T by the er2rel design.*

Note that this "completeness" result is non-trivial, since, as explained earlier, it would not be satisfied by the current Clio algorithm [11], if applied blindly to $\mathcal{E}$ viewed as a relational schema with unary and binary tables. Since getTree may return multiple answers, the following converse "soundness" result is significant

Proposition 2. *If S' is any tree returned by* getTree(T, L)*, with T as above, then the formula returned by* encodeTree(S', L) *represents the semantics of* some *table T' derivable by er2rel design from $\mathcal{E}$, where T' is isomorphic*[9] *to T.*

Such a result would not hold of an algorithm which returns only minimal spanning trees, for example.

We would like to point out that the above algorithm performs reasonably on some non-standard designs as well. For example, consider the relational table $T(\underline{personName}, cityName, countryName)$, where the columns correspond to, respectively, attributes $pname$, $cname$, and $crname$ of concepts $Person, City$ and $Country$ in a CM. If the CM contains a path such that `Person` `-- bornIn ->-` `City` `-- locatedIn ->-` `Country`, then the above table, which is not in 3NF and was not obtained using

[9] Informally, two tables are isomorphic if there is a bijection between their columns which preserves key and foreign key structure.

er2rel design (which would have required a table for *City*), would still get the proper semantics:

T(personName,cityName,countryName) :-
Person(x_1), City(x_2),Country(x_3), bornIn(x_1,x_2), locatedIn(x_2,x_3),
pname(x_1,personName), cname(x_2,cityName),crname(x_3,countryName).

If on the other hand, there was a shorter functional path from $Person$ to $Country$, say an edge labeled `citizenOf`, then the mapping suggested would have been:

T(personName, cityName, countryName) :-
Person(x_1), City(x_2), Country(x_3), bornIn (x_1,x_2),citizenOf(x_1,x_3), ...

which corresponds to the er2rel design. Moreover, had `citizenOf` not been functional, then once again the semantics produced by the algorithm would correspond to the non-3NF interpretation, which is reasonable since the table, having only $personName$ as key, could not store multiple country names for a person.

5.2 Reified Relationships

It is desirable to also have n-ary relationship sets connecting entities, and to allow relationship sets to have attributes (called "association classes" in UML). Unfortunately, these features are not directly supported in most CMLs, such as OWL, which only have binary relationships. Such notions must instead be represented by *"reified relationships"* [2] (we use an annotation * to indicate the reified relationships in a diagram): concepts whose instances represent tuples, connected by so-called "roles" to the tuple elements. So, if *Buys* relates *Person*, *Shop* and *Product*, through roles *buyer*, *source* and *object*, then these are explicitly represented as (functional) binary associations, as in Figure 2. And a relationship attribute, such as when the buying occurred, becomes an attribute of the *Buys* concept, such as *whenBought*.

Unfortunately, reified relationships cannot be distinguished reliably from ordinary entities in normal CMLs on purely formal, syntactic grounds, yet they need to be treated in special ways during recovery. For this reason we assume that they can be distinguished on *ontological grounds*. For example, in Dolce [4], they are subclasses of top-level concepts *Quality* and *Perdurant/Event*. For a reified relationship R, we use functions roles(R) and attribs(R) to retrieve the appropriate (binary) properties.

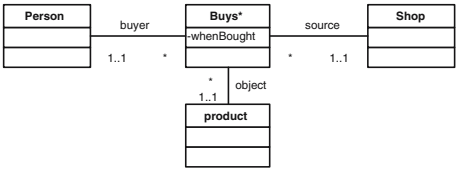


Fig. 2. N-ary Relationship Reified

The design τ of relational tables for reified relationships is shown in Table 2. To discover the correct anchor for reified relationships and get the proper tree, we need to modify getSkeleton, by adding the following case between steps 2(b) and 2(c):

– If key(T)=$F_1F_2 \ldots F_n$ and there exist reified relationship R with n roles $r_1, \ldots, r_n$ pointing at the singleton nodes in $Anc_1, \ldots, Anc_n$ respectively,
then let $S =$ combine($\{r_j\}, \{Ss_j\}$), and return $(S, \{R\})$.

Table 2. er2rel Design for Reified Relationship

ER model object O	Relational Table $\tau(O)$	
Reified Relationship R	*columns:*	$ZX_1 \ldots X_n$
if $r_1, \ldots, r_n$ are roles of R	*primary key:*	$X_1 \ldots X_n$
let Z=attribs(R)	*fk's:*	$X_1, \ldots, X_n$
X_i=key($\tau(E_i)$)	*anchor:*	R
where E_i fills role r_i	*semantics:*	$T(ZX_1 \ldots X_n)$:- $R(y), E_i(w_i)$, hasAttribs(y, Z), $r_i(y, w_i)$, identify$_{E_i}(w_i, X_i), \ldots$
	identifier:	identify$_R(y, \ldots X_i \ldots)$:- $R(y), \ldots E_i(w_i)$, $r_i(y, w_i)$, identify$_{E_i}(w_i, X_i)$,...

The main change to getTree is to compensate for the fact that if getSkeleton finds a *reified* version of a many-many binary relationship, it will no longer look for an unreified one. So after step 1. we add

– if key(T) is the concatenation of two foreign keys F_1F_2, and nonkey(T) is empty, compute (Ss_1,Anc_1) and (Ss_2, Anc_2) as in step 2. of getSkeleton; then find ρ=shortest many-many path connecting Anc_1 to Anc_2;
return $(S') \cup$ (combine(ρ, Ss_1, Ss_2))

The previous version of getTree was set up so that with these modifications, attributes to reified relationships will be found properly, and the previous propositions continue to hold.

5.3 Replication

If we allow recursive relationships, or allow the merger of tables for different functional relationships connecting the same pair of concepts (e.g., `works_for` and `manages`), the mapping in Table 1 is incorrect because column names will be repeated in the multiple occurrences of the foreign keys. We will distinguish these (again, for ease of presentation) by adding superscripts as needed. For example, if *Person* is connected to itself by the *likes* property, then the table for *likes* will have schema $T[\underline{ssn^1}, \underline{ssn^2}]$.

During mapping discovery, such situations are signaled by the presence of multiple columns c and d of table T corresponding to the same attribute f of concept C. In such situations, the algorithm will first make a copy C_{copy} of node C in the ontology graph, as well as its attributes. C_{copy} participates in all the object relations C did, so edges must be added. After replication, we can set onc$(c) = C$ and onc$(d) = C_{copy}$, or onc$(d) = C$ and onc$(c) = C_{copy}$ (recall that onc(c) gets the concept corresponded by column c in the algorithm). This ambiguity is actually required: given a CM with *Person* and *likes* as above, a table $T[\underline{ssn^1}, \underline{ssn^2}]$ could have alternate semantics corresponding to *likes*, and its inverse, $likedBy$. (A different example would involve a table $T[\underline{ssn}, addr^1, addr^2]$, where *Person* is connected by two relationships, *home* and *office*, to concept *Building*, which has an *address* attribute.

The main modification needed to the getSkeleton and getTree algorithms is that no tree should contain both a functional edge D --- p ->-- C and its replicate

D --- p ->-- C_{copy}, (or several replicates), since a function has a single value, and hence the different columns of a tuple will end up having identical values: a clearly poor schema.

5.4 Addressing Class Specialization

The ability to represent subclass hierarchies, such as the one in Figure 3 is a hallmark of CMLs and modern so-called Extended ER (EER) modeling.

Almost all textbooks (e.g., [14]) describe two techniques for designing relational schemas in the presence of class hierarchies

1. Map each concept/class into a separate table following the standard er2rel rules. This approach requires two adjustments: First, subclasses must inherit identifier attributes from a single super-class, in order to be able to generate keys for their tables. Second, in the table created for an immediate subclass C' of class C, its key $\mathsf{key}(\tau(C'))$ should also be set to reference as a foreign key $\tau(C)$, as a way of maintaining inclusion constraints dictated by the is-a relationship.
2. Expand inheritance, so that *all* attributes and relations involving a class C appear on all its subclasses C'. Then generate tables as usual for the subclasses C', though not for C itself. This approach is used only when the subclasses cover the superclass.
   some researchers also suggest a third possibility:
3. "Collapse up" the information about subclasses into the table for the superclass. This can be viewed as the result of $\mathsf{merge}(T_C, T_{C'})$, where $T_C[K, A]$ and $T_{C'}[K, B]$ are the tables generated for C and its subclass C' according to technique (1.) above. In order for this design to be "correct", [8] requires that $T_{C'}$ not be the target of any foreign key references (hence not have any relationships mapped to tables), and that B be non-null (so that instances of C' can be distinguished from those of C).

The use of the key for the root class, together with inheritance and the use of foreign keys to also check inclusion constraints, make many tables highly ambiguous. For example, according to the above, table $T(\underline{ss\#}, crsId)$, with $ss\#$ as the key and a foreign key referencing T', could represent at least

(a) *Faculty teach Course*
(b) *Lecturer teach Course*
(c) *Lecturer coord Course*.

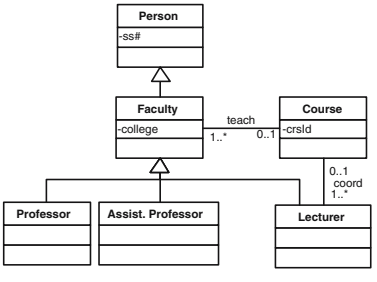


Fig. 3. Specialization Hierarchy

This is made combinatorially worse by the presence of multiple and deep hierarchies (e.g., imagine a parallel *Course* hierarchy), and the fact that not all ontology concepts are realized in the database schema, according to our scenario. For this reason, we have chosen to try to deal with some of the ambiguity relying on users, during the establishment of correspondences. Specifically, the user is supposed to provide a correspondence from column c to attribute f *on the lowest class whose instances provide data appearing in the column*. Therefore, in

the above example of table $T(\underline{ss\#}, crsId)$, *ss#* is made to correspond to *ssn* on *Faculty* in case (a), while in cases (b) and (c) it is made to correspond to *ss#* on *Lecturer*. This decision was also prompted by the CM manipulation tool that we are using, which automatically expands inheritance, so that *ss#* appears on all subclasses.

Under these circumstances, in order to capture designs (1.) and (2.) above, we do not need to modify our earlier algorithm in any way, if we first expand inheritance in the graph. So the graph would show `Lecturer -- teaches;coord ->- Course` in the above example, and *Lecturer* would have all the attributes of *Faculty*.

To handle design (3.), we can add to the graph an actual edge for the inverse of the **is-a** relation: a functional edge labeled *alsoA*, with lower-bound 0: `C --- alsoA ->- C'`, connecting superclass *C* to each of its subclasses *C'*. It is then sufficient to allow functional paths between concepts to consist of *alsoA* edges, in addition to the normal kind, in getTree.

5.5 Outer Joins

The observant reader has probably noticed that the definition of the semantic mapping for $T = \mathsf{merge}(T_E, T_p)$ was not quite correct: $T(\underline{K}, V, W) : -\phi(K, V), \psi(K, W)$ describes a join on K, rather than a left-outer join, which is what is required if p is a non-total relationship. In order to specify the equivalent of outer joins in a perspicuous manner, we will use conjuncts of the form $\lceil \mu(X, Y) \rceil^Y$, which will stand for the formula $\mu(X, Y) \vee (Y = null \wedge \neg \exists Z.\mu(X, Z))$, indicating that null should be used if there are no satisfying values for the variables Y. With this notation, the proper semantics for merge is $T(\underline{K}, V, W) : -\phi(K, V), \lceil \psi(K, W) \rceil^W$.

In order to obtain the correct formulas from trees, encodeTree needs to be modified so that when traversing a non-total edge p_i that is not part of the skeleton, in the second-to-last line of the algorithm we must allow for the possibility of v_i not existing.

Our formal results still hold under the replication, the treatment of specialization hierarchy, and the encoding of the merging of non-total functional relationships into outer joins.

6 Experience

So far, we have developed the mapping inference algorithm by investigating the connections between the semantic constraints in both relational models and ontologies. The theoretical results show that our algorithm will report the "right" semantics for schemas designed following the widely accepted design methodology. Nonetheless, it is crucial to test the algorithm in real-world schemas and ontologies to see its overall performance. To do this, we have implemented the mapping inference algorithm in our prototype system MAPONTO, and have applied it on a set of schemas and ontologies. In this section, we provide some evidence for the effectiveness and usefulness of the prototype tool by discussing the set of experiments and our experience.

Our test data were obtained from various sources, and we have ensured that the databases and ontologies were developed independently. The test data are listed in Table 3. They include the following databases: the Department of Computer Science

Table 3. Characteristics of Schemas and ontologies for the Experiments

Database Schema	Number of Tables	Number of Columns	Ontology	Number of Nodes	Number of Links
UTCS Department	8	32	Academic Department	62	1913
VLDB Conference	9	38	Academic Conference	27	143
DBLP Bibliography	5	27	Bibliographic Data	75	1178
OBSERVER Project	8	115	Bibliographic Data	75	1178
Country	6	18	CIA factbook	52	125

database in University of Toronto; the VLDB conference database; the DBLP computer science bibliography database; the COUNTRY database appearing in one of reverse engineering papers; and the test schemas in OBSERVER [9] project. For the ontologies, our test data include: the academic department ontology in the DAML library; the academic conference ontology from the SchemaWeb ontology repository; the bibliography ontology in the library of the Stanford's Ontolingua server; and the CIA factbook ontology. Ontologies are described in OWL. For each ontology, the number of links indicates the number of edges in the multi-graph resulted from object properties. We have made all these schemas and ontologies available on our web page: www.cs.toronto.edu/~yuana/research /maponto/relational/testData.html.

To evaluate our tool, we sought to understand whether the tool could produce the intended mapping formula if the simple correspondences were given. We were concerned about the number of formulas presented by the tool for users to sift through. Further, we wanted to know whether the tool was still useful if the correct formula was not generated. In this case, we expected that a user could easily debug a generated formula to reach the correct one instead of creating it from scratch. A summary of the experimental results are listed in Table 4 which shows the average size of each relational table schema in each database, the average number of candidates generated, and the average time for generating the candidates. Notice that the number of candidates is the number of semantic trees obtained by the algorithm. Also, a single edge of an semantic tree may represent the multiple edges between two nodes, collapsed using our $p; q$ abbreviation. If there are m edges in a semantic tree and each edge has n_i $i = 1, .., m$ original edges collapsed, then there are $\prod_i^m n_i$ original semantic trees. We show below a formula generated from such a collapsed semantic tree:

TaAssignment(courseName, studentName) :-
Course(x_1), GraduateStudent(x_2), **[hasTAs;takenBy]**(x_1,x_2),
workTitle(x_1,courseName), personName(x_2,studentName).

where, in the semantic tree, the node *Course* and the node *GraduateStudent* are connected by a single edge with label **hasTAs;takenBy** which represents two separate edges, *hasTAs* and *takenBy*.

Table 4 shows that the tool only present a few mapping formulas for users to examine. This is due in part to our compact representation of parallel edges between two nodes shown above. To measure the overall performance, we manually created the mapping formulas for all the 28 tables and compared them to the formulas generated by the tool. We observed that the tool produced correct formulas for 24 tables. It demonstrated

Table 4. Performance Summary for Generating Mappings from Relational Tables to Ontologies

Database Schema	Avg. Number of Cols/per table	Avg. Number of Candidates generated	Avg. Execution time(ms)
UTCS Department	4	4	279
VLDB Conference	5	1	54
DBLP Bibliography	6	3	113
OBSERVER Project	15	2	183
Country	3	1	36

that the tool is able to understand the semantics of many practical relational tables in terms of an independently developed ontology.

However, we wanted to know the usefulness of the tool. To evaluate this, we examined the generated formulas which were not the intended ones. For each such formula, we compared it to the manually created and correct one, and we used a very coarse measurement to record how much effort we had to spend to debug the generated formula in order to make it correct. Such a measurement only recorded the changes of predicate names in a formula. For example, the tool generated the following formula for the table *Student(name, office, position, email, phone, supervisor)*:

$$\textit{Student}(X_1),\ \textit{emailAddress}(X_1,\textit{email}),\ \textit{personName}(X_1,\textit{name}),\ \textit{Professor}(X_2),\ \textit{Institute}(X_3),\ \textit{head}(X_3,X_2),\ \textit{affiliatedOf}(X_3,X_1),\ \textit{personName}(X_2,\ \textit{supervisor})\ldots \quad (1)$$

If the intended semantics for the above table columns is:

$$\textit{Student}(X_1),\ \textit{emailAddress}(X_1,\textit{email}),\ \textit{personName}(X_1,\textit{name}),\ \textit{Professor}(X_2),\ \textit{GraduateStudent}(X_3),\ \textit{hasAdvisor}(X_3,X_2),\ \textit{isA}(X_3,X_1),\ \textit{personName}(X_2,\ \textit{supervisor})\ldots \quad (2)$$

then, one can change the three predicates $\textit{Institute}(X_3),\ \textit{head}(X_3,X_2),\ \textit{affiliatedOf}(X_3,X_1)$ in formula (1) to $\textit{GraduateStudent}(X_3),\ \textit{hasAdvisor}(X_3,X_2),\ \textit{isA}(X_3,X_1)$ instead of writing the entire formula (2) from scratch. Our experience working with the tool has shown that significant effort have been saved when building semantic mappings from tables to ontologies, because in most cases one only needed to change a relatively small number of predicates in an existing formula.

Tables 4 indicate that execution times were not significant, since, as predicted, the search for subtrees and paths took place in a relatively small neighborhood.

7 Conclusion and Future Work

Semantic mappings between relational database schemas and ontologies in the form of logic formulas play a critical role in realizing the semantic web as well as in many data sharing problems. We have proposed a solution to infer the LAV mapping formulas from simple correspondences, relying on information from the database schema (key and foreign key structure) and the ontology (cardinality restrictions, **is-a** hierarchies). Theoretically, our algorithm infers all and only the semantics implied by the ER-to-relational design if a table's schema follows ER design principles. In practice, our ex-

perience working with independently developed schemas and ontologies has shown that significant effort has been saved in specifying the LAV mapping formulas.

We are working towards disambiguation between multiple possible semantics by exploiting the following sources of information: first, a richer modeling language, supporting at least disjointness/coverage in **is-a** hierarchies, but also more complex axioms as in OWL ontologies; second, the use of the *data* stored in the relational tables whose semantics we are investigating. For example, queries may be used to check whether complex integrity constraints implied by the semantics of a concept/relationship fail to hold, thereby eliminating some candidate semantics.

Acknowledgments. We are most grateful to Renée Miller and Yannis Velegrakis for their clarifications concerning Clio, comments on our results, and encouragement. Remaining errors are, of course, our own.

References

1. D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. *J. of Coop. Info. Sys.*, 10(3):237–271, 2001.
2. M. Dahchour and A. Pirotte. The Semantics of Reifying n-ary Relationships as Classes. In *ICEIS'02*, pages 580–586, 2002.
3. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD'04*, pages 383–394, 2004.
4. A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening Ontologies with DOLCE. In *EKAW'02*, pages 166–181, 2002.
5. J.-L. Hainaut. *Database Reverse Engineering*. http:// citeseer.ist.psu.edu/ article/ hainaut98database.html, 1998.
6. S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *Proc. WWW'03*, 2003.
7. A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *J. of Intelligent Info. Sys.*, 5(2):121–143, Dec 1996.
8. V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE TSE*, 16(8):777–790, August 1990.
9. E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Preexisting Ontologies. In *CoopIS'96*, pages 14–25, 1996.
10. R. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB'00*, pages 77–88, 2000.
11. L. Popa, Y. Velegrakis, R. J. Miller, M. Hernandes, and R. Fagin. Translating Web Data. In *VLDB'02*, pages 598–609, 2002.
12. M. R. Quillian. Semantic Memory. In *Semantic Information Processing*. Marvin Minsky (editor). 227-270. The MIT Press. 1968.
13. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10:334–350, 2001.
14. R. Ramakrishnan and M. Gehrke. *Database Management Systems (3rd ed.)*. McGraw Hill, 2002.
15. H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *IJCAI'01 Workshop. on Ontologies and Information Sharing*, 2001.

Ontology Transformation and Reasoning for Model-Driven Architecture

Claus Pahl

Dublin City University, School of Computing, Dublin 9, Ireland
cpahl@computing.dcu.ie

Abstract. Model-driven Architecture (MDA) is a software architecture framework proposed by the Object Management Group OMG. MDA emphasises the importance of modelling in the architectural design of service-based software systems. Ontologies can enhance the modelling aspects here. We present ontology-based transformation and reasoning techniques for a layered, MDA-based modelling approach. Different ontological frameworks shall support domain modelling, architectural modelling and interoperability. Ontologies are beneficial due to their potential to formally define and automate transformations and to allow reasoning about models at all stages. Ontologies are suitable in particular for the Web Services platform due to their ubiquity within the Semantic Web and their application to support semantic Web services.

1 Introduction

The recognition of the importance of modelling in the context of software architecture has over the past years led to model-driven architecture (MDA) – a software engineering approach promoted by the Object Management Group (OMG) [1]. MDA combines service-oriented architecture (SOA) with modelling techniques based on notations such as the Unified Modelling Language (UML). Recently, ontologies and ontology-based modelling have been investigated as modelling frameworks that enhance the classical UML-based approaches. While formal modelling and reasoning is, to some extent, available in the UML context in form of OCL, ontologies offer typically full reasoning support, for instance based on description logic. A second benefit of ontologies is the potential to easily reuse and share models.

Modelling and developing software systems as service-based architectures is gaining increasing momentum. Modelling and describing services is central for both providers and clients of services. Providers need to provide an accurate description or model for a service that can be inspected by potential clients. In particular the attention that the Web Services Framework (WSF) [2, 3] has received recently emphasises the importance of service-orientation as the architectural paradigm. Service-oriented architecture (SOA) [4, 5] is becoming a central software engineering discipline that aims at supporting the development of distributed systems based on reusable services.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1170–1187, 2005.

MDA has been developed within the context supported and standardised by the OMG, i.e. UML as the modelling notation and CORBA as the implementation platform. It is based on a layered architecture of models at different abstraction levels. Our focus is the Web Services platform based on techniques supported by the World-Wide Web Committee (W3C) [2], such as WSDL and SOAP, but also extensions like WS-BPEL for service processes [6, 7, 8]. This specific area is particularly suitable to demonstrate the benefits of ontology-based modelling due the distributed nature of service-based software development with its emphasis on provision and discovery of descriptions and on sharing and reuse of models and services.

While the OMG has started a process towards the development of an Ontology Definition Metamodel (ODM) [9] that can support ontological modelling for particular MDA model layers, we take a more comprehensive approach here with ontologies for all layers. Our approach will also be more concrete, i.e. based on concrete ontologies. Other authors, e.g. [10], have already explored OWL – the Web Ontology Language – for MDA frameworks. We will extend these approaches by presenting here a layered, ontology transformation-based modelling approach for software services. We will introduce ontology-based modelling approaches for three MDA layers – computation-independent, platform-independent, and platform-specific. Ontologies will turn out to support a number of modelling tasks – from domain modelling to architectural configuration and also service and process interoperability. We will put an emphasis on processes, which will play an important role in modelling domain activities but also in modelling interaction processes in software architecture configurations.

Our contribution is a layered ontological transformation framework with different ontologies focussing on the needs of modelling at particular abstraction layers. We will indicate how such a framework can be formally defined – a full formalisation is beyond the scope of this paper. We will present our approach here in terms of an abstract notation, avoiding the verbosity of XML-based representations for services and ontologies.

We will start with an overview of service-oriented architecture and ontology-based modelling in Section 2. Service modelling with ontologies is introduced in Section 3. We will address the transformations between ontological layers in Section 4. We discuss our efforts in the context of OMG adoption and standardisation in Section 5. Related Work is discussed in Section 6. We end with some conclusions in Section 7.

2 Service Architectures and Models

The context of our work is model-driven architecture (MDA). Our platform is service-based; our modelling approach is ontology-based. These aspects shall now be introduced.

2.1 Services and the Web Services Framework

A service provides a coherent set of operations at a certain location [3]. The service provider makes an abstract service interface description available that can

be used by potential service users to locate and invoke the service. Services have so far usually been used 'as is' in single request-response interactions [11]. However, the configuration and coordination of services in service-based architectures and the composition of services to processes is becoming equally important in the second generation of service technology. Existing services can be reused and composed to form business or workflow processes. The principle of architectural composition is process assembly.

The discovery and invocation infrastructure of the Web Services Framework (WSF) [2] – a registry or marketplace where potential users can search for suitable services and an invocation protocol – with the services and their clients form our platform, i.e. a service-oriented architecture. Languages for description are central elements of a service-oriented architecture. With the second generation of service technology and efforts such as MDA, the emphasis has shifted from description to the wider and more comprehensive activity of modelling.

Behaviour and interaction processes are essential parts of modelling and understanding software system architectures [12, 13, 14, 15]. However, the support that architectural description languages offer with regard to behavioural processes in architectures is sometimes limited. MDA focuses on UML modelling to support architectural designs. In [16], scenarios, i.e. descriptions of interactions of a user with a system, are used to operationalise requirements and map these to a system architecture.

2.2 Ontologies and the Semantic Web

Making the Web more meaningful and open to manipulation by software applications is the objective of the Semantic Web initiative. Knowledge representation and logical inference techniques form its backbone [17, 18]. Ontologies – the key to a semantic Web – express terminologies and precisely defined semantical properties and a create shared understanding of annotations of Web resources such as Web pages or services. Ontologies usually consist of hierarchical definitions of important concepts in a domain and descriptions of the properties of each concept, supported by logics for knowledge representation and reasoning.

Ontologies are, however, important beyond sharable and processable annotations of Web resources. Some effort has already been made to exploit Semantic Web and ontology technology for the software engineering domain in general and modelling in particular [19]. OWL-S [20] for instance is a service ontology, i.e. it is a language that provides a specific vocabulary for describing properties and capabilities of Web services, which shows the potential of this technology for software engineering.

An ontology is defined in terms of concepts and relationships. An ontology is a model of a domain made available through the vocabulary of concepts and relationships. Concepts capture the entities of the domain under consideration. Instances are concrete objects of a particular concept. Relationships capture the relationships between concepts and their properties. In this respect, ontologies are similar to modelling notations such as UML. Ontologies, however, combine modelling with logic-based reasoning. Properties of concepts are specified

in terms of (universal or existential) quantifications over relationships to other concepts.

Formality in the Semantic Web framework facilitates machine understanding and automated reasoning. OWL (the Web Ontology Language), in particular OWL-DL, is equivalent to a very expressive description logic [21]. This fruitful connection provides well-defined semantics and reasoning systems. Description logic is particularly interesting for the software engineering context due to a correspondence between description logic and dynamic logic (a modal logic of programs).

2.3 Model-Driven Architecture

Model-driven architecture (MDA) is a software architecture approach emphasising the importance of modelling for the architectural design of software systems [1]. The platform targeted by MDA are service-based architectures. MDA suggests a three-layered approach:

- The Computation Independent Model (CIM) describes a system from the computation-independent viewpoint, addressing structural aspects of the system. A CIM is often called a domain model.
- The Platform Independent Model (PIM) can be seen as defining a system in terms of a technology-neutral virtual machine or a computational abstraction.
- The Platform Specific Model (PSM) usually consists of a platform model that captures the technical concepts and services that make up the platform and an implementation-specific model geared towards the concrete implementation technique.

The archetypical OMG MDA is based on UML for platform independent modelling and CORBA as the platform with its languages such as IDL as the platform-specific notation. In our context, the platform is a service-based infrastructure. Different platform types can be distinguished. The generic platform is SOA here, the technology-specific platform is the WSF, and vendor-specific platform technologies are for instance the Apache Axis or Collaxa service engines.

3 Modelling with Ontologies

MDA proposes three modelling layers – each with a distinct focus that, as we aim to demonstrate, can be supported ontologically.

- The computation-independent layer focuses on domain capture.
- The platform-independent layer focuses on architecture configuration and service process composition.
- The platform-specific layer focuses on interoperability and discovery support.

A case study from the banking domain will accompany our investigations.

3.1 CIM – Computation Independent Model

The purpose of the Computation Independent Model (CIM) is to capture a domain with its concepts and properties. Typically, two viewpoints of domain modelling can be distinguished. Concepts are represented in form of hierarchies – called the information viewpoint in MDA. Behaviour is represented in a process-based form – called the enterprise viewpoint in MDA, based on open distributed processing (ODP) concepts. Our aim is to provide a single ontological notation that can capture both viewpoints. A process-oriented ontology shall capture both types of domain entities, see Fig. 1.

– Two types of concepts shall be distinguished: objects, which are static entities, and processes, which are dynamic entities.

Domain ontology (OWL-style) – CIM layer

model	::= relationship* | constraint*
concept	::= object | process
relationship_type	::= is_a | has_part | depends
relationship	::= concept relationship_type concept
constraint	::= conceptConstraint | relationshipConstraint

Service process ontology (WSPO) – PIM layer

model	::= pre process post
process	::= preState procExpr postState
pre	::= preState preCond condition | preState inObj syntax
post	::= postState postCond condition | postState outObj syntax
procExpr	::= process | ! procExpr | procExpr ; procExpr | procExpr + procExpr | procExpr || procExpr

---

Service ontology (WSMO) – PSM layer

model	::= service
service	::= interface capabilities
interface	::= messageExchange nonFunctProp
capabilities	::= preCond postCond assumption effect nonFunctProp

---

Fig. 1. Abstract Syntax of Ontologies

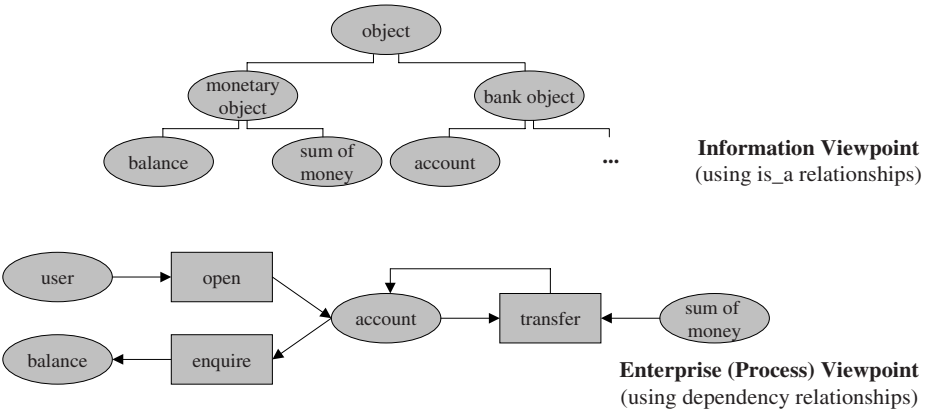


Fig. 2. CIM-level Excerpts from a Banking Domain Ontology

- Three relationship types shall be distinguished: is_a (the subclass relationship), has_part (the component relationship), and depends (the dependency relationship).
- Constraints, or properties, on concepts and relationships can be expressed.

The subclass relationship is the classical form of relating concepts in ontologies. For domain-specific software systems, the composition of objects and processes from a component perspective is an additional, but also essential information. Dependencies are useful to describe input-output relationships between objects and activities that process them. Specific ordering requirements on composed processes can be expressed through constraints. The abstract syntax of this ontology language is summarised in Fig. 1, upper part. We will discuss the semantics at the end of this section.

We need to define or identify an ontology language that can provide the necessary notational framework. An OWL-based ontology with support for the component and dependency relationships can form the notational framework here.

Example 1. The example that we will use to illustrate the modelling and transformation techniques throughout the paper is taken from the banking domain. We can identify:

- objects such as *account* and *sum* (of money),
- activities such as *account create, close, lodge, transfer,* and *balance* and processes such as for instance *create; !(balance + lodge + transfer); close* which describes sequencing, iteration, and choice of activities[1],
- constraints such as a precondition *balance* $\geq$ *sum* on the transfer activity.

[1] The process combinators are ';' (sequential composition), '!' (iteration), '+' (choice), and '||' (parallel composition).

The example in Fig. 2 shows a simplified domain ontology for the bank account example.

Reasoning facilities of an ontological framework can be deployed to check the consistency of ontologically defined domain models.

Example 2. With instances attached to the entities, an inference engine can, for example, determine all bank account instances with a negative account balance. Another example is the satisfaction of a precondition for a money transfer on a particular account.

3.2 PIM – Platform Independent Model

The Platform Independent Model (PIM) changes the focus from the computation-independent capture of the domain to a focus on constraints imposed on the knowledge representation by the computational environment. Architectures and processes are here the key aspects at this modelling level. The architectural focus is on services, their architectural configuration, and interaction processes [13, 22]. Architectural configuration addresses the interaction processes (remote invocation and service activation) between different agents in a software system. Again, we will use an ontology to express these aspects.

Services are the components of the system architecture. They form the starting point of architecture modelling. Different approaches for service ontologies have been proposed. These differ in the way service and processes are represented in the ontologies – see Section 6 for a more detailed review. Since representing not only services, but also their configuration and assembly into processes is important here, we use the Web Service Process Ontology (WSPO), whose foundations were developed in [23, 24]. This ontology will bring us closer to the architectural perspective than more abstract service ontologies such as OWL-S [20], which however also provides support for service composition. Services (and processes) in WSPO are not represented as concepts, but as relationships denoting accessibility relations between states of the system. A PIM service process template, see Fig. 3, defines the basic structure of states and service processes.

The abstract syntax of this ontology is presented in Fig. 1, middle part.

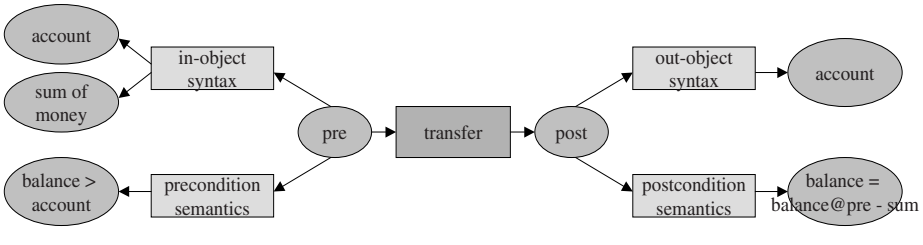


Fig. 3. Ontological Service Process Template (WSPO)

- Concepts in this approach are states (pre- and poststates), parameters (in- and outparameters), and conditions (pre- and postconditions).
- Two forms of relationships are provided. The processes themselves are called transitional relationships. Syntactical and semantical descriptions – here parameter objects (syntax) and conditions (semantics) – are associated through descriptional relationships.

This ontological representation in WSPO is actually an encoding of a simple dynamic logic (a logic of programs) in a description logic format [23, 24], allowing us to avail of modal logic reasoning in this framework.

WSPO provides a standard template for service or service process description. Syntactical parameter information in relation to the individual activities – to be implemented through service operations – and also semantical information such as pre-conditions like are attached to each activity as defined in the PIM template. Example 4 will illustrate this. WSPO can be distinguished from other service ontologies by two specific properties. Firstly, although based on description logics, it adds a relationship-based process sublanguage enabling process expressions based on iteration, sequential and parallel composition, and choice operators. Secondly, it adds data to processes in form of parameters that are introduced as constant process elements into the process sublanguage.

Example 3. The architecture- and process-oriented PIM model of the bank account focuses on the activities and how they are combined to processes. The process $create; !(balance + lodge + transfer); close$, which describes a sequence of account creation, an iteration of a choice of balance enquiry, lodgment, and transfer activities, and a final account closing activity, can be represented in WSPO as a composed relationship expression:

$$\begin{array}{l} create \circ acc; \\ !\ (\ balance \circ acc;\ lodge \circ (acc, sum);\ transfer \circ (from, to, sum)\ );\\ close \circ acc \end{array}$$

Ontologies enable reasoning about specifications. WSPO enables reasoning about the composition of services in architectures. In [23], we have presented an ontological matching notion that can be applied to determine whether a service provider can be connected to a service user based on their individual service and process requirements.

Example 4. Assume that in order to implement an account process, a transfer service needs to be integrated. For any given state, the process developer might require[2]

$$\begin{array}{l} \forall preCond\ .\ (balance > amount) \\ \forall transfer\ .\ \forall postCond\ . \\ \qquad\qquad (balance() = balance()@pre - amount) \end{array}$$

[2] The @-construct refers to the attribute in the prestate.

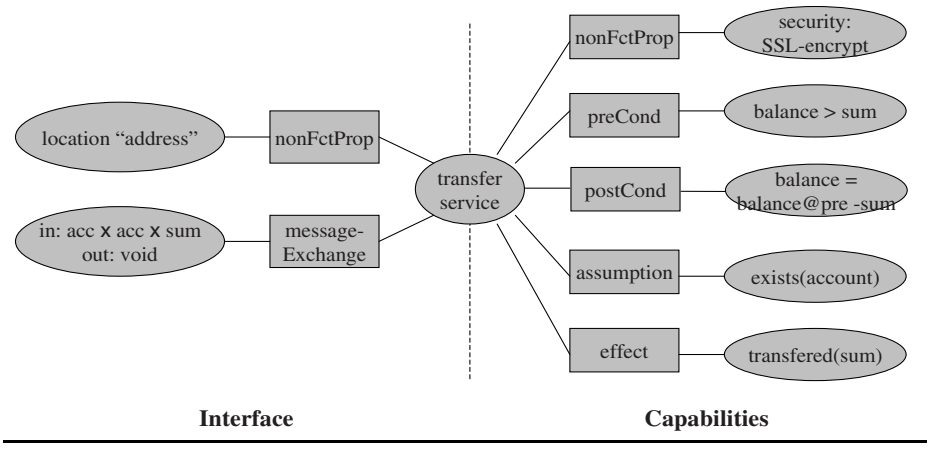


Fig. 4. Ontological Service Template (WSMO)

which would be satisfied by a provided service

$$\forall preCond \,.\, true$$
$$\forall transfer \,.\, \forall postCond \,.$$
$$(balance() = balance()@pre - amount) \wedge$$
$$(lastActivity = \text{'transfer'})$$

based on a refinement condition (weakening the precondition and strengthening the postcondition).

The refinement notion used in the example above is based on the consequence inference rule from dynamic logic integrated into WSPO.

While architecture is the focus of this model layer, the approach we discussed does not qualify as an architecture description language (ADL) [26], although the aim is also the separation of computation (within services) and communication (interaction processes between services). ADLs usually provide notational means to describe components (here services), connectors (channels between services), and configurations (the assembly of instantiations of components and connectors). Our approach comes close to this aim by allowing services as components and process expressions as configurations to be represented.

3.3 PSM – Platform Specific Model

Our platform is the Web Service Framework (WSF) – consisting of languages, protocols, and software tools. Models for the platform specific layer (PSM) need to address two aspects: a platform model and implementation specific models. The platform model is here defined by the Web Services Framework and its service-oriented architecture principles. The implementation specific models characterise the underlying models of the predominant languages of the platform.

Interoperability of services is the key objective of the WSF. Two concerns determine the techniques used at this layer: the abstract description of services to support their discovery and the standardised assembly of services to processes. Two different models capturing executable and tool-supported languages are therefore relevant here:

- Description and Discovery. Abstract syntactical and semantical service interfaces shall be supported. The Web Services Description Language (WSDL) supports syntactical information. WSDL specifications can be created by converting syntactical information from the WSPO into abstract WSDL elements. We will, however, focus here on semantically enhanced descriptions enabled through service ontologies specific to the platform.
  Services as the basic components of processes can be represented as concepts in ontologies [27]. This approach is followed by widely used service ontologies such as OWL-S [20] and WSMO [25]. WSMO defines a template for the representation of service-related knowledge, see Figs. 1 and 4. The WSMO concepts are the central services concept and auxiliary domains for descriptional entities, i.e. expressions of different kinds. Relationships in the template represent service properties of two kinds. Properties such as *preCond*, *postCond*, *assumption*, and *effects* are called capabilities relating to the service semantics. Properties such as *messageExchange* are syntactically oriented interface aspects. In addition to these functional aspects, a range of non-functional aspects is supported.
- Processes and Composition. The Business Process Execution Language for Web Services (WS-BPEL) is one of the proposed service coordination languages [6]. WS-BPEL specifications can be created by converting process expressions from WSPO. We do not discuss this further here as the semantical support required here is already available at the PIM-level.

The benefit of using an ontology for description and discovery can easily be seen when the discovery and matching of OWL-S or WSMO-based service descriptions is compared with syntax-oriented WSDL descriptions.

Example 5. WSMO descriptions capture syntactical and semantical descriptions as WSPO does, see Examples 3 and 4. It adds, however, various non-functional aspects that can be included into the discovery and matching task. Standardised description and invocation formats enable interoperability. Required functionality can be retrieved from other locations. An example are authentication features for an online banking system.

3.4 Semantics of the Ontology Layers

The abstract syntax of the ontologies we discussed has already been presented in Fig. 1. Now, we focus on the semantics of the individual ontology layers. Ontologies are based on ontology languages, which in turn are defined in terms of logics. Here, we can exploit the description logic foundation of ontology languages such as OWL [21]. While a full treatment is beyond the scope of this paper,

we address the central ideas due to the definition of ontology transformations requires underlying formal semantical models.

Ontology languages are logics defined by interpretations and satisfaction relations on semantical structures such as algebras (sets and relations) and state-based labelled transitions systems (e.g. Kripke transition systems). A semantical metamodel for each of the layers can be formulated based on standard approaches in this context [21]:

- A domain ontology can be defined in terms of sets (for concepts) and relations (for relationships).
- The architectural and process aspects can be defined in terms of labelled transition systems, such as Kripke transition systems (KTS), where sets represent states and relations represent transitions between states.
- The interoperability aspects can be split into interface (defined in terms of sets and relations) and configuration and process behaviour (defined in terms of state transition mechanisms).

This shall form the basis for the approach presented here. In the future, we plan to map our semantics to the OMG-supported Ontology Definition Metamodel (ODM). This can be expected to be straightforward due to an ODM-OWL mapping as part of ODM. We will address this aspect in Section 5.

4 Ontology-Based Model Transformations

Without explicitly defined transformations, a layered modelling approach will not be feasible. The transformations play consequently a central role. Transformations between the model layers need to be automated to provide required tool support and to enable the success of the approach. Following the OMG-style of defining transformation, we define transformation rules based on patterns and templates.

4.1 Transformation Principles

While it is evident that the transformations we require here are about adding new structures, for instance notions of state and state transition for the architectural PIM layer, the original model should be recoverable and additional information on that layer should not be added. What we aim at is therefore not a refinement or simulation notion in the classical sense – although these notions will help us to define the transformations.

Refinements where additional application-specific model specifications are added can occur within a given layer. Refinement within a model layer can be based on subsumption, the central reasoning construct of ontology languages that is based on the subclass relation between concepts or relationship classes, respectively. In [23], we have developed a constructive refinement and simulation framework for service processes based on refinement and simulation relations as special forms of subsumption.

Rule	Aspect	Description
CP0	template	For each process element in the CIM, create a PIM template.
CP1	process element	The PIM process element is the process element of CIM.
CP2	states	Create default concepts for pre- and post-states.
CP3	syntax	For each in- and out-parameter of processes, create a separate syntax (object) element.
CP4	semantics	Create pre- and postconditions depending on availability of external additional information in form of constraints.
CP5	process expressions	If process expressions available in form of constraints, then create complex process using relationship expressions in WSPO.

Fig. 5. Transformation Rules for the CIM-to-PIM Mapping

Our focus in this paper is the illustration of the different modelling capabilities of ontology languages and ontologies on the different model layers. Our objective is to motivate the need for and the benefits of a layered ontological modelling and transformation approach. A formal model of transformations is beyond the scope of this paper. Graph transformation and graph grammars provide suitable frameworks for this task [28, 29].

4.2 CIM-to-PIM Mapping

The CIM-layer supports abstract, computation-independent domain modelling. This model is mapped to a computation-oriented, but still platform-independent service-based model. The PIM-layer supports analysis and reasoning for architecture and process aspects, such as configuration and composition, on an abstract level. Consequently, information needs to be added to a CIM to provide a sufficient level of structure for the PIM-level. A process-specific PIM template, see Fig. 3 for a template application to the banking context, guides the transformation process. We have defined the rules for the CIM-to-PIM transformation in Fig. 5.

In MDA, the transformation steps are defined in terms of model markings and applications of templates. Marks are annotations (or metadata) of entities in the original model to support the mapping that indicates how these entities are used in the target model. Marks can support the determination of the mapping template to be deployed. The CIM-to-PIM transformation rule, that defines the creation of a PIM-template for CIM-concepts marked as 'process', is an example of this.

Example 6. Fig. 3 represents the result of the transformation of the 'transfer' process from Fig. 2 using the rules defined in Fig. 5. The 'transfer' concept in Fig. 2 is marked as a process, which based on rule CP0 creates a PIM process template with explicit states (rule CP2). The CIM concept 'transfer' becomes the transitional relationship element at the centre of the PIM template (rule

CP1). The input and output elements, associated to 'transfer' using dependencies (see Fig. 2), are mapped to syntax descriptions (rule CP2). Equally, additional constraints in the CIM are mapped to the PIM semantical descriptions (rule CP4).

Proposals for a mapping language are, similar to the ontology metamodel proposals, currently being requested by the OMG. Graph transformations and graph grammars [28, 29] would suit the need here to formalise the transformation rules. We have used graphs as the visualisation mechanism for ontological models. Graph-based models and CIM-to-PIM transformation semantics are therefore a natural combination. The semantics of a CIM can be seen as a directed labelled graph with nodes (objects and processes) and edges (relationships). The semantics of a PIM can be seen directed labelled graph, where descriptional and transitional roles are distinguished. This is equivalent to a KTS, see Section 3.4. This can be implemented as a graph expansion, where essentially state concepts are introduced. The original CIM can be retrieved by projecting on individual PIMs and then merging all process PIMs into one CIM.

4.3 PIM-to-PSM Mapping

The platform specific model (PSM) is defined in our approach by two separate models: service ontology descriptions to address service discovery and process orchestration and choreography descriptions to address service composition. The corresponding transformation rules for these two aspects – we chose WSMO for

Rule	Aspect	Description
PP1	WSMO	From the WSPO-based PIM, map process relationships to WSMO service concept and fill messageExchange and pre/postCond properties accordingly, see WSMO-template in Fig. 4.
PP1.1	WSMO messageExchange	Map the WSPO in and out objects onto WSMO messageExchange descriptions.
PP1.2	WSMO pre-/postconditions	Map the WSPO pre- and postconditions onto WSMO pre- and postconditions.
PP2	WS-BPEL	The complex WSPO process relationships can be mapped to BPEL processes.
PP2.1	WS-BPEL process partners	For each process create a BPEL partner process.
PP2.2	WS-BPEL orchestration	Convert each process expression into BPEL-invoke activities and the client side BPEL-receive and -reply activities at the server side.
PP2.3	WS-BPEL process activities	Convert the process combinators ';', '+', '!', and '||' to the BPEL combinators sequence, pick, while, and flow, resp.

Fig. 6. Transformation Rules for the PIM-to-PSM Mapping

ontology-based description and WS-BPEL for service orchestration to illustrate this mapping – are presented in Fig. 6.

The WSPO-to-WSMO mapping copies functional properties – both syntax and semantics – to the PSM. Similar to states that are added to CIMs to provide the structure to express process behaviour, here we add structure in form of non-functional aspects to PIMs to add further descriptions for service discovery.

Example 7. The WSMO example in Fig. 4 is the result of mapping the PIM, presented in Fig. 3, to the WSF platform layer according to rule PP1 defined in Fig. 6. Syntactical elements for the interface and semantical capabilities such as pre- and postconditions are directly mapped from the corresponding WSPO elements according to the transformation rules PP1.1 and PP1.2.

The WSPO-to-WS-BPEL mapping converts process expressions into a BPEL business process skeleton, see Fig. 6. WS-BPEL is an implementation language. Process specifications in form of process orchestrations is supported by service engines available from various providers.

5 OMG-Adopted Technologies

Our efforts have to be seen in the context of the OMG approach to MDA. The OMG supports selected modelling notations and platforms through an adoption process. Example of OMG-adopted techniques are UML as the modelling notation and CORBA as the platform [1]. While Web technologies are not (yet) adopted, the need for a specific MDA solution for the Web context is, due to

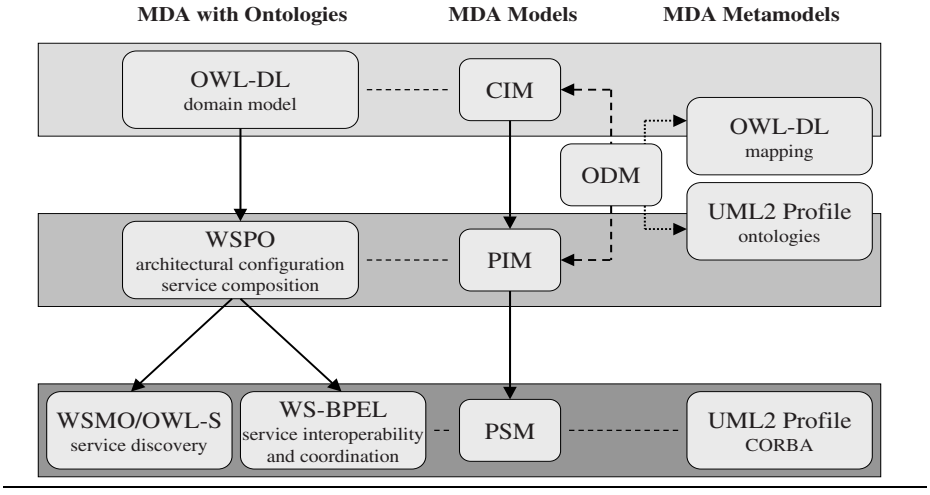


Fig. 7. Overview of MDA and Ontologies – with transformations between the layers and the influence of ODM for the ontology layers

the Web's ubiquity and the existence of standardised and accepted platform and modelling technology, a primary concern.

The current effort of defining and standardising an ontology metamodel (ODM) will allow us to integrate our technique further with OMG standards [9]. ODM will provide mappings to OWL-DL and also a UML profile for ontologies to make UML's graphical notation available. This might lead to a 'Unified Ontology Language' in the future, i.e. OWL in a UML-style notation [30]. A UML profile is about the use of the UML notation, which would allow ontologies to be transformed into UML notation. MOF compliancy for ODM is requested to facilitate tool support. XMI, i.e. production rules using XSLT, can be used to export model representations to XML, e.g. to generate XML Schemas from models using the production rules. We have summarised the MDA framework and compared it with our proposed extension in Fig. 7.

6 Related Work

Service ontologies are ontologies to describe Web services, essentially to support their semantics-based discovery in Web service registries. WSMO [25] and OWL-S [20] are the two predominant examples. WSMO is not an ontology, as OWL-S is, but rather a framework in which ontologies can be created. The Web Service Process Ontology WSPO [23, 24] is also a service ontology, but the focus has shifted here to the support of description and reasoning about service composition and service-based architectural configuration. Both OWL-S and WSPO are or can be written in OWL-DL. WSMO is similar to our endeavour here, since it is a framework of what can be seen as layered ontology descriptions. We have already looked at the technical aspects of WSMO descriptions. WSMO supports the description of services in terms more abstract assumptions and goals and more concrete pre- and postconditions.

We have already discussed the OMG efforts to develop an ontology definition metamodel (ODM) in the previous section, which due to its support of OWL would allow an integration with UML-style modelling. ODM, however, is a standard addressing ontology description, but not reasoning. The reasoning component, which is important here, would need to be addressed in addition to the standard.

Some developments have started exploiting the connection between OWL and MDA. In [31], OWL and MDA are integrated. An MDA-based ontology architecture is defined, which includes aspects of an ontology metamodel and a UML profile for ontologies – corresponding to OMG's ODM. A transformation of the UML ontology to OWL is implemented. The work by [10, 31] and the OMG [1, 9], however, needs to be carried further to address the ontology-based modelling and reasoning of service-based architectures. In particular, the Web Services Framework needs to be addressed in the context of Web-based ontology technology.

7 Conclusions

We have presented a layered ontological modelling and transformation framework for model-driven architecture (MDA). The effort leading towards model-

driven architecture acknowledges the importance of modelling for the architectural design of software systems. We have focused on two aspects:

- Firstly, ontologies are a natural choice to enhance modelling capabilities. While this is recognised in the community, we have exploited the new degree of sharing and ubiquity enabled through Web ontology languages and the reasoning capabilities of logic-based ontology languages.
- Secondly, ontology-based transformations allow the seamless transition from one development focus to another. These omtology transformations allow the integration of domain modelling, architectural design, the description and discovery of services.

Our approach addresses a Web-specific solution, reflecting the current development of the Web Services Framework and the Semantic Web. The primary platform we aim to support is the Web platform with the second Web services generation focusing on processes, utilising the Semantic Web with its ontology technology support. A platform of the expected importance in the future, such as the Web, requires an adequate and platform-specific MDA solution.

A critical problem that has emerged from this investigation is the need for conformity and interoperability. As MDA and the Web as a platform are developed and standardised by two different organisations (the OMG and the W3C, respectively), this can potentially cause problems. The current OMG developments, such as the Ontology Definition Metamodel (ODM), however, aim to reconciliate some of these problems. With ODM soon to be available, our proposed ontologies can, due to their grounding in OWL, be expected to fit into the ODM.

Our aims here were to demonstrate the benefits and the feasibility of layered ontological modelling and transformation for service-oriented architecture, but a number of issues have remained open. We have developed a conceptual modelling and transformation framework. The automation of the transformation processes – central for the success of the technology – needs to be fully implemented. While we have developed some basic reasoning support specific to the architectural modelling activities, more techniques are also possible that exploit the full range of modal reasoning for service description, discovery, and composition and architectural configuration.

References

1. Object Management Group. *MDA Guide V1.0.1.* OMG, 2003.
2. World Wide Web Consortium. *Web Services Framework.* http://www.w3.org/2002/ws, 2004. (visited 08/07/2005).
3. World Wide Web Consortium. *Web Services Architecture Definition Document.* http://www.w3.org/2002/ws/arch, 2003.
4. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications.* Springer-Verlag, 2004.
5. E. Newcomer and G. Lomow. *Understanding SOA with Web Services.* Addison-Wesley, 2005.

6. The WS-BPEL Coalition. *WS-BPEL Business Process Execution Language for Web Services – Specification Version 1.1.* http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, 2004. (visited 08/07/2005).
7. C. Peltz. Web Service orchestration and choreography: a look at WSCI and BPEL4WS. *Web Services Journal*, 3(7), 2003.
8. D.J. Mandell and S.A. McIllraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 227–226. Springer-Verlag, LNCS 2870, 2003.
9. Object Management Group. *Ontology Definition Metamodel - Request For Proposal (OMG Document: as/2003-03-40).* OMG, 2003.
10. D. Gašević, V. Devedžić, and D. Djurić. MDA Standards for Ontology Development – Tutorial. In *International Conference on Web Engineering ICWE2004*, 2004.
11. J. Williams and J. Baty. Building a Loosely Coupled Infrastructure for Web Services. In *Proc. International Conference on Web Services ICWS'2003*. 2003.
12. R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transacions on Software Engineering and Methodology*, 6(3):213–249, 1997.
13. F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, 28(11):1056–1075, 2002.
14. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd Edition).* SEI Series in Software Engineering. Addison-Wesley, 2003.
15. N. Desai and M. Singh. Protocol-Based Business Process Modeling and Enactment. In *International Conference on Web Services ICWS 2004*, pages 124–133. IEEE Press, 2004.
16. R. Kazman, S.J. Carriere, and S.G. Woods. Toward a Discipline of Scenario-based Architectural Evolution. *Annals of Software Engineering*, 9(1-4):5–33, 2000.
17. W3C Semantic Web Activity. Semantic Web Activity Statement, 2004. http://www.w3.org/2001/sw. (visited 06/07/2005).
18. M.C. Daconta, L.J. Obrst, and K.T. Klein. *The Semantic Web.* Wiley, 2003.
19. M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
20. DAML-S Coalition. DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
21. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2003.
22. J. Rao, P. Küngas, and M. Matskin. Logic-Based Web Services Composition: From Service Description to Process Model. In *International Conference on Web Services ICWS 2004*, pages 446–453. IEEE Press, 2004.
23. C. Pahl. An Ontology for Software Component Matching. In M. Pezzè, editor, *Proc. Fundamental Approaches to Software Engineering FASE'2003*, pages 6–21. Springer-Verlag, LNCS 2621, 2003.
24. C. Pahl and M. Casey. Ontology Support for Web Service Processes. In *Proc. European Software Engineering Conference and Foundations of Software Engineering ESEC/FSE'03.* ACM Press, 2003.
25. R. Lara, D. Roman, A. Polleres, and D. Fensel. A Conceptual Comparison of WSMO and OWL-S. In L.-J. Zhang and M. Jeckle, editors, *European Conference on Web Services ECOWS 2004*, pages 254–269. Springer-Verlag. LNCS 3250, 2004.

26. N. Medvidovic and R.N. Taylor. A Classification and Comparison framework for Software Architecture Description Languages. In *Proceedings European Conference on Software Engineering / International Symposium on Foundations of Software Engineering ESEC/FSE'97*, pages 60–76. Springer-Verlag, 1997.
27. T. Payne and O. Lassila. Semantic Web Services. *IEEE Intelligent Systems*, 19(4), 2004.
28. J. Kong, K. Zhang, J. Dong, and G. Song. A Graph Grammar Approach to Software Architecture Verification and Transformation. In *27th Annual International Computer Software and Applications Conference COMPSAC'03*. 2003.
29. L. Baresi and R. Heckel. Tutorial Introduction of Graph Transformation: A Software Engineering Perspective. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. 1st Int. Conference on Graph Transformation ICGT 02*. Springer-Verlag, LNCS 2505, 2002.
30. D. Gašević, V. Devedžić, and V. Damjanović. Analysis of MDA Support for Ontological Engineering. In *Proceedings of the 4th International Workshop on Computational Intelligence and Information Technologies*, pages 55–58, 2003.
31. D. Djurić. MDA-based Ontology Infrastructure. *Computer Science and Information Systems (ComSIS)*, 1(1):91–116, 2004.

Multidimensional RDF*

Manolis Gergatsoulis and Pantelis Lilis

Department of Archive and Library Sciences, Ionian University,
Palea Anaktora, Plateia Eleftherias, 49100 Corfu, Greece
mgerg@otenet.gr, manolis@ionio.gr, pantelis@ionio.gr.gr

Abstract. RDF has been proposed by W3C as a metadata model and language for representing information about resources in WWW. In this paper we introduce *Multidimensional RDF* (or *MRDF*), as an extension of RDF, suitable for representing context-dependent RDF data. In MRDF we have a set of dimensions whose values define different contexts (or worlds) under which different parts of an RDF graph may hold. We define the semantics of MRDF in terms of the semantics of RDF. Additionally, we propose appropriate extensions, suitable for representing MRDF graphs in triples notation and RDF/XML syntax. Finally, we demonstrate how an MRDF graph, embodying a single time dimension, can be used to model the history of a conventional RDF graph.

Keywords: RDF databases, RDF model, Semantic Web, versioning.

1 Introduction

The success of the Internet and the Web during the last few years led to an explosion of the amount of data available. Managing and processing such a huge collection of interconnected data proved to be difficult due to the fact that the Web lacks semantic information. The Semantic Web is a proposal to build an infrastructure of machine-readable metadata (expressing semantic information) for the data on the Web.

In 1998, W3C proposed *Resource Description Framework* (RDF) [5], a metadata model and language which can serve as the basis for such infrastructure. Apart from being a metadata model and language, RDF can also express semantics, empowering the vision of semantic Web. However, RDF falls short when it comes to represent multidimensional information; that is, information that presents different facets under different contexts. Actually, there are many cases where variants of the same information do exist. As a simple example imagine a report that needs to be represented at various degrees of detail and in various languages. A solution would be to create a different document for every possible combination. Such an approach is certainly not practical, since it involves excessive duplication of information. What is more, the different variants are not

* This research was partially co-funded by the European Social Fund (75%) and National Resources (25%) - Operational Program for Educational and Vocational Training (EPEAEK II) and particularly by the Research Program "PYTHAGORAS II".

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1188–1205, 2005.

associated as being parts of the same entity. A similar problem arises when we want to represent the history of a changing RDF graph. In this case, we want to retain information about the past states of the graph as well as about the sequence of changes applied to this graph. Querying on and reasoning about context dependent information is also important.

To the best of our knowledge there is a limited number of papers [19, 14] on this subject. Quite recently, in [14], a temporal extension of RDF, based on the idea of assigning timestamps to RDF triples, is proposed. The same idea is employed in [19] where a technique to track changes of RDF repositories is presented.

In this paper we demonstrate how RDF can be extended so as to be suitable for expressing context-dependent information. The extension that we propose, called *Multidimensional RDF* (or *MRDF* in short), is capable of representing in a compact way multiple facets of information related to the same RDF triple. MRDF employs a set of parameters, called *dimensions*, that are used to determine specific environments, called *worlds*, under which specific pieces of information hold. Conventional RDF graphs, holding under specific worlds, called *snapshots*, can be easily extracted from MRDF graphs. As we demonstrate, MRDF is suitable for representing the history of conventional RDF graphs. Our approach is general enough to be used with more rich formalisms such as RDFS or OWL based ontologies which may also change over time.

The work presented in this paper is based on previous results on providing multidimensional extensions to OEM and XML [20, 11, 21]. The main contributions of this paper can be summarized as follows:

1. A multidimensional extension of RDF, called MRDF, is proposed and its semantics is defined in terms of the semantics of RDF.
2. The notions of *snapshots*, *canonical forms*, and *projections* of MRDF-graphs are defined and some useful properties are discussed.
3. Reification is extended so as to apply to MRDF statements.
4. It is demonstrated that MRDF graphs can be used to represent the history of (conventional) RDF graphs. Basic change operations on RDF graphs are proposed, and it is shown how their effect on RDF triples, can be represented in MRDF-graphs.
5. Extensions of the Triples Notation and the XML/RDF syntax suitable for representing MRDF graphs are proposed.

The rest of the paper is organized as follows: In Section 2, some preliminaries are given. In Section 3, Multidimensional RDF is introduced and some of its properties are discussed. In Section 4, the notions of snapshots, projections and canonical forms of MRDF graphs are introduced. In Section 5, it is illustrated how reification can be extended to MRDF statements. In Section 6, the semantics of MRDF are discussed. In Section 7, extensions of Triples Notation and XML/RDF syntax suitable for representing MRDF graphs are proposed. In Section 8, it is demonstrated how an MRDF graph with a single time dimension can be used to represent the history of conventional RDF graphs. In Section 9, it is shown how history graphs can be stored in RDBMS. In Section 10, related work is discussed. Finally, in Section 11 some hints for future work are given.

2 Preliminaries

Resource Description Framework (RDF) [1, 5] has been proposed by W3C as a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources. RDF is based on the idea of identifying resources using Web identifiers (called *Uniform Resource Identifiers*, or URIs), and describing them in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a *graph* of nodes and arcs representing the resources, with their properties and values.

2.1 RDF Graph

In this subsection we give an abstract representation of the RDF model as graphs, based on some definitions borrowed from [13]. This model consists of an infinite set U (called the *RDF URI references*); an infinite set $B = \{N_j : j \in \mathcal{N}\}$ (called the *blank nodes*); and an infinite set L (called the *RDF literals*). A triple $(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple*. In such a triple, v_1 is called the *subject*, v_2 the *predicate* (also called *property*), and v_3 the *object*. Each triple represents a *statement* of a relationship between the things denoted by the nodes that it links.

Definition 1. *An* RDF graph *is a set of RDF triples. An* RDF-subgraph *is a subset of an RDF-graph. The* universe *of an RDF-graph G, universe(G), is the set of elements of $(U \cup B \cup L)$ that occur in the triples of G. The* vocabulary *of G is the set universe$(G) \cap (U \cup L)$. An RDF-graph is* ground *if it has no blank nodes.*

Graphically, RDF graphs are represented as follows: each triple (a, b, c) is represented by $a \xrightarrow{b} c$. Note that the set of arc labels may have non-empty intersection with the set of node labels. The direction of the arc is significant: it always points toward the object.

A *map* is a function $\mu : (U \cup B \cup L) \to (U \cup B \cup L)$ such that $\mu(u) = u$ and $\mu(l) = l$ for all $u \in U$ and $l \in L$. If G is a graph then $\mu(G)$ is defined as follows: $\mu(G) = \{(\mu(s), \mu(p), \mu(o)) \mid (s, p, o) \in G\}$. A map μ is *consistent* with an RDF-graph G if $\mu(G)$ is also an RDF-graph, i.e. if $(s, p, o) \in G$ then $\mu(s) \in (U \cup B)$, and $\mu(p) \in U$. In this case we say that $\mu(G)$ is an *instance* of G. An instance $\mu(G)$ is a *proper instance* of G if $\mu(G)$ has less blank nodes that G.

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

Let G_1, G_2 be RDF-graphs. Then, the *union* of G_1, G_2, denoted by $G_1 \cup G_2$, is the set theoretical union of their sets of triples. The *merge* of G_1, G_2, denoted by $G_1 + G_2$, is the union $G_1 \cup G_2'$, where G_2' is an isomorphic copy of G_2 whose set of blank nodes is disjoint with the set of blank nodes of G_1.

The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by the subject and the object of

the triple. The assertion of an RDF graph amounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction (logical AND) of the statements corresponding to all the triples it contains. A formal account of the meaning of RDF graphs is given in [2].

Example 1. A fragment of an RDF-graph representing information about a book is shown in Figure 1.

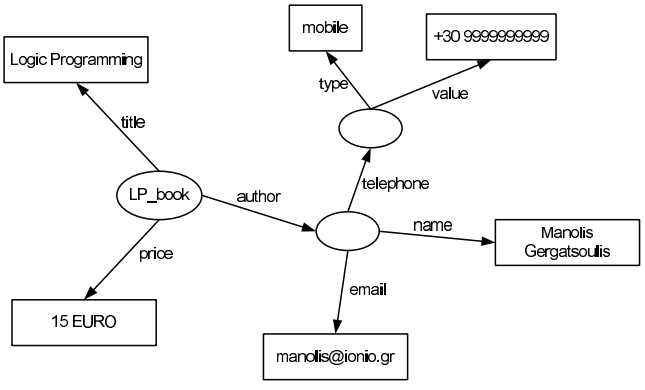


Fig. 1. An RDF-graph

2.2 RDF Triples Notation

Sometimes it is convenient instead of drawing RDF graphs, to have an alternative way of writing down their statements. In the *triples notation*, each statement in the graph is written as a simple triple of the order, subject, predicate, and object.

Example 2. The statements in the RDF-graph of Figure 1 would be written in the triples notation as:

```
LP_book title      "Logic Programming"
LP_book price      "15 EURO"
LP_book author     _:abc
_:abc   email      "manolis@ionio.gr"
_:abc   name       "Manolis Gergatsoulis"
_:abc   telephone  _:def
_:def   type       "mobile"
_:def   value      "+30 9999999999"
```

2.3 RDF/XML Syntax

RDF also provides an XML-based syntax (called RDF/XML) for encoding and exchanging RDF graphs [1, 4].

Example 3. The RDF-graph of Figure 1 is written in RDF/XML as follows:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://www.ionio.gr/LP_book">
       <title>Logic Programming</title>
       <price>15 EURO</price>
       <author rdf:nodeID="abc"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="abc"/>
       <email>manolis@ionio.gr</email>
       <name>Manolis Gergatsoulis</name>
       <telephone rdf:nodeID="def"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="def"/>
       <type>mobile</type>
       <value>+30 9999999999</value>
    </rdf:Description>
</rdf:RDF>
```

3 Adding Dimensions to RDF-Graphs

In this section we introduce the notion of *Multidimensional RDF graph* (*MRDF graph* in short). In a Multidimensional RDF graph, *contexts* may be used to determine the circumstances under which RDF triples are present or not in a graph. RDF triples whose existence depends on a number of dimensions are called *multidimensional RDF triples*.

3.1 Dimensions and Worlds

The notion of *world* is fundamental in our approach (see also [11, 20]). A world represents an environment under which RDF data obtain a substance. A world is determined by assigning values to a set $\mathcal{S}$ of *dimensions*.

Definition 2. *Let $\mathcal{S}$ be a set of* dimension names *and for each $d \in \mathcal{S}$, let $\mathcal{D}_d$, with $\mathcal{D}_d \neq \emptyset$, be the* domain *of d. A* world *w is a set of pairs (d, u), where $d \in \mathcal{S}$ and $u \in \mathcal{D}_d$ such that for every dimension name in $\mathcal{S}$ there is exactly one element in w. The set of all possible worlds is denoted by $\mathcal{U}$.*

In Multidimensional RDF, a triple may have different objects under different worlds. To specify sets of worlds under which a specific node plays the role of the object of a triple we use syntactic constructs called *context specifiers*. Context specifiers qualify the so called *context arcs* which determine the resource that plays the role of the object of a specific property and the worlds under which this holds. The different resources that may be objects of the same property under different worlds are called the *facets* of that object. In the rest of this paper, by $\mathcal{W}(c)$ we denote the set of worlds specified by a context specifier c.

Two context specifies c_1 and c_2 are said to be *equivalent* if they represent the same set of worlds i.e. $\mathcal{W}(c_1) = \mathcal{W}(c_2)$. Two context specifiers c_1 and c_2 are said to be *mutually exclusive* if and only if $\mathcal{W}(c_1) \cap \mathcal{W}(c_2) = \emptyset$.

In this paper we consider the following syntax for context specifiers: A context specifier consists of one or more *context specifier clauses* separated by "|". Each context specifier clause consists of one or more *dimension specifiers* separated by comma. Thus a context specifier clause is of the form:

`dimension_1_specifier, ..., dimension_m_specifier`

where `dimension_i_specifier`, $1 \leq \mathtt{i} \leq \mathtt{m}$, is a *dimension specifier* of the form:

`dimension_name specifier_operator dimension_value_expression`

A *specifier_operator* is one of $=$, $!=$, `in`, `not in`. If the *specifier_operator* is either $=$ or $!=$, the *dimension_value_expression* consists of a single dimension value. Otherwise, if the *specifier_operator* is either `in` or `not in`, the *dimension value expression* is a set of values of the form $\{\mathtt{value}_1, \ldots, \mathtt{value}_k\}$. For linear and discrete domains we will also use the notation $\{a..b\}$ for dimension value expressions to denote the set of all values x such that $a \leq x \leq b$.

Example 4. The following are context specifiers:

```
1) [time=7:45]
2) [lang=Greek, detail in {low,medium} | lang=French, detail=high]
3) [currency in {EURO,USD}, customer_type = student]
4) [season != summer, weekday not in {Saturday,Sunday}]
```

Notice that the set of worlds represented by the second context specifier is $\{(\mathtt{lang} = \mathtt{greek}, \mathtt{detail} = \mathtt{low}), (\mathtt{lang} = \mathtt{greek}, \mathtt{detail} = \mathtt{medium}),$ $(\mathtt{lang} = \mathtt{French}, \mathtt{detail} = \mathtt{high})\}$, while the set of worlds represented by the third context specifier is: $\{(\mathtt{currency} = \mathtt{EURO}, \mathtt{customer\_type} = \mathtt{student}),$ $(\mathtt{currency} = \mathtt{USD}, \mathtt{customer\_type} = \mathtt{student})\}$. Concerning the fourth context specifier we have to take into account the domains of the dimensions `season` and `weekday` in order to find the set of worlds that it represents.

Context specifiers impose constraints that restrict the set of worlds under which an entity holds. In this sense, if a context specifier does not contain a dimension specifier for a specific dimension then no constraint is imposed on the values of this dimension. Thus, the context specifier [ ] represents the set of all possible worlds $\mathcal{U}$.

3.2 Multidimensional RDF-Graphs

In Multidimensional RDF-graphs we will use the following sets of symbols: a set U of RDF URI references, a set B of blank nodes, a set L of RDF literals, a set M of *multidimensional nodes*, and a set C of context specifiers.

A triple $(v_1, v_2, v_3) \in (U \cup B) \times U \times (M \cup U \cup B \cup L)$ is called *statement triple*, while a triple $(v_1, c, v_3) \in M \times C \times (U \cup B \cup L)$ is called a *context triple*.

Definition 3. *A* Multidimensional RDF-graph *(*MRDF-graph*) is a set $\mathcal{S}_t \cup \mathcal{C}_t$, where $\mathcal{S}_t$ is a set of statement triples and $\mathcal{C}_t$ is a set of context triples, such that:*

1. *For every context triple* $(v_3, c, v_4) \in \mathcal{C}_t$ *there exist a node* $v_1 \in (U \cup B)$ *and a node* $v_2 \in U$ *such that* $(v_1, v_2, v_3) \in \mathcal{S}_t$.
2. *For every statement triple* $(v_1, v_2, v_3) \in \mathcal{S}_t$ *for which* v_3 *is a multidimensional node, there exist a context specifier* $c \in C$ *and a node* $v_4 \in (U \cup B \cup L)$ *such that* $(v_3, c, v_4) \in \mathcal{C}_t$.

Example 5. A Multidimensional RDF-graph is shown in Figure 2. This graph is a variant of the RDF graph in Figure 1. Some parts of this graph are context-dependent. In particular, we have two dimensions; the dimension `time` (abbreviated as `t`) and the dimension `customer_type` (abbreviated as `ct`). The value of the property `price` of the book is `15 EURO` for all worlds in which the value of the dimension `ct` is `student` while the value of the same property is `20 EURO` for all worlds in which the value `ct` is `library`. The value of the property `email` is time-dependent. This value is `manolis@ionio.gr` for the time points in the interval {`start..t1-1`}, where the reserved words `start` represents the beginning of time, while the value of the property is `mgerg@otenet.gr` for the time points in the interval {`t1..now`}, where `now` represents the current time. Finally, notice that the property `telephone` has meaning only under the worlds in which the value of the dimension `ct` is `library` and the value of the dimension `t` is in the interval {`t2..now`}. This means that from the time point `t2` the mobile telephone of the author is available only for customers that are libraries.

Notice that in an MRDF-graph statement triples are represented by thin lines while context triples are represented by thick lines.

An MRDF graph G is said to be *deterministic*, if for every multidimensional node m in G, the context specifiers of all context triples departing from m are mutually exclusive each other. Although non-deterministic MRDF-graphs may have interesting properties, in this paper we focuss mainly on deterministic MRDF-graphs.

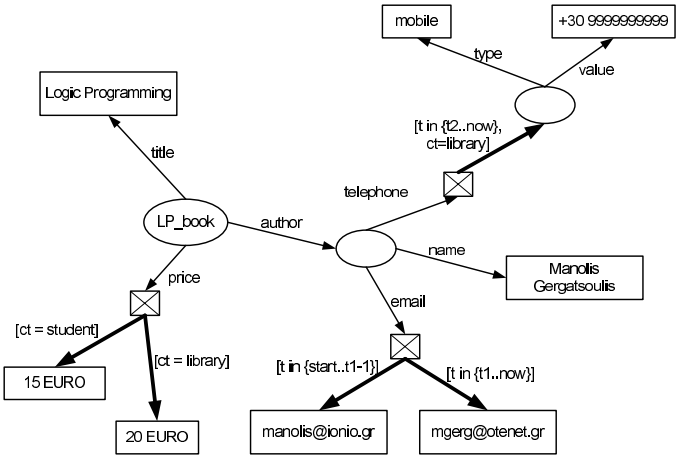


Fig. 2. A Multidimensional RDF-graph

4 Snapshots, Projections and Canonical Forms of MRDF Graphs

4.1 RDF Snapshots of MRDF-Graphs

A Multidimensional RDF-graph G can be seen as a compact representation of a set of conventional RDF-graphs called the *snapshots* of G:

Definition 4. *Let $G = (\mathcal{S}_t \cup \mathcal{C}_t)$ be an MRDF-graph and w be a world. We define the* snapshot of G under w*, denoted by $Snap(G, w)$, as follows: $Snap(G, w) = \{(r1, p, r2) \mid r2 \in U \cup B \cup L$ and $(r1, p, r2) \in \mathcal{S}_t\} \cup \{(r1, p, r2) \mid \exists m \in M, \exists c \in C$ such that $(r1, p, m) \in \mathcal{S}_t$ and $(m, c, r2) \in \mathcal{C}_t$ and $w \in \mathcal{W}(c)\}$.*

Notice that to each multidimensional node of a deterministic MRDF-graph corresponds at most one RDF triple at each world. In non-deterministic MRDF-graphs, multiple RDF triples may correspond to every multidimensional node.

According to the above definition, a multidimensional RDF-graph G can be seen as a compact representation of the set $Snap(G,\mathcal{U}) = \{Snap(G, w) \mid w \in \mathcal{U}\}$, where $\mathcal{U}$ is the set of all possible worlds, that is, $Snap(G,\mathcal{U})$ represents the set of all (conventional) RDF-graphs each of them holding under a specific world.

Example 6. Consider the world $w = \{$`t = t2`$^+$`, ct = library`$\}$, where `t2`$^+$ is a time point such that `t1 < t2 < t2`$^+$. Then the snapshot of the MRDF-graph in Figure 2 under the world w, is the RDF-graph shown in Figure 3.

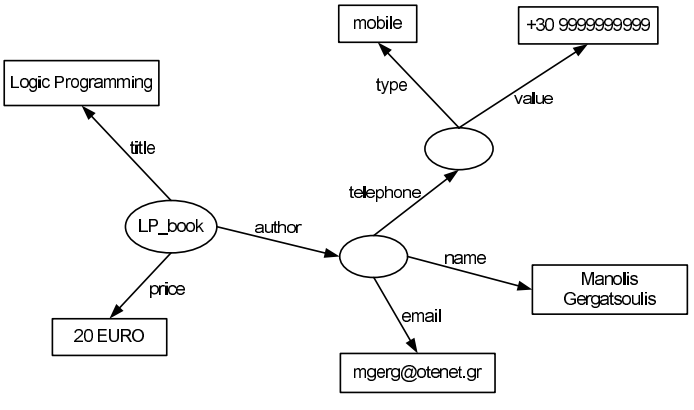


Fig. 3. A snapshot of the Multidimensional RDF-graph in Figure 5

Based on the notion of snapshots we define equivalence of MRDF-graphs:

Definition 5. *Let G_1 and G_2 be MRDF graphs. We say that G_1* is equivalent *with G_2 if and only if for every world w, $Snap(G_1, w) \cong Snap(G_2, w)$.*

4.2 Projections of MRDF Graphs

Another useful operation on MRDF-graphs is *projection* with respect to a set of dimensions. Let S be a set of dimensions and c a context specifier. Then the *projection of a context specifier* c with respect to S is a context specifier c' obtained from c by eliminating all dimension specifiers of c whose dimension name does not belong to S.

Definition 6. *Let $G = (\mathcal{S}_t \cup \mathcal{C}_t)$ be a multidimensional RDF-graph and S be a set of dimensions. Then the* projection of G with respect to S, *denoted by $Proj(G, S)$, is defined as follows:*

$$Proj(G,S) = \mathcal{S}_t \cup \{(m, c', r2) \mid (m, c, r2) \in \mathcal{C}_t \text{ and } c' \text{ is the projection of } c \text{ with respect to } S\}.$$

Projecting an MRDF-graph means that we remove all constraints concerning the values of the dimensions not belonging to S. Notice that the projection of a deterministic MRDF-graph G, may be a non-deterministic MRDF-graph.

4.3 Canonical Form of MRDF Graphs

In this section we define the notion of *canonical form* of an MRDF graph. In a canonical MRDF graph all statement arcs point to multidimensional nodes.

Definition 7. *An MRDF-graph $G = (\mathcal{S}_t \cup \mathcal{C}_t)$ is said to be in* canonical form *if for every statement triple $(v_1, v_2, v_3) \in \mathcal{S}_t$, v_3 is a multidimensional node.*

Definition 8. *Let $G = (\mathcal{S}_t \cup \mathcal{C}_t)$ be an MRDF-graph. The* canonical representation *$Can(G)$ of G is an MRDF-graph obtained from G by replacing each statement triple $(v_1, v_2, v_3) \in \mathcal{S}_t$ for which $v_3 \notin M$, by a pair of a statement triple and a context triple of the form (v_1, v_2, m) and $(m, [\,], v_3)$ respectively, where m is a fresh multidimensional node in M.*

This construction of the canonical representation of an MRDF-graph according to the above definition is shown in Figure 4. Recall that the context specifier $[\,]$ represents the set of all possible worlds $\mathcal{U}$.

Fig. 4. Transforming an MRDF-graph in canonical form

Example 7. The canonical representation of the graph in Figure 2 is shown in Figure 5. This graph is obtained by replacing each statement triple in the graph of Figure 2 leading to a non-multidimensional node, by a pair of a statement triple followed by a context triple as described above (and is depicted in Figure 4).

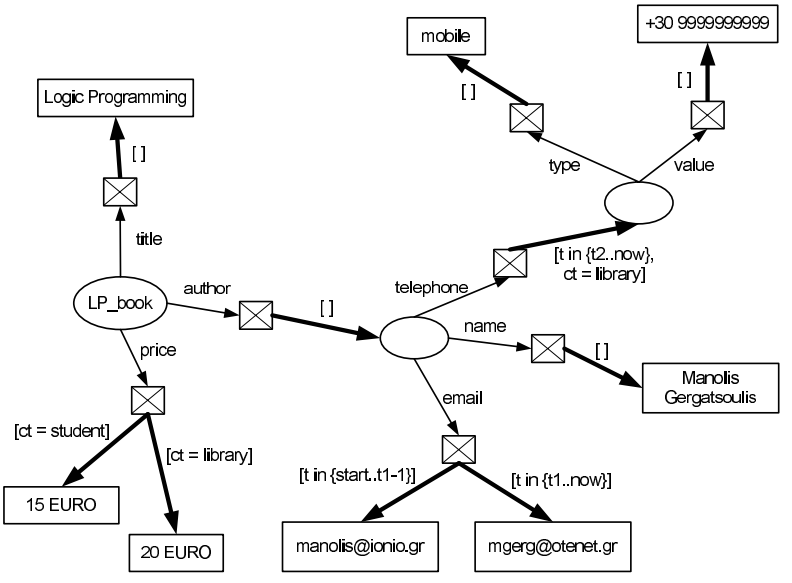


Fig. 5. The canonical representation of the MRDF-graph in Figure 2

The following lemma demonstrates that an MRDF-graph and its canonical representation are equivalent graphs.

Lemma 1. *Let G be an MRDF-graph and $Can(G)$ be its canonical representation. Then G and $Can(G)$ are equivalent.*

Proof. It is easy to prove that for every world w, $Snap(G, w) \cong Snap(Can(G), w)$.

5 Reification in Multidimensional RDF-Graphs

RDF applications often need to describe other RDF statements or, in general, to express statements about other statements. It is useful, for instance, to record in RDF, information about when statements were made, or who made them. RDF embodies [1] a built-in vocabulary intended for describing RDF statements. A statement description using this vocabulary is called a *reification* of the statement. The RDF reification vocabulary consists of the type `rdf:Statement`, and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`. The use of this vocabulary to describe RDF statements is shown in Figure 6 where the triple $(R1, P, R2)$ shown in Figure 6(a) is represented by the RDF graph in Figure 6(b). This graph says that the anonymous resource is an RDF statement, the subject of the statement is $R1$, its predicate is P, and its object is $R2$.

For MRDF an extension of the RDF reification mechanism is needed. Such an extension is shown in Figure 7. Since in MRDF we may have multiple facets of an object of a triple, corresponding to different sets of worlds, the property

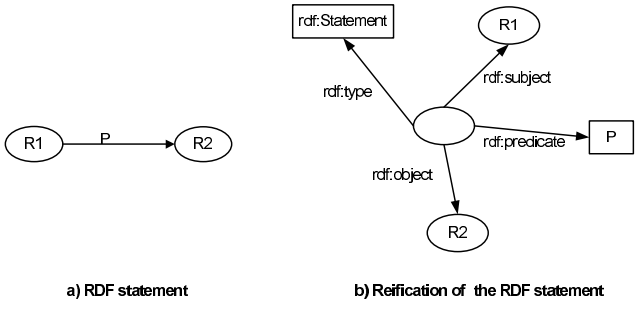


Fig. 6. Reification in RDF

`rdf:object` is now represented by a statement triple whose arc points to a multidimensional node. From this multidimensional node depart different context triples each of them leading to a different facet of the statement's object. Notice also that we now use a type `mrdf:Statement` which states that the reified statement is not a conventional but a multidimensional one.

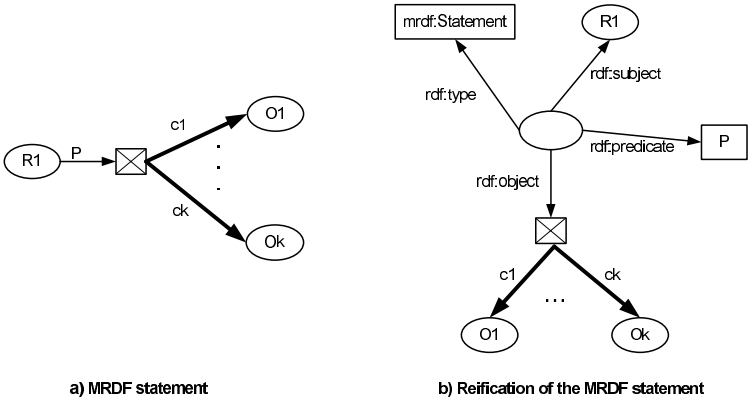


Fig. 7. Reification in Multidimensional RDF

It is important to note that the definition of snapshots has to be slightly modified so as to take into account the presence of reification. However, we will not discuss it further here due to space limits.

6 Semantics of MRDF

In this section we define the semantics of MRDF in terms of the semantics of RDF.

Definition 9. *Let F be an MRDF-graph, G be an RDF-graph, w be a world, and W be a set of worlds. We say that F* entails *G in the world w, denoted by $F \models_w G$, if and only if $Snap(F, w) \models G$. We say that F* entails *G in a set of worlds W, denoted by $F \models_W G$, if and only if for every world $w \in W$, $Snap(F, w) \models G$.*

The following lemma can be easily proved.

Lemma 2. *Let F be an MRDF-graph and $Can(F)$ be its canonical form. Then for every RDF graph G and every world w, $F \models_w G$ if and only if $Can(F) \models_w G$.*

7 Triples Notation and RDF/XML Syntax for MRDF

Syntactic constructs suitable for representing MRDF-graphs are presented in this section.

7.1 Extended Triple Notation

In order to express MRDF graphs in triples notation it is necessary to extent this notation in order to be capable of representing context triples. The extension that we propose retains the basic structure of the triples but allows their third component to be a list, called *object list*. Object list is used when the object of a statement triple is a multidimensional node. In this case the third component contains pairs of values. The first value of each pair is a context specifier while the second value is the object (resource/literal) corresponding to that specifier.

Example 8. The representation in the extended triple notation of the MRDF graph in Figure 2 is as follows:

```
LP_book title "Logic Programming"
LP_book price [([ct = student],"15 EURO"), ([ct = library],"20 EURO")]
LP_book author _:abc
_:abc email [([t in {start..t1-1})],"manolis@ionio.gr"),
             ([t in {t1..now}],"mgerg@otenet.gr")]
_:abc name "Manolis Gergatsoulis"
_:abc telephone [([t in {t2..now},ct = library],_:def)]
_:def type "mobile"
_:def value "+30 9999999999"
```

7.2 RDF/XML Syntax for MRDF

In order to express MRDF in RDF/XML syntax we need an extension of XML capable of expressing contexts. Such an extension of XML is *Multidimensional XML* (or MXML), which has been proposed in [11]. In Example 9 below we illustrate how MRDF can be expressed in MXML/RDF syntax.

Example 9. The following RDF/MXML document represents the MRDF graph in Figure 2:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://www.ionio.gr/LP_book">
       <title>Logic Programming</title>
       <@price>
         [ct=student]<price>15 EURO</price>[/]
         [ct=library]<price>20 EURO</price>[/]
       </@price>
       <author rdf:nodeID="abc"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="abc"/>
       <@email>
         [t in {start..t1-1}]<email>manolis@ionio.gr</email>[/]
         [t in {t1..now}]<email>mgerg@otenet.gr</email>[/]
       </@email>
       <name>Manolis Gergatsoulis</name>
       <@telephone>
         [t in {t2..now},ct=library]<telephone rdf:nodeID="def"/>[/]
       </@telephone>
    </rdf:Description>
    <rdf:Description rdf:nodeID="def"/>
       <type>mobile</type>
       <value>+30 9999999999</value>
    </rdf:Description>
</rdf:RDF>
```

It should be noted that instead of using MXML we can use conventional (but more verbose) XML syntax for representing MRDF-graphs (see [10]).

8 Representing Changes in RDF Graphs Using MRDF

As described previously, MRDF is a general formalism and powerful enough to represent context-dependent RDF data that may occur in real world RDF applications. In this section we demonstrate that MRDF graphs can be used as a formalism for tracking the history of conventional RDF-graphs. We assume the following scenario: the user manipulates an RDF graph and applies changes to it, at specific time points. The changes are described through specific primitives called *basic change operations*. In order to keep track of the sequence of changes and in particular of the sequence of the (conventional) RDF graphs obtained by applying these changes, the system keeps a Multidimensional RDF graph, called the *History Graph*, which encodes all these changes applied by the user to the conventional RDF graph. The History Graph employs a single dimension `t` representing time. For this dimension, we assume a time domain T which is linear and discrete. We also assume a reserved value `start`, such that `start` $<$ `t` for every `t` $\in T$, representing the *beginning of time*, and a reserved value `now`, such that `t` $<$ `now` for every `t` $\in T$, representing the *current time*. Notice that, in our scenario, the user is only capable to apply change operations on a conventional RDF graph being unaware of the History Graph that lies beneath.

8.1 Basic Change Operations on RDF triples

We consider three primitive change operations on RDF graphs, namely *update*, *delete*, and *insert*, and demonstrate how their effect on RDF-graph can be mapped into changes on the underlying History Graph:

a) **Update:** Update operation can be used to change the value of a property (i.e. the object of a triple). Updating a triple can be seen (at the level of the RDF graph being updated) as the replacement of the triple with another triple which has the same subject and predicate but different object. The way that update operation affects the underlying History Graph is depicted in Figure 8(a). The value of the property P in the triple on the left part of the figure is updated at time `t1` from `R2` to the new value `R3`. The MRDF representation of this operation is shown on the right side of the figure. The multidimensional node has now two facets. The first one is valid in all time points of the interval {`start..t1-1`}, while the second is valid for all time points in the interval {`t1..now`}.

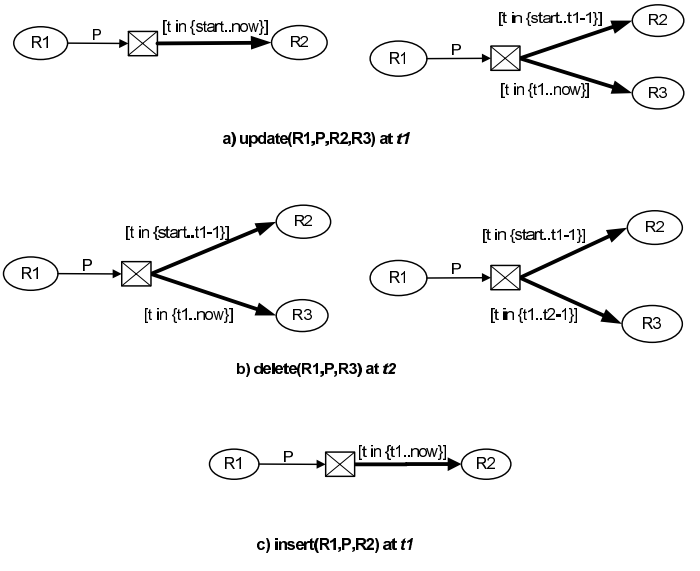


Fig. 8. Representation of the basic change operations in the History Graph

Note that a subsequent update of the same statement at a time point `t2` will be represented in the History Graph as follows: a) by simply adding a new context triple departing from the same multidimensional node and holding in the interval {`t2..now`} and b) by changing the value of the time dimension `t` of the most recent context triple from {`t1..now`} to {`t1..t2-1`}. Note also that the resource $R3$ may be a new resource which is introduced by the update operation or it may be a resource that already exists in the graph. In both cases the update operation results in adding a context arc from the multidimensional node to $R3$.

b) **Delete:** The deletion of a triple $(R1, P, R3)$ from the RDF graph at time `t2`, is represented in the History Graph by simply changing the end time point of the most recent interval from `now` to `t2-1`, as shown in Figure 8(b). Note that, if the deleted triple is a conventional triple in the History Graph, then deletion is modeled by first obtaining the canonical form of the triple and then applying the process described above to the caninical triple as described above.
c) **Insert:** As depicted in Figure 8(c), the new triple $(R1, P, R2)$ inserted at the time point `t1`, is modeled in the History Graph by adding a new statement triple followed by a single context triple holding during the interval {`t1..now`}.

Notice that the triple being inserted on the RDF graph may refer either to resources (subject and/or object) that already exist in the RDF-graph or to new resources that are added by the insert operation.

9 Storing the History Graphs in RDBMS

A simple relational database schema can be easily designed for storing the History Graphs using an RDBMS. For this we assume that the graph is in its canonical form. Such a graph can be stored in a database which has two relations, namely `statement` and `context` to store the statement and the context triples respectively. The schema of these relations is as follows:

`statement(Subject,Predicate,MultidimensionalNode)`: where `Subject` is the subject, `Predicate` is the predicate and `MultidimensionalNode` is the multidimensional node identifier.
`context(MultidimensionalNode,Object,S,E)`: where `MultidimensionalNode` is the multidimensional node identifier from which the arc departs, `Object` is the object, `S` is the start time point and `E` is the end time point of the time interval.

Notice that this schema is appropriate only for MRDF which has only one dimension which takes as values time intervals.

Concerning the RDF/XML syntax it is important to note that in the case of the History Graph, where we have only a single time dimension, we could use (instead of MXML), a temporal extension of XML. Such a temporal extension of XML can be found in [22, 23], where two extra attributes are added to XML elements, namely `vstart` and `vend`, representing the end points of the time interval in which this elements version is valid.

10 Related Work

The Multidimensional extension to RDF proposed in this paper is based on similar ideas to that on which Multidimensional OEM [20] and the Multidimensional XML [11] are based. However, to the best of our knowledge, there is no other research work in the direction of incorporating dimensions in RDF. Quite recently [14], Temporal RDF, a transaction time extension of RDF has been proposed. However, our approach is more general than Temporal RDF which may be considered as a special case of Multidimensional RDF, since we allow multiple dimensions (and even multiple time dimensions).

To the best of our knowledge, only a few papers refer to the problem of representing changes in RDF databases. One approach to this problem has been proposed in [19]. Their model is based on the admission that an RDF statement is the smallest manageable piece of knowledge in an RDF repository. They propose only two basic operations, addition and removal, since they argue that an RDF statement cannot be changed, it can only be added or removed. In their approach, they used versions as the labeled states of the repository. However, our approach is more general and flexible than the approach in [19] as besides addition and deletion we also introduce an update operation. In this way different resources that are in fact different versions of an object of a property are grouped together and its relationship is recorded. Besides, our representation formalism is more general as it allows multiple dimensions (which might be multiple times such as *valid time* or *transaction time*). Consequently, in our approach one can encode multiple versioning information not only with respect to time but also to other context parameters such as language, degree of detail, geographic region etc.

Some related research to the problem of RDF versioning has also been done in the field of ontology versioning. In [16, 15], Ontoview, a web-based management system for evolving ontologies in the Web, is used. Ontoview has the ability to compare ontologies at a structural level. It finds changes in ontologies and it visualizes them, helping the user to specify the conceptual implication of the differences. In [17] a component-based framework for ontology evolution is proposed. This framework integrates a description of different representations about change information. Thus, they present an ontology of change operations, which is the kernel of their framework. The problem of managing (storing, retrieving and querying) multiple versions of XML documents is also examined in [8, 9]. Recently, an approach of representing XML document versions was proposed [22, 23]. The basic idea is to add two extra attributes, namely `vstart` and `vend`, that represent the time interval in which the corresponding version of the element is valid. Temporal extensions of XML have also been proposed in [6, 12].

The problem of representing and querying changes in semistructured data has also been studied in [7], where *Delta OEM* (DOEM in short), a graph model that extends OEM with annotations containing temporal information, was proposed. Four basic change operations on OEM graphs, namely *creNode*, *updNode*, *addArc*, and *remArc* are considered. Those operations are mapped to annotations, which are tags attached to nodes or edges, containing information that encodes the history of changes for these nodes or edges. Recently [21], Multidimensional OEM [20], has been proposed as a formalism for representing the history of time-evolving semistructured data. Finally, in [10], the authors propose the use of Multidimensional XML [11] for the representation of the history of XML documents.

11 Discussion and Future Work

Investigation of other real application domains to demonstrate usefulness of multidimensional RDF is between our plans for future work. An attempt to explore

such applications is described in [18], where the problem of representing and manipulating time-dependent information in collection-level cultural metadata is investigated. In that paper, MRDF employing two independent time dimensions is used as a formalism to enriches a metadata application profile for the collection-level description of cultural collections, with the ability of time representation and manipulation.

Investigation of query languages and inference systems for MRDF repositories when RDFS vocabulary is used [3], are important problems for future work. The study of the semantics of non-deterministic MRDF-graphs and their applications are also interesting problems.

References

1. RDF Primer (W3C Recommendation, 10 February 2004). http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004.
2. RDF Semantics (W3C Recommendation, 10 February 2004). http://www.w3.org/TR/2004/REC-rdf-mt-20040210/, 2004.
3. RDF Vocabulary Description Language 1.0: RDF Schema (W3C Recommendation, 10 February 2004). http://www.w3.org/TR/2004/REC-rdf-schema-20040210, 2004.
4. RDF/XML Syntax Specification (Revised), (W3C Recommendation 10 February 2004). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, 2004.
5. Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation, 10 February 2004). http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004.
6. T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In M. T. Ibrahim, J. Kung, and N. Revell, editors, *Database and Expert Systems Applications, 11th International Conference (DEXA 2000), London, UK, Sept. 4-8, Proceedings*, Lecture Notes in Computer Science (LNCS) 1873, pages 334–344. Springer-Verlag, Sept. 2000.
7. S. S. Chawathe, S. Abiteboul, and J. Widom. Managing Historical Semistructured Data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999.
8. S.-Y. Chien, V. Tsotras, and C. Zaniolo. Efficient Schemes for Managing Multiversion XML Documents. *The VLDB Journal*, 11(4):332–353, 2002.
9. S.-Y. Chien, V. J. Tsotras, C. Zaniolo, and D. Zhang. Efficient Complex Query Support for Multiversion XML Documents. In *Advances in Database Technology - EDBT 2002, Proceedings of the 8th Conference on Extending Database Technology*, Lecture Notes in Computer Science (LNCS) Vol 2287, pages 161–178. Springer-Verlag, 2002.
10. M. Gergatsoulis and Y. Stavrakas. Representing Changes in XML Documents using Dimensions. In Z. Bellahsene, A. B. Chaudhri, E. Rahm, M. Rys, and R. Unland, editors, Database and XML Technologies, 1st International XML Database Symposium, XSym' 03, Berlin, Germany, September 2003, Proceedings, Lecture Notes in Computer Science (LNCS), Vol. 2824, pages 208–222. Springer-Verlag, 2003.
11. M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating Dimensions to XML and DTD. In H. C. Mayr, J. Lanzanski, G. Quirchmayr, and P. Vogel, editors, Database and Expert Systems Applications (DEXA' 01), Munich, Germany, September 2001, Proceedings, Lecture Notes in Computer Science (LNCS), Vol. 2113, pages 646–656. Springer-Verlag, 2001.

12. F. Grandi and F. Mandreoli. The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. In T. M. Yakhno, editor, *Advances in Information Systems. First International Conference (ADVIS'02), Izmir, Turkey, 25-27 October*, pages 294–303, 2000.
13. C. Gutiérrez, C. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of DataBase Systems*, pages 95–106. ACM Press, 2004.
14. C. Gutiérrez, C. Hurtado, and A. Vaisman. Temporal RDF. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings,*, volume 3532 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.
15. M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Finding and characterizing changes in ontologies. In *Proceedings of the 21st International Conference on Conceptual Modelling (ER'02)*, volume 2503 of *Lecture Notes in Computer Science*, pages 79–89. Springer-Verlag, Heidelberg, 2002.
16. M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. Ontology versioning and change detection on the web. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW'02)*, volume 2473 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, Heidelberg, 2002.
17. M. Klein and N.F. Noy. A component-based framework for ontology evolution. Technical Report IR-504, Department of Computer Science, Vrije Universiteit, Amsterdam, March 2003.
18. P. Lilis, E. Lourdi, C. Papatheodorou, and M. Gergatsoulis. A metadata model for representing time-dependent information in cultural collections. Submitted for publication, 2005.
19. D. Ognyanov and A. Kiryakov. Tracking Changes in RDF(S) Repositories. In A. Gomez-Perez and V. Richard Benjamins, editors, *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings.* Lecture Notes in Computer Science (LNCS) 2473, pages 373–378. Springer-Verlag, 2002.
20. Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In A. B. Pidduck, J. Mylopoulos, C. Woo, and T. Oszu, editors, *Advanced Information Systems Engineering, 14th International Conference (CAiSE'02), Toronto, Ontario, Canada, May 2002. Proceedings.*, Lecture Notes in Computer Science (LNCS), Vol. 2348, pages 183–199. Springer-Verlag, 2002.
21. Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, and V. Zafeiris. Representing and Querying Histories of Semistructured Databases Using Multidimensional OEM. *Information Systems*, 29(6):461–482, 2004.
22. F. Wang and C. Zaniolo. Temporal Queries in XML Document Archives and Web Warehouses. In *Peoceedings of the 10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003), 8-10 July 2003, Cairns, Queensland, Australia*, pages 47–55. IEEE Computer Society, 2003.
23. Fusheng Wang. *Efficient Support for Queries and Revisions in XML Document Archives.* PhD thesis, Computer Science Department, University of California, Los Angeles (UCLA), August 2004.

GeRoMe: A Generic Role Based Metamodel for Model Management

David Kensche[1], Christoph Quix[1], Mohamed Amine Chatti[1], and Matthias Jarke[1,2]

[1] RWTH Aachen University, Informatik V (Information Systems), 52056 Aachen, Germany
[2] Fraunhofer FIT, Schloss Birlinghoven, 53574 St. Augustin, Germany
{kensche, quix, chatti, jarke}@cs.rwth-aachen.de

Abstract. The goal of *Model Management* is the development of new technologies and mechanisms to support the integration, evolution and matching of models. Such tasks are to be performed by means of a set of model management *operators* which work on models and their elements, without being restricted to a particular metamodel (e.g. the relational or UML metamodel).

We propose that generic model management should employ a generic metamodel (GMM) which serves as an abstraction of the features of particular metamodels while preserving the semantics of its different elements. A naive generalization of the elements of concrete metamodels in generic metaclasses would loose some of the specific features of the metamodels, or yield a prohibitive number of metaclasses in the GMM. To avoid these problems, we propose the *Generic Role Based Metamodel GeRoMe* in which each model element is *decorated* with a set of role objects that represent specific properties of the model element. Roles may be added to or removed from elements at any time, which enables a very flexible and dynamic yet accurate definition of models.

Roles constitute to operators different views on the same model element. Thus, operators concentrate on features which affect their functionality but may remain agnostic about other features. Consequently, these operators can use polymorphism and have to be implemented only once using *GeRoMe*, and not for each specific metamodel. We verified our results by implementing *GeRoMe* and a selection of model management operators using our metadata system ConceptBase.

1 Introduction

Design and maintenance of information systems require the management of complex models. Research in *model management* aims at developing technologies and mechanisms to support the integration, merging, evolution, and matching of models. These problems have been addressed for specific modeling languages for a long time. Model management has become an active research area recently, as researchers now address the problem of *generic* model management, i.e. supporting the aforementioned tasks without being restricted to a particular modeling language [3, 4]. To achieve this goal, the definition of a set of *generic* structures representing models and the definition of *generic* operations on these structures are required.

According to the IRDS standard [10], metamodels are *languages* to define models. Examples for metamodels are *XML Schema* or the *UML Metamodel*. Models are the

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1206–1224, 2005.

description of a concrete application domain. Within an (integrated) information system, several metamodels are used, a specific one for each subsystem (e.g. DB system, application). Thus, the management of models in a generic way is necessary.

1.1 The Challenge: A Generic Mechanism for Representing Models

This paper addresses the first challenge mentioned in [4], the development of a mechanism for representing models. Since the goal is the support of *generic* model management, this has to be done in some generic way. Currently, model management applications often use a generic graph representation but operators have to be aware of the employed metamodel [5, 8, 13]. While a graph representation is often sufficient for the purpose of finding correspondences between schemas, it is not suitable for more complex operations (such as merging of models) as it does not contain detailed semantic information about relationships and constraints. For example, in [15] a generic (but yet simple) metamodel is used that distinguishes between different types of associations in order to merge two models. A more detailed discussion about the related work on the representation of models is given in section 2.

The intuitive approach to develop a truly generic metamodel (GMM) identifies abstractions of the metaclasses of different metamodels. Its goal is to define a comprehensive set of generic metaclasses organized in an inheritance lattice. Each metaclass in a given concrete metamodel then has to be mapped to a unique metaclass of the GMM.

The sketched approach exhibits a prohibitive weak point: elements of particular metamodels often have semantics that overlap but is neither completely different nor equivalent. For example, a generic Merge operator has to merge elements such as classes, relations, entity types and relationship types. All of these model elements can have attributes and should therefore be processed by the same implementation of an operator. In this setting, such polymorphism is only possible if the given model elements are represented by instances of the same metaclass in the GMM, or at least by instances of metaclasses with a common superclass. Thus, one has to choose the features of model elements which are combined in one metaclass.

Actually, in each metamodel there may be elements incorporating an entirely new combination of such aspects. One approach to cope with this problem is to focus on the "most important" features of model elements while omitting such properties which are regarded as less important. But to decide which properties are important and which are not results in loss of information about the model.

All properties of model elements could be retained if the GMM introduced a set of metaclasses as comprehensive as possible and combined them with multiple inheritance such that any combination of features is represented by a distinct metaclass. Despite the modeling accuracy of such a GMM, it will suffer from another drawback, namely that it leads to a combinatorial explosion in the number of sparsely populated intersection classes which add no new state.

1.2 Our Solution: Role Based Modeling

In such cases, a role based modeling approach is much more promising. In role based modeling, an object is regarded as playing roles in collaborations with other objects.

Applied to generic metadata modeling this approach allows to *decorate* a model element with a combination of multiple predefined aspects, thereby describing the element's properties as accurately as possible while using only metaclasses and roles from a relatively small set. In such a GMM, the different features of a model element (e.g. it is not only an *Aggregate* but also an *Association*) are only different views on the same element. During model transformations, an element may gain or lose roles, thereby adding and revoking features. Thus, the combinatorial explosion in the number of metaclasses is avoided but nevertheless most accurate metadata modeling is possible.

Therefore, the GMM proposed in this work retains these characteristics by employing the *role based* modeling approach, resulting in the *Generic Role Based Metamodel GeRoMe*. Implementations of model management operators can assure that model elements have certain properties by checking whether they play the necessary roles. At the same time the operator remains agnostic about any roles which do not affect its functionality. Thus, while role based metamodeling allows to formulate accurate models, the models appear to operators only as complex as necessary. *GeRoMe* will be used only by model management applications; users will use their favorite modeling language.

The definition of the GMM requires a careful analysis and comparison of existing metamodels. Since it has to be possible to represent schemata in various metamodels in order to allow generic model management, we analyzed five popular yet quite different metamodels (Relational, EER, UML, OWL DL, and XML Schema). We identified the common structures, properties, and constraint mechanisms of these metamodels. This part of our work can be seen as an update to the work in [9], in which several semantic database modeling languages have been compared.

The paper is structured as follows. Section 2 provides some background information on model management and role based modeling, and presents a motivating scenario. In section 3, we analyze and compare existing metamodels and derive the *Generic Role Based Metamodel GeRoMe*. Section 4 shows several examples of models in different metamodels represented in *GeRoMe*. Section 5 explains how model management operations can be performed using *GeRoMe*. As an example, we describe some atomic operations necessary for the transformation of an EER model into a a relational model. The implementation of our model management prototype is discussed in section 6. Finally, section 7 summarizes our work and points out future work.

2 Background and Motivation

2.1 Model Management

Model management aims at providing a formalization for the definition and modification of complex models [4]. To achieve this goal, a model management system has to provide definitions for *models* (i.e. schemas represented in some metamodel), *mappings* (i.e. relationships between different models), and *operators* (i.e. operations that manipulate models and mappings). There have been earlier approaches to model management [1, 12], which did address especially the transformation of models between different metamodels. Model management has become more important recently, as the integration of information systems requires the management of complex models. The most important operations in model management are Merge (integration of two models),

Match (creating a mapping between two models), Diff (finding the differences between two models), and ModelGen (generating a model from another model in a different metamodel representation).

Rondo [13] is the first complete prototype of model management. It represents models as directed labeled graphs. Each node of such a graph denotes one model element, e.g. an XML complex type or relational table. A model is represented by a set of edges between these nodes. A model element's type (Table, Column, Class, ...) is also specified by such an edge with the label *type*. Furthermore, types of attributes are specified by other dedicated edges, e.g. *SQLtype*. For each of the supported metamodels a different set of types is available. Although the models are represented in a generic graph structure, the implementation of the operators is not truly generic. For example, the implementation of the match operator requires two models of the same type as input, and some operators (such as Extract) have specific implementations for each metamodel.

Clio [8] is a tool for creating schema mappings. Whereas schema matching algorithms just discover correspondences between schemas, Clio goes one step further and derives a mapping from a set of correspondences. The mapping is a query that transforms the data from one schema into another schema. However, Clio supports only XML and relational schemas.

More sophisticated model management operators such as Merge (integration of two models according to a given mapping, resulting in a new model) require even more semantic information about the models involved. For example, in [15] a meta model with several association types (e.g. has-a, is-a) is used.

The various approaches to model management show that each operator requires a different view on a model. Schema matching focuses on labels and structure of schema elements, whereas merging and transformation of models require more detailed information about the semantics of a model (e.g. association types, constraints). These different views are supported by our role based approach, as operators will see only those roles which are relevant in their context.

2.2 Scenario

Complex information systems undergo regular changes due to changes of the requirements, of the real world represented by the information system, or of other systems connected to the information system. As an example, we consider the following eBusiness scenario: a supplier of an automotive manufacturer receives orders from a business partner in some XML format. The orders are entered into the ERP system of the supplier by a transformation program, which uses a mapping between the XML schema and the relational DB of the ERP system.

In order to generate this mapping, the two models are represented as models in a generic metamodel (GM1 and GM2). A Match operator can then be used to create a mapping GM1_GM2 between the two models, which can be further translated into a mapping XS1_RM2 between the original models.

Due to a change in the system of the manufacturer, the schema of the orders has changed. Then, this change has to be applied to the mapping between the XML schema and relational DB. Focusing on the models, this scenario can be seen as an example of schema evolution (Fig. 1). The original XML schema XS1 is mapped to the relational

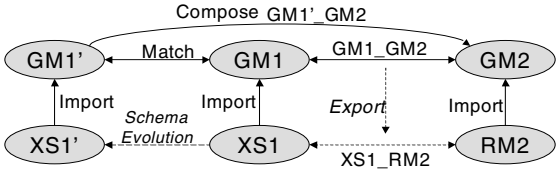


Fig. 1. Schema evolution using *GeRoMe* and Model Management

model (RM2) of the DB using the mapping XS1_RM2. The schema evolution generates a new version of the XML schema, namely XS1'.

Again, instead of applying the model management operators to the level of specific schemas, we will first generate a corresponding representation of the specific model in our GMM (GM1'). Then, we have to apply the Match operator to GM1 and GM1', resulting in a mapping GM1'_GM1 between these models. This match operation should be simpler than matching the new version GM1' with GM2 directly, as two versions of the same model should be quite similar. Then, we can compose the mappings GM1'_GM1 and GM1_GM2 to a new mapping GM1'_GM2. Note, that this operation has just to consider mappings between models represented in *GeRoMe*, which should simplify the implementation of such an operator. The result of this step is a mapping from GM1' to GM2 of those elements which are also present in GM1.

In order to map elements which have been added during the schema evolution a Diff operator has to be used on GM1' and GM1 which takes into account the mapping GM1'_GM1. The difference then has to be mapped individually.

The important difference to other approaches is that the operations in *GeRoMe* are truly generic, they do not have to take into account different representations of models. Therefore, the operators have to be implemented only once, namely for the *GeRoMe* representation.

2.3 Role Based Modeling

The concept of role (or aspect) based modeling has first been described in detail in the context of the network model [2] and later on in several works on object-oriented development and object-oriented databases [6, 16, 17].

Different formalizations have been proposed, which exhibit significant differences, but all have in common that a role or aspect extends the features of an existing object while being a view on the object and *not* an object in its own right. In [6] *multiple direct class membership* is considered as a solution to the problem of artificial intersection classes. That is, instead of defining an intersection class, the combination of state and behavior is achieved by defining an object to be instance of several classes at the same time, which are not necessarily on the same specialization path.

In [16] the notion of aspects of objects is discussed. It is stated that at any given moment an entity may have many different types that are not necessarily related. Often this issue cannot be handled by multiple inheritance since this would lead to a large number of sparsely populated "intersection classes" which add no new state. This approach is different from multiple direct class membership in that each object can have multiple

aspects of the same type, e.g. a person can at the same time be a student at more than one university while still being the same individual.

Other approaches, such as the one considered in [17], treat the different features of an object as roles, which are themselves instances of so called *role classes* and have identity by state. This representation also allows model elements to play directly or implicitly more than one instance of the same role. In addition, [17] introduces the concept of *role player qualification* which means that not every object may play every role but that certain conditions have to hold.

3 The Generic Role Based Metamodel *GeRoMe*

In this section, we will first explain the role model which we have employed to define *GeRoMe*. Based on our analysis of existing metamodels (section 3.2), we have derived the generic role based metamodel, which is described in detail in section 3.3.

3.1 Description of the Role Model

GeRoMe employs the following role model. A model element is represented by an object which has no characteristics in its own right. Roles can be combined to make up a model element encompassing several properties. Therefore, the model element is *decorated* with its features by letting it *play* roles. A role maintains its own identity and may be player of other roles itself. Every model element has to play at least one role and every role object has exactly one player. In our model, some role classes may be used more than once by a model element, e.g. an *Attribute* may play the role of a *Reference* to more than one other *Attribute*. Thus, the complete representation of a model element and its roles forms a tree with the model element as its root.

We used three different relationships between role classes, namely *inheritance*, *play*, and *precondition*. The *play* relationship defines which objects may be player of certain roles. In addition, a role may be a precondition of another role. Thus, in order to be qualified to play a role of a certain class, the player must be already the player of another role of a certain other class. Except for namespaces, all links between model elements are modeled as links between roles played by the elements.

To tap the full power of role modeling, we have to define role classes in such a way that each of them represents an "atomic" property of a model element. Then roles can be combined to yield the most accurate representation of an element.

3.2 Role Based Analysis of Concrete Metamodels

A generic metamodel should be able to represent both the structures and constraints expressible in any metamodel. Thus, to define such a metamodel it is necessary to analyze and compare the elements of a set of metamodels. Our choice of metamodels comprises the relational model (RM) [7] and the enhanced entity relationship model (EERM) [7] because these two models are rather simple and are in widespread use. The metamodel of the Unified Modeling Language (UML, version 1.5) has been analyzed as an example for object-oriented languages. The description logics species of the Web Ontology

Table 1. Roles played by concrete metaclasses

Role	EER	Relational	OWL DL	XML Schema
Domain	domain	domain	xsd datatype	any simple type
Aggregate	entity/rel.-ship type, composed attribute	relation	class	complex type
Association	relationship type	-	a pair of inverse object properties	element
Identified	entity/rel.-ship type	-	class	complex type, schema
BaseElement	base of anon. domain, supertype in isA, comp. type in Union	base of anonymous domain	superclass, superproperty	base simple / complex type
DerivedElement	subtype in isA or union type	anonymous domain constraint	subclass, subproperty	derived simple / complex type
Union	derivation link of union type	-	derivation link of union class	derivation link of union type
IsA	isA derivation link	-	subclassing derivation link	restriction / extension derivation link
Enumeration	enumerated domain restriction	enumerated domain restriction	enumeration	enumeration
Attribute	(composite / multivalued) attribute	column	data type property	attribute, element with simple type
AssociationEnd	link between relationship type and its participator	-	object property	links between two elements one of which is element of the other's complex type
Value	any instance	domain value, tuple	data type value, individual	xsd value, valid XML
Visible	entity type, relationship type, attribute	relation, column	named class, property	named type, attribute element
Reference	-	foreign key	-	keyref
Disjointness	constraint on subtypes	-	constraint on classes	-
Injective	primary/partial key	unique, primary key	inverse functional	unique, key
Identifier	primary/partial key	primary key	-	key
Universal	anonymous domain of attribute	anonymous domain constraint of column	allValuesFrom	restriction of complex type
Existential	-	-	someValuesFrom	-
Default	-	default value	-	default value

Language (OWL DL, `http://www.w3.org/2004/OWL/`) has been included since it follows different description paradigms due to its purpose. For example, properties of concepts are not defined within the concepts themselves but separately. Finally, XML Schema (`http://www.w3.org/XML/Schema`) has been analyzed as it is the most important metamodel for semistructured data.

We analyzed the elements and constraints available in these five metamodels and made out their differences and similarities. In doing so, we identified the role classes, which make up our role based metamodel. In total, we compared about seventy structural properties and elements and twenty types of constraints. Some of them are very easily abstracted, such as data types or aggregates. Others, such as the XML Schema *element* or OWL object properties, are rather intricate and need closer inspection. The XML Schema element is an association (associating a parent element with its children). The root element of a document is a special element which does not have a parent. Furthermore, an XML Schema may allow different types of root elements for a document.

Another problematic example are object properties in OWL DL: the *Association* role is played by a "pair of properties" and the *AssociationEnd* role is played by object properties. Thus, it is difficult to represent such specific model elements in a GMM.

In section 4, we describe some of the representation problems in more detail. Furthermore, some metamodels provide redundant options for representing the same semantics, e.g. there is no semantic difference between an XML Schema attribute and a simple-typed XML Schema element with a maximum cardinality of 1.

Table 1 shows a selection of role classes and states the related model elements in the considered metamodels. The table contains roles which are used to define structural model elements (e.g. relation, class) and roles to define relationships and constraints (e.g. association, disjointness). Due to space constraints, the table does not embody all metamodel elements and correspondences in the different metamodels. Furthermore, we omitted the UML metamodel for the same reason.

3.3 Description of *GeRoMe*

Figure 2 presents the Generic Role Based Model *GeRoMe* at its current state, based on the analysis of the previous section. Below, we will describe the elements of *GeRoMe* according to their basic characteristics: structural elements, derivations, and constraints.

Structural Elements. Every model element representing a primitive data type plays the role of a *Domain*. *GeRoMe* contains a collection of predefined domains such as *int*

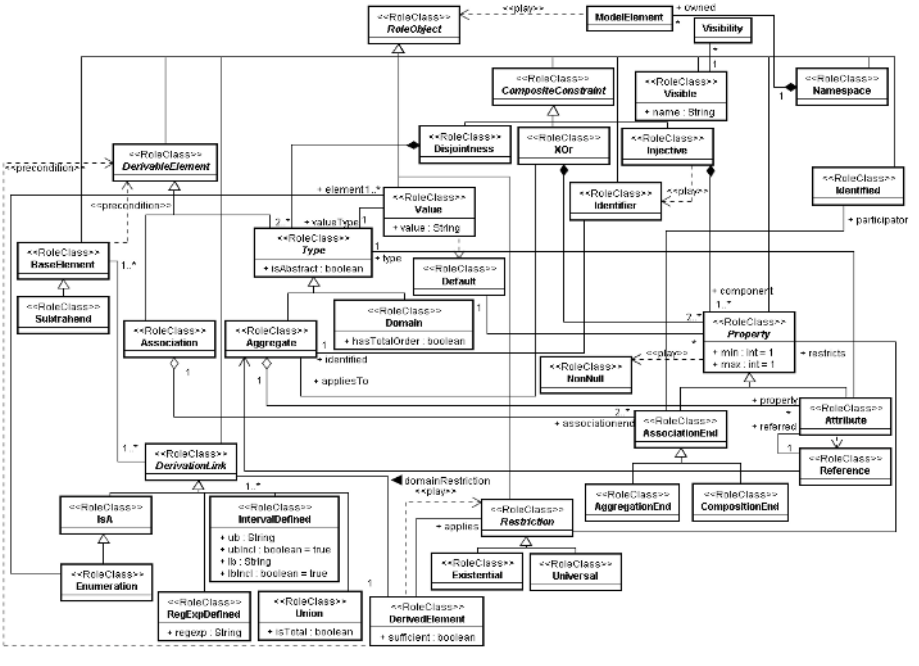


Fig. 2. The Generic Role Based Metamodel (*GeRoMe*)

and *string*. In contrast, model elements which may have attributes play an *Aggregate* role (e.g. entity and relationship types, composite attributes in EER; relations, classes and structs in other metamodels).

Thus, the *Aggregate* role is connected to a set of *Attribute* roles. Each of these *Attribute* roles is part of another tree-structured model element description. An *Attribute* role is a special kind of *Property* and has therefore the *min* and *max* attributes which can be used to define cardinality constraints. Every attribute has a *Type*, which may be a primitive type or an *Aggregate* for composite attributes. Furthermore, an *Attribute* role may itself play the role of a *Reference*, which defines a referential constraint to another *Attribute* of the same type.

The *Aggregate* role and the *Domain* role are specializations of *Type*. *Type* is a specialization of *DerivableElement* which is the abstract class of roles to be played by all model elements which may be specialized. Another kind of *DerivableElement* is the *Association* role. Properties of associations are *AssociationEnd* roles. For example, association roles are played by EER relationship types, UML associations, or UML association classes. A model element which provides object identity to its instances may participate in one or more associations. This is modeled by specifying the element's *Identified* role to be the participator of one or more *AssociationEnd* roles. Thus, an association end is a model element in its own right, and the association is a relationship between objects with identity. In addition, the roles *AggregationEnd* and *CompositionEnd* can be used to model the special types of associations available in UML.

The *Association* and *Aggregate* roles are an intuitive example of two role classes that can be used in combination to represent similar concepts of different metamodels. If the represented schema is in a concrete metamodel which allows relationship types to have attributes, such as the EER metamodel, then every model element playing an *Association* role may play additionally an *Aggregate* role. If associations may not have attributes, which is the case in OWL, a model element may only play either of both roles. On the other hand, the representation of a relational schema may not contain *Association* roles at all. Thus, these two roles can be combined to represent the precise semantics of different metamodel elements. Of course any of these combinations can be further combined with other roles, such as the *Identified* role, to yield even more description choices.

Finally, model elements can be *Visible*, i.e. they can be identified by a name. The *name* attribute of a *Visible* role has to be unique within the *Namespace* it is defined in. A model's root node is represented by a model element which plays a *Namespace* role.

Derivation of New Elements. A *BaseElement* role is played by any model element used in the definition of a derived element. Thus, a *DerivedElement* can have more than one *BaseElement* and vice versa. The type of base element determines the properties of the derived element. A *Subtrahend* is an element whose instances are never instances of the derived element (e.g. a `complementOf` definition in OWL).

BaseElement and *DerivedElement* roles are connected via dedicated model elements representing the *DerivationLink*. Each *DerivationLink* connects one or more *BaseElement*s to one *DerivedElement*. For example, new types can be defined by *Enumeration*, *IsA*, or *Union* definitions. The *IsA* role can be used to define specialization relationships. It extends the definition of a superclass by adding new properties (e.g.

inheritance in UML). A *DerivedElement* role which is connected to an *isA* role with more than one *BaseElement* role can be used to define a type to be the intersection of its base elements.

We identified two different kinds of *isA* relationships which are often not distinguished from each other. All metamodels allow *extension* (i.e. the subtype defines additional attributes and associations) if they allow specialization at all. In EER and OWL, model elements can specialize base elements also by constraining the ranges of inherited properties. In EER, this is called *predicate defined specialization* [7–p.80], whereas in OWL it is called *restriction* and comprises a very important description facility for inheritance. Such derivations can be expressed in our metamodel by connecting a *Universal* or *Existential* role played by the restricted range to the *DerivedElement* role. This *Restriction* role has to define a property, which it constrains.

Furthermore, there are several different ways to define new domains based on existing ones. In XML Schema, named domains can be derived from others whereas in the relational metamodel derived domains occur only as an anonymous type of attributes with enumeration or interval domains.

Constraints. Constraints are represented by separate model elements. For example, a disjointness constraint on a set of derived elements (or any other types) has to be defined by a model element representing this constraint. The element has to play a *Disjointness* role which references the types to be disjoint. In the case of OWL or UML, any collection of classes can be defined to be disjoint; in EER this constraint can be used to define a disjoint *isA* relationship by referencing at least all of the derived elements.

Another constraint is the *Injective* constraint which can be defined on a set of properties. Such an *Injective* role is equivalent to a uniqueness constraint. It can also define a composite key by being connected to multiple properties. An injective constraint playing an *Identifier* role defines a primary key. This reflects the fact that a primary key is only a selected uniqueness constraint, and thereby one of multiple candidate keys.

The *XOr* constraint is a modeling feature that is available in UML metamodels. It can be defined on association ends and states that an object may participate only in one of these associations. Thus, in *GeRoMe* an *XOr* constraint is related to at least two properties. *GeRoMe* can be extended with new role classes representing other features of constraints and structures while existing models and operators still remain correct.

4 Representation Examples

This section presents some example models based on a small airport database in [7–p.109] (see fig. 3). We represented EER, XML Schema and OWL DL models for this example. The model contains simple entity types composed of attributes as well as some advanced features, which are not supported by all metamodels (e.g. composite attributes, *isA* relationship).

4.1 Representation of an EER Schema

Fig. 4 shows a part of the representation of the airport model in *GeRoMe*. For the sake of readability, we refrain here from showing the whole model and omitted repeating

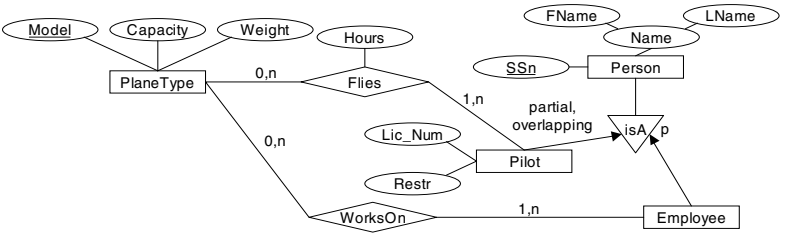


Fig. 3. Part of an airport EER schema

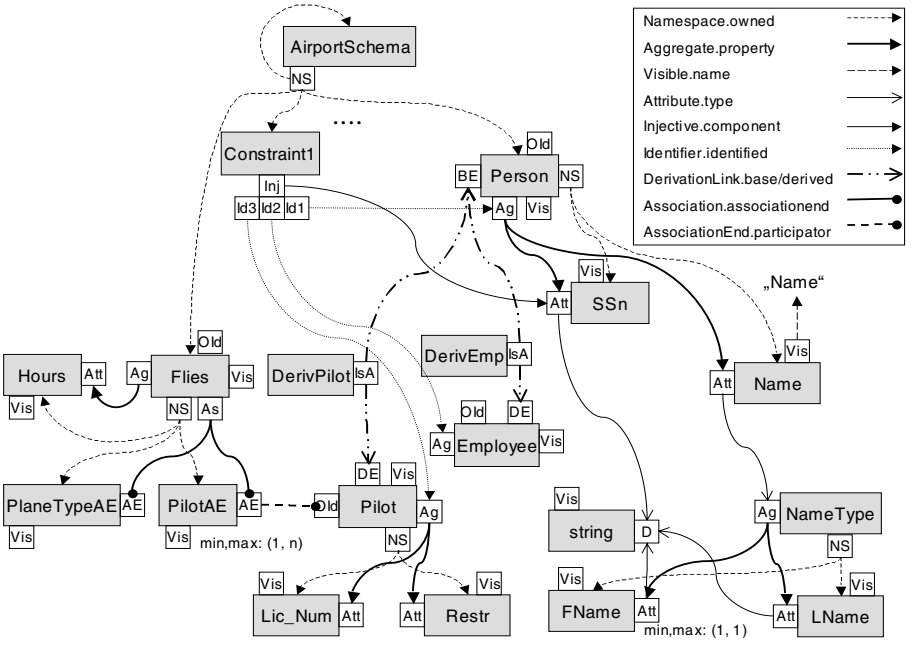


Fig. 4. *GeRoMe* representation of an EER schema

structures with the same semantics such as literal values or links from namespaces to their owned elements. The *GeRoMe* representation shows each model element as a *ModelElement* object (gray rectangle) which plays a number of roles (white squares) directly or by virtue of its roles playing roles themselves. Each such role object may be connected to other roles or literals, respectively. Thus, the roles act as interfaces or views of a model element. The links between role objects connect the model element descriptions according to the semantics of the represented schema.

The root model element of the airport schema is a model element representing the schema itself (*AirportSchema*). It plays a *Namespace* role (NS) referencing all model elements directly contained in this model.

The *Name* attribute is a visible model element and therefore its model element object plays the *Visible* role (Vis). The role defines a name of the element as it could

be seen in a graphical EER editor (note that we omitted the names for other *Visible* roles).

Since entity types are composed of attributes, every object representing an entity type plays an *Aggregate* role (Ag). Furthermore, instances of entity types have object identity. Consequently, representations of entity types also play an *Identified* role (OId). The *Aggregate* role is again connected to the descriptions of the entity type's attributes.

The EER model defines a primary key constraint on the *SSn* attribute. Therefore, a model element representing the constraint (*Constraint1*) and playing an *Injective* role (Inj) is connected to this attribute. This is a uniqueness constraint which is special in the sense that it has been chosen to be a primary key for the entity type *Person*. This fact is represented by the constraint playing an *Identifier* role (Id1) connected to the identified aggregate. Since *Person*'s subtypes must have the same identifier, the injectiveness constraint plays also *Identifier* roles (Id2, Id3) with respect to these model elements.

Specification of domain constraints is usually not possible in the EER model, but the addition of default domains does not hurt. Therefore, attributes always have a type in *GeRoMe*. Domains are themselves represented as model elements playing domain roles (D) (e.g. string). It is also possible to derive new types from existing ones as this is also possible in most concrete metamodels.

In addition, note that the composite attribute *Name* has not a domain but another *Aggregate* as type. Unlike the representation of an entity type, *Name_Type* is not player of an *Identified* role. Consequently, this element cannot be connected to an *AssociationEnd*, which means that it cannot participate in associations. Furthermore, *Name_Type* is not visible as it is an anonymous type. However, the representation is very similar to that of entity types and this eases handling both concepts similarly. For example, in another schema the composite attribute could be modeled by a weak entity type. If these two schemata have to be matched, a generic Match operator would ignore the *Identified* role. The similarity of both elements would nevertheless be recognized as both elements play an *Aggregate* role and have the same attributes.

Furthermore, the figure shows the representation of the *isA* relationship. Since every instance of *Pilot* and *Employee* is also an instance of *Person*, the *Person* model element plays a *BaseElement* role (BE) referenced by two *IsA* roles (IsA). These roles define two children, namely the *DerivedElement* roles (DE) which are played by the respective subtypes *Employee* and *Pilot*. Any attribute attached to the *Aggregate* roles of the subtypes defines an extension to the supertype. The children could also be defined as predicate-defined subtypes by associating to the *DerivedElement* roles a number of *Restriction* roles.

The subtype *Pilot* participates in the relationship type *Flies*. The representation of this relationship contains an *Association* role (As) which is attached to two *AssociationEnd*s (AE) (i.e. a binary relationship). Furthermore, the relationship has an attribute, and consequently, it plays the role of an *Aggregate*. The representations of the two association ends define cardinality constraints and are linked to the *Identified* roles (OId) of their respective participators. They also may play a *Visible* role which assigns a name to the association end.

4.2 Representation of an XML Schema

Fig. 5 shows the XML Schema representation of the example model in *GeRoMe*. The XML Schema element is a relationship between the type defined for the element and the complex type of the nested element. But it is always a 1:n relationship since an XML document is always tree structured. Cross links between elements in different subtrees have to be modeled by references. But what about root elements in a schema? These elements are related to the schema itself which in our role based model is represented by the *AirportSchema* model element. This is just one example of a concrete model element which is not obviously mapped to a generic metamodel.

An XML document conforming to an XML Schema can have any element as root element which is defined in the schema file as a direct son of the `schema` element. Consequently, any such element is represented in *GeRoMe* as a model element playing an association role with its complex type as one participator and the schema node as the other participator. In the example, *Airport* is the only such element. This element is visible and its name is "airport". *AssociationEnd*s of XML elements have no names attached and therefore are anonymous. Complex types may be anonymously nested into an element definition. In the example, this is the case for *AirportType*. Since definitions of keys have labels in XML Schema, the identifier of *Person* plays a *Visible* role with its label "personKey" assigned to it.

Model elements defined within other model elements such as attributes and XML elements are referenced by the *Namespace* role of the containing element. For example,

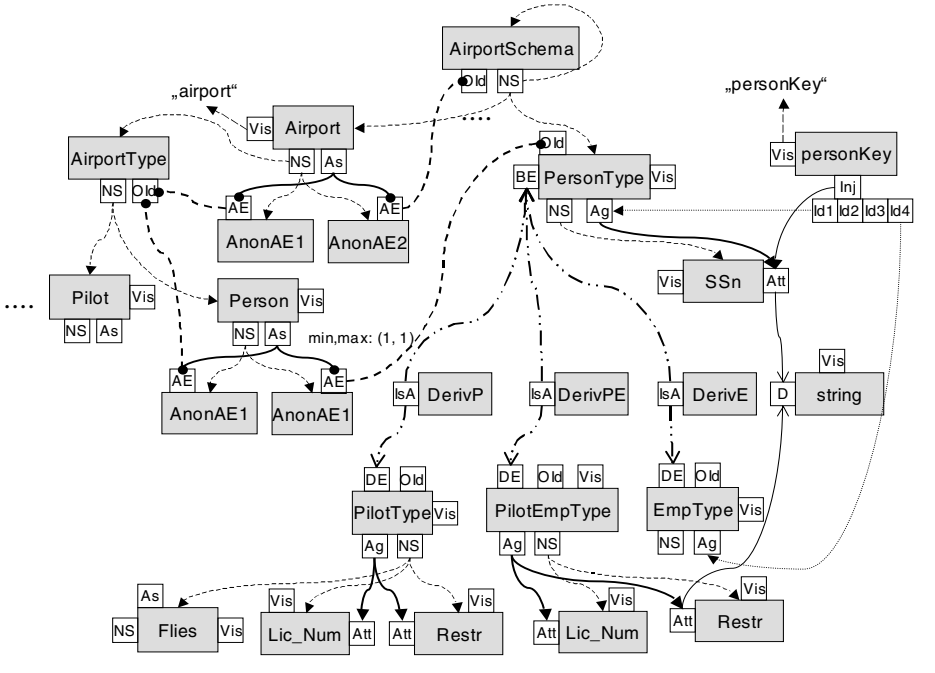


Fig. 5. Representation of a similar XML Schema

the element *Flies* is owned by the *Namespace* role of *PilotType*. Another consequence of the structure of semistructured data is that the *AssociationEnd* of the nested type always has cardinality (1,1), i.e. it has exactly one parent. Finally, the model element *PilotEmpType* has been introduced as it is not possible to represent overlapping types in XML Schema.

4.3 Representation of an OWL DL Ontology

In table 1, we stated that OWL DL object properties are represented by model elements playing *AssociationEnd* roles and that a pair of these model elements is connected by an *Association*. This is another good example for the problems which occur when integrating heterogenous metamodels to a GMM. The reasons for the sketched representation can be explained with the semantics of the relationship type *WorksOn* in fig. 3.

Intuitively and correctly, one represents *WorksOn* as a model element playing an *Association* role. *WorksOn* has two *AssociationEnd*s: one with cardinality (0,n) pointing on *PlaneType* and one with cardinality (1,n) pointing on *Employee*. This is represented analogous to *Flies* in fig. 4. Now what are the problems if you would regard an object property *WorksOn* as corresponding to the given relationship type?

Firstly, an object property always has domain and range. Thus, it has a direction. But the direction of a relationship type is only suggested by its name. On the other hand, an association end has a direction. The role name describes the role which the participator plays in the relationship type w.r.t. the participator at the opposite end. Furthermore, these role names are often phrasal verbs as are the names of object properties in OWL. Actually, in description logics object properties are often called *roles*. Thus, "WorksOn" should be the role name assigned to the link between the relationship type and the entitiy type *PlaneType*.

Secondly, an object property may have one cardinality restriction, whereas a relationship type has at least two (one for each participating entity). This shows that an object property corresponds to an association end, and that a pair of object properties (of which one is the inverse of the other) is a correct representation of a binary association. Note that OWL DL allows only binary relationships.

In order to allow other constraints, such as *Symmetric*, new roles can be added to *GeRoMe*. Adding a new role to the metamodel will render existing models and operator implementations valid and correct. Thus, it is also easy to extend *GeRoMe* if this is necessary in order to include new modeling constructs.

5 Model Management Using *GeRoMe* Models

In this section, we show how model management operators can make use of *GeRoMe*. Transformation of models is a typical task for model management applications. We will explain the transformation of the EER model of fig. 4 into a relational schema. Therefore, the original representation has to undergo several transformations in order to become a representation of a relational schema. Fig. 6 shows the final result of the transformation steps which will be discussed in detail in the following paragraphs.

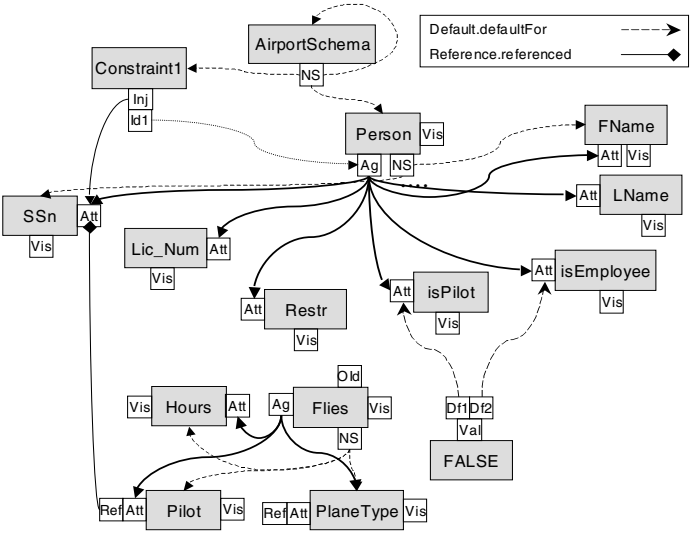


Fig. 6. Representation of the resulting relational schema

In model management, transformation of models is performed by a ModelGen operator, i.e. the operator generates a model from another existing model. We have implemented the transformation of constructs such as composite attributes or inheritance from an EER schema by several ModelGen_X operators. Each operator transforms the modeling constructs not allowed in the relational model into equivalent modeling elements of the relational model. The decomposition of the operators into several "atomic" operators has the advantage that they can be reused in combination with other operators to form new operators. Note that the following operators are not aware about the original representation of the models, i.e. the operators just use the *GeRoMe* representation. Thus, these operators could also be used to transform a UML model into XML Schema if similar transformation tasks are required (e.g. transformation of associations to references).

It has to be emphasized that mapping of models from one metamodel to another is just one popular example application of model management. The goal of our generic metamodel is *not only* to provide a platform for schema translation but to provide a generic model representation that serves as a foundation for the polymorphic usage of *any* model management operator. Thereby, other applications of model management, such as schema evolution, are also supported in a generic way.

Transformation of Relationship Types. Relationship types are not allowed in the relational metamodel. According to properties such as cardinality constraints, they have to be transformed to relations by executing the operator ModelGen_AssocToRef for each *Association* role. First, it looks for attached *AssociationEnd* roles, the arity of the association, and cardinality constraints. Depending on these constraints the transformation is either performed automatically or the user is asked for a decision before the operator can proceed. Copies of all attributes in the participators' identifiers are at-

tached to the relationship's *Aggregate* role. An *Aggregate* role has to be created first, if not yet available. Furthermore, these copies play *Reference* roles (Ref) referencing the original attributes, and thereby defining referential constraints. After performing all these transformations, the association ends and the relationship's *Association* role are deleted.

This yields an intermediate result which cannot be interpreted as a valid schema in the EER or relational metamodel, since it now contains constructs disallowed in both metamodels. An Export operator to the Relational or EER metamodel would have to recognize this invalidity and reject to export.

Transformation of IsA Relationships. The *isA* relationships also have to be removed depending on their characteristics (partial and overlapping), the attributes of the extensions *Pilot* and *Employee* thereby become attributes of the supertype.

The operator *ModelGen_FlattenIsA* fulfills this task by receiving a *BaseElement* role as input. It first checks for disjointness of connected isA relationships and whether they are total or not. Depending on these properties, the user is presented a number of choices on how to flatten the selected isA relationships. In the example, the base type *Person* and its subtypes *Pilot* and *Employee* have been selected to be transformed to one single aggregate due to the fact that the isA relationship is neither total nor disjoint. The resulting aggregate contains all attributes of the supertype and of the subtypes. Additionally, the boolean attributes *isPilot* and *isEmployee* (including *Default* roles Df1 and Df2 related to these attributes) have been introduced.

Transformation of Composite Attributes. The result yet contains a composite attribute and *Identified* roles (OId), which are not allowed in a relational model. The *Identified* roles can be removed directly, as the associations have been transformed to attribute references (earlier by the operator ModelGen_AssocToRef). The transformation of composite attributes is done by another atomic operator. First, it collects recursively all "atomic" attributes of a nested structure. Then, it adds all these attributes to the original *Aggregate* and removes all the structures describing the composite attribute(s) (including the anonymous type). This operator also needs to consider cardinality constraints on attributes, since set-valued attributes have to be transformed into a separate relation.

In this way, the whole EER schema has been transformed to a corresponding relational schema. Of course, more operators are needed to handle other EER features, such as *Union* derivations of new types.

Please note that the differences in the representations stem from the constraints and semantics of the concrete metamodels. Nevertheless the representations use the same role classes in all models, while accurately representing the features of the constructs in the concrete modeling languages. For example, the XML Schema *PersonType* plays the same roles as the EER *Person*, since entity types have the same semantics as XML Schema complex types. Furthermore, the relational *Person* does not play the *Identified* and *BaseElement* roles since these are not allowed in the relational model. On the other hand, all these roles play an *Aggregate* role, and therefore they look the same to an operator which is only interested in this role.

6 Implementation

Our implementation of a model management platform is based on a multi-layered architecture. The lowest layer provides facilities to store and retrieve models in the *GeRoMe* representation and is implemented using the deductive metadatabase system ConceptBase [11]. ConceptBase uses Telos as modeling language [14], which allows to represent multiple abstraction levels and to formulate queries, rules and constraints. Objects are represented using a frame-like or graphical notation on the user side, and a logical representation (triples similar to RDF) based on Datalog$^{\neg}$ internally. Using a logical foundation for the implementation of *GeRoMe* gives the advantage that some of the operators can be implemented in a declarative way. Furthermore, the semantics of the concrete metamodels can also be encoded in logical rules (e.g. inheritance of attributes).

Models represented in *GeRoMe* can also be stored as XMI documents (XML Metadata Interchange) to ease the exchange of metadata. Import and export operators to the native format of the various modeling languages are currently being implemented. The implementation of import and export operators is also a validation of *GeRoMe* since it proves that the modeling constructs from various metamodels can be represented. Before a model can be exported to a concrete metamodel, the export operator has to check whether all roles used can be represented in the target metamodel. If not, the problematic roles have to be transformed into different elements as described, for example, in the previous section. Note that an import to and an export from *GeRoMe* will result in a different model than at the beginning, as there are redundant ways to represent the same modeling construct in specific metamodels. For example, consider a simple typed XML Schema element with a maximum occurrence of one; this could also be modeled as an attribute.

On top of the storage layer, an abstract object model corresponding to the model in fig. 2 has been implemented as a Java library. It uses ConceptBase as storage mechanism and to evaluate rules and queries over a *GeRoMe* representation. The next layer is formed by atomic operators. Operators have to be implemented "as atomically as possible" in order to allow maximum reuse. These atomic operators are not aware of the original metamodel, i.e. their implementations should use only roles and structures in *GeRoMe*.

The operator layer is used by a scripting engine which enables the definition of more complex operators as scripts. A ModelGen_RM operator for transformation of schemata to the relational model could then be defined as a script which reuses operators as described in section 5. This scripting facility is analogous to the scripts which can be defined in the model management application *Rondo* [13].

7 Conclusion

Generic model management requires a generic metamodel to represent models defined in different modeling languages (or metamodels). The definition of a generic metamodel is not straightforward and requires the careful analysis of existing metamodels. In this paper, we have presented the generic role based metamodel *GeRoMe*, which is based on our analysis and comparison of five popular metamodels (Relational, EER, UML, OWL, and XML Schema).

We recognized that the intuitive approach of identifying generic metaclasses and one-to-one correspondences between these metaclasses and the elements of concrete metamodels is not appropriate for generic metamodeling. Although classes of model elements in known metamodels are often similar, they also inhibit significant differences which have to be taken into account. We have shown that role based metamodeling can be utilized to capture both, similarities and differences, in an accurate way while avoiding sparsely populated intersection classes. In addition, the role based approach enables easy extensibility and flexibility as new modeling features can be added easily. Implementations of operators access all roles they need for their functionality but remain agnostic about any other roles. This reduces the complexity of models from an operator's point of view significantly. Furthermore, the detailed representation of *GeRoMe* models is used only by a model management application, users will still use their favorite modeling language.

Whereas role based modeling has yet only been applied to the model level, we have shown that a generic metamodel can benefit from roles. In particular, *GeRoMe* enables *generic* model management. As far as we know, the role based approach to the problem of generic metadata modeling is new.

Using *GeRoMe*, model management operators can be implemented polymorphically, i.e. they just have to be implemented only once using the GMM. The role based approach has been validated by representing several models from different metamodels in *GeRoMe*. We are implementing *automatic* import/export operators in order to verify that the model elements of different metamodels can be represented accurately and completely in *GeRoMe*. Furthermore, we have implemented *GeRoMe* and some ModelGen operators using our metadatabase system ConceptBase.

Future work will concentrate on evaluating and refining *GeRoMe*. The approach has to be further validated by implementing model management operators that make use of the GMM. While it might be necessary to integrate new modeling features of other languages, or features which we did not take into account so far, we are confident that our work is a basis for a generic solution for model management.

Acknowledgements. This work is supported in part by the EU-IST project SEWASIE (`www.sewasie.org`) and the EU Network of Excellence ProLearn (`www.prolearn-project.org`).

References

1. P. Atzeni, R. Torlone. Management of Multiple Models in an Extensible Database Design Tool. *Proc. EDBT'96*, pp. 79–95. Springer, Avignon, 1996.
2. C. W. Bachman, M. Daya. The Role Concept in Data Models. *Proc. VLDB Conf.*, pp. 464–476. Tokyo, 1977.
3. P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. *1st Biennial Conf. on Innovative Data Systems Research (CIDR2003)*. Asilomar, CA, 2003.
4. P. A. Bernstein, A. Y. Halevy, R. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, **29**(4):55–63, 2000.
5. P. A. Bernstein, S. Melnik, M. Petropoulos, C. Quix. Industrial-Strength Schema Matching. *SIGMOD Record*, **33**(4):38–43, 2004.
6. E. Bertino, G. Guerrini. Objects with Multiple Most Specific Classes. *Proc. European Conference on Object-Oriented Programming (ECOOP)*, pp. 102–126. Springer, 1995.

7. R. A. Elmasri, S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Reading, MA, 3rd edn., 1999.
8. M. A. Hernández, R. J. Miller, L. M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. *Proc. ACM SIGMOD Conf.* Santa Barbara, CA, 2001.
9. R. Hull, R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, **19**(3):201–260, 1987.
10. ISO/IEC. Information technology – Information Resource Dictionary System (IRDS) Framework. *Tech. Rep. ISO/IEC 10027:1990*, 1990.
11. M. A. Jeusfeld, M. Jarke, H. W. Nissen, M. Staudt. ConceptBase – Managing Conceptual Models about Information Systems. P. Bernus, K. Mertins, G. Schmidt (eds.), *Handbook on Architectures of Information Systems*, pp. 265–285. Springer, 1998.
12. M. A. Jeusfeld, U. A. Johnen. An Executable Meta Model for Re-Engineering of Database Schemas. *Proc. ER94*, pp. 533–547. Springer, Manchester, 1994.
13. S. Melnik, E. Rahm, P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. *Proc. ACM SIGMOD Conf.*, pp. 193–204. San Diego, 2003.
14. J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, **8**(4):325–362, 1990.
15. R. Pottinger, P. A. Bernstein. Merging Models Based on Given Correspondences. *Proc. VLDB*, pp. 862–873. Berlin, 2003.
16. J. Richardson, P. Schwarz. Aspects: extending objects to support multiple, independent roles. *Proc. ACM SIGMOD Conf.*, pp. 298–307. Denver, 1991.
17. R. K. Wong, H. L. Chau, F. H. Lochovsky. A Data Model and Semantics of Objects with Dynamic Roles. *Proc. ICDE*, pp. 402–411. Birmingham, UK, 1997.

Probabilistic Iterative Duplicate Detection

Patrick Lehti and Peter Fankhauser

Fraunhofer IPSI, Dolivostr. 15, Darmstadt, Germany
{Patrick.Lehti, Peter.Fankhauser}@ipsi.fraunhofer.de

Abstract. The problem of identifying approximately duplicate records between databases is known, among others, as duplicate detection or record linkage. To this end, typically either rules or a weighted aggregation of distances between the individual attributes of potential duplicates is used. However, choosing the appropriate rules, distance functions, weights, and thresholds requires deep understanding of the application domain or a good representative training set for supervised learning approaches. In this paper we present an unsupervised, domain independent approach that starts with a broad alignment of potential duplicates, and analyses the distribution of observed distances among potential duplicates and among non-duplicates to iteratively refine the initial alignment. Evaluations show that this approach supersedes other unsupervised approaches and reaches almost the same accuracy as even fully supervised, domain dependent approaches.

1 Introduction

The goal of data integration is to provide uniform, non-redundant access to a set of typically heterogeneous data sources. If the sources contain overlapping data, not only their schemas need to be integrated, but also their instances, i.e., duplicate instances that refer to the same real world object need to be to detected and merged.

Traditional scenarios for duplicate detection are data warehouses, which are populated by several data sources. Analyses on the data warehouse influences business decisions, therefore a high data quality resulting from the data cleansing process is of high importance. Reaching such a high data quality is currently very costly.

More recently duplicate detection is also arising in ad-hoc integration over the internet, e.g. in P2P, web service or grid settings, where datasets and services are virtually integrated for temporary applications. In such scenarios no long and expensive data cleansing process can be carried out, but good duplicate estimations must be available directly.

Several communities have addressed the problem of duplicate detection. The database community came up with knowledge-intensive approaches that deploy domain dependent rules [1] or keys [2], possibly combined with distance functions [3–5] to take into account errors in the instances. Designing such domain dependent rules, keys and distance functions requires significant manual effort and a good understanding of the instances and their errors.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1225–1242, 2005.

The AI community has focused on learning such rules and distance functions supervised from a set of example duplicates [6–9].

More recently, object identification methods have been adopted to detect duplicate objects in graph data models. In this context the knowledge of aligned related objects can be taken into account for duplicate detection [10–13].

The statistics community has conducted a long line of research in probabilistic duplicate detection, largely based on the seminal theory for record linkage of Fellegi and Sunter [14].

This paper describes a refinement of the Fellegi-Sunter model and presents an efficient method of unsupervised learning of the required parameters in combination with continuous distance measures and taking attribute dependencies into account. The approach starts with the set of potential duplicates identified in a preprocessing phase and iteratively determines their matching probability by analyzing the distribution of observed distances among the potential duplicates and among the non-duplicates.

In more detail, the contributions of this paper are the following:

- a new probabilistic model for duplicate detection, which is related to the Fellegi-Sunter model
- the handling of continuous distance measures in the Fellegi-Sunter and our model, as opposed to thresholded boolean values
- an iterative algorithm for determining the required parameters for the Fellegi-Sunter and our model
- the consideration of attribute dependencies in the required parameters for the Fellegi-Sunter and our model
- an evaluation of our approach on two test data sets

The remainder of this paper is organized as follows. Section 2 defines the general duplicate detection problem and shortly introduces into the Fellegi-Sunter model. Section 3 presents our approach to duplicate detection. Section 4 presents the results of the evaluation of our approach compared to other approaches. Section 5 summarizes related work and Section 6 concludes.

2 Detecting Duplicates

2.1 Preliminaries

The problem of detecting duplicates can be defined as, given two lists of records A and B divide the set of all record-pairs $(a, b) \in A \times B$ into a set of matching record-pairs M and unmatching record-pairs U. A record is basically a vector of attributes (or fields), thus, a record-pair is a vector of attribute-pairs.

In general the overall duplicate detection process consists of several phases. The first phase, often called blocking, tries to efficiently find a candidate set of duplicates, then a second phase, sometimes called matching, is doing the actual duplicate decision.

The matching phase in general involves an in-depth comparison of the candidate duplicate pairs, i.e. comparing the individual attribute-pairs, resulting in

what is known as comparison vector $\gamma[a, b]$ for every candidate pair. The individual components of the comparison vector (γ_i) represent the comparison results between the individual attribute-pairs. These individual comparison results can be boolean (attribute matches or does not match), discrete (e.g., matches, possibly matches or does not match) or continuous (e.g., attribute values have a similarity of 0.2).

The task for duplicate detection is now to classify such a given comparison vector γ into the sets M or U. Therefore the components of the comparison vector must be combined to an overall result. It can be easily seen that the individual attributes do not have the same relevance for this overall result. E.g. a matching "person name" is a much stronger indication for a duplicate record than a matching "year of birth", because of the higher *uniqueness* of "person name" in contrast to "year of birth". However, a non-matching "year of birth" is a strong indication for a non-duplicate, may be even higher as a slightly different "person name", under the assumption of a high *reliability* for "year of birth".

Some existing approaches either ignore the different relevance of individual attributes and concatenate all attributes [10, 4] or leave the user with the task to define this relevance, e.g. in [2] this is done by a manual and knowledge-intensive declaration of logic rules. In our evaluation one baseline experiment uses an unweighted mean of the results of individual attribute comparisons, which ignores the different relevance of individual attributes.

2.2 The Fellegi-Sunter Model for Record Linkage

Based on the initial ideas and problem description of Newcombe [15], Fellegi-Sunter [14] defined a theory for record linkage including relevance for individual attributes. To this end they define the following probabilities on γ:

$$\begin{aligned} m(\gamma) &= P(\gamma|(a, b) \in M) \\ u(\gamma) &= P(\gamma|(a, b) \in U) \end{aligned} \tag{1}$$

Here $m(\gamma)$ is the conditional probability of γ, given that a and b are elements of M and $u(\gamma)$ is the conditional probability of γ, given that a and b are elements of U. They have shown that the ratio $w(\gamma) = m(\gamma)/u(\gamma)$ can then be used to decide for a duplicate, non-duplicate or potential duplicate. To this end they define how to set the appropriate thresholds on $w(\gamma)$ given acceptable error rates for false misses and false matches.

In order to determine the parameters $u(\gamma)$ and $m(\gamma)$ often a conditional independence assumption between the individual components of the comparison vector is made. Under this assumption the parameters $u(\gamma)$ and $m(\gamma)$ can be computed using m_i and u_i for the individual probabilities for the comparison vector component γ_i:

$$\begin{aligned} m(\gamma) &= m_1(\gamma_1) * m_2(\gamma_2)...m_k(\gamma_k) \\ u(\gamma) &= u_1(\gamma_1) * u_2(\gamma_2)...u_k(\gamma_k) \end{aligned} \tag{2}$$

This simplification only produces good results if the conditional independence assumption really holds, which is often not the case for real data. E.g. in the

address domain the attributes street and city are not independent, as a particular street is only present in one or a few cities.

Fellegi-Sunter propose two methods for calculating $m(\gamma)$ using the conditional independence assumption, one relies on additional knowledge of error rates in the values, the second is limited on comparison vectors of size 3.

Winkler [16] showed that the EM algorithm can be used for unsupervised learning of the $m(\gamma)$ and $u(\gamma)$ parameters under the independence assumption. Using the independence assumption is the second baseline experiment for our approach.

Several work e.g. from Winkler [17] and Larsen and Rubin [18] tried to determine the $m(\gamma)$ and $u(\gamma)$ values in cases where the conditional independence assumption does not hold. All these approaches try to explicitly model the dependencies, which only works for boolean variables.

2.3 Using Continuous Distance Measures

The use of edit distance functions for attribute-pair comparison have shown to be very effective to detect matching errorneous values. However, the result of such a distance function is a continuous distance measure, which makes it difficult to determine a corresponding probability $m_i(\gamma_i)$ and $u_i(\gamma_i)$ and it precludes the normal multinomial probability model for a Bayesian network for taking dependencies into account.

There are two basic approaches to handle continuous measures. One is to discretize the values e.g. into binary values ("match", "does not match") using a threshold [19]. This also enables to model dependencies with a Bayesian network, but it has the problem that discretization into a small number of values leads to a poor approximation of the continuous distribution. The other method is to use specific parametric families such as the Gaussian distribution, with the same problem of loss of accuracy.

Cohen et al. [20] use an unsupervised learning approach based on the Fellegi-Sunter model that takes attribute dependencies into account and use continuous distance measures. They use a hierarchical latent variable graphical model to model dependencies between the continuous valued individual attributes. Such an hierarchical graphical model (HGM) show better results than the two basic models.

The HGM approach serves as the third baseline experiment we compare our method to.

3 Probabilistic Theory for Duplicate Detection

The ratio $w(\gamma)$ in the Fellegi-Sunter model is difficult to interpret and requires the calibration of a threshold. A slightly different view on it using probability theory makes the interpretation much easier. If we can calculate the conditional probability of a and b being duplicates (element of M), given the comparison vector γ, then a threshold on this probability can guide our duplicate decision,

e.g. if the probability of a and b being a duplicate is greater than 50% then they are declared as duplicates. This conditional probability can be calculated as follows:

$$P((a,b) \in M \mid \gamma) = \frac{m(\gamma) * P(M)}{m(\gamma) * P(M) + u(\gamma) * P(U)} \tag{3}$$

This formula follows directly from the Bayes rule and the total probability theorem:

$$P((a,b) \in M \mid \gamma) = \frac{m(\gamma) * P(M)}{P(\gamma)} \tag{4}$$

$$P(\gamma) = m(\gamma) * P(M) + u(\gamma) * P(U) \tag{5}$$

$u(\gamma)$ and $m(\gamma)$ are defined in the same way as in Fellegi-Sunter. The probability $P(M)$ is the prior probability that two records are duplicates and is defined as the ratio between the set of duplicates and the set of all pairs:

$$P(M) = \frac{|M|}{|M| + |U|} \tag{6}$$

$P(U)$ is simply the complement of $P(M)$; $P(U) = 1 - P(M)$. It can be easily seen that $P(M)$ is always very small and $P(U)$ always nearly 1. In particular for duplicate free individual data sets (A and B), P(M) is between 0 (no overlap) and $\frac{min(|A|,|B|)}{|A|*|B|} = \frac{1}{max(|A|,|B|)}$ (one data set is a subset of the other).

The main differences to the original Fellegi-Sunter ratio is that the result is a probability and therefore the value range is {0,1}, whereas in Fellegi-Sunter the value range of $w(\gamma)$ is {0,infinite}. This allows to use a fixed threshold for duplicate decision, instead of the problem to find an appropriate threshold for every duplicate detection application. Additionally our ratio introduces the term $P(M)$. If there are only few duplicates expected, $P(M)$ will be very small and accordingly the matching probability for potential duplicates will be small as well. However, it can be easily shown that the resulting order of the records with formula (3) is identical to the order of the Fellegi-Sunter ratio.

3.1 Determination of $u(\gamma)$

For the determination of the $u(\gamma)$ parameter a set of non-duplicates is needed, which is representative for U. Under the assumption that the individual data sets are duplicate free, all pairs within these individual data sets can be used as such a set further called U'.

Our evaluations showed that U' is really representative for U and this method therefore a valid approximation. This method comparing all pairs in the individual data sets is not practical for large data sets, in this case the complexity must be reduced by sampling (see also Section 3.5).

Without any independence assumptions between the attributes, the probability $u(\gamma)$ is then the ratio between the number of all pairs (a,b) in U' whose

comparison vector $\gamma[a,b]$ is equal to γ (equal for boolean valued comparison vector components) to the size of U'.

$$u(\gamma) = \frac{|\{(a,b) \in U' \mid \forall \gamma_i[a,b] = \gamma_i\}|}{|U'|} \tag{7}$$

For the calculation of this ratio, the problem can be alternatively described as query issue. Therefore we assume that U' is a relation that contains the comparison vectors for every pair as tuples. These queries can be efficiently answered using standard query optimization techniques, e.g. indices on the individual attributes. Reducing the size of U' by sampling, further increases the query performance.

3.2 Iterative Determination of $m(\gamma)$ and $P(M)$

For the determination of the $m(\gamma)$ parameter a set of duplicates, which is representative for M is also required. As we found no way to identify a representative subset of M, we use the set of potential duplicates (call it M'_0) from the blocking phase as a superset of M for a first determination of $m(\gamma)$ ($m_0(\gamma)$) and $P(M)$ ($P(M'_0)$). Without any independence assumptions between the attributes, the probability $m_0(\gamma)$ is then the ratio between the number of all pairs (a,b) in M'_0 whose comparison vector $\gamma[a,b]$ is equal to γ (equal for boolean valued comparison vector components) to the size of M'_0. $P(M'_0)$ is the ratio between the size of M'_0 to the number of all possible pairs.

$$m_0(\gamma) = \frac{|\{(a,b) \in M'_0 \mid \forall \gamma_i[a,b] = \gamma_i\}|}{|M'_0|} \tag{8}$$

$$P(M'_0) = \frac{|M'_0|}{|A| * |B|} \tag{9}$$

This approach will result in an overestimation of $m(\gamma)$ and $P(M)$, as they are determined on a set, which still contains non-duplicates. With these values we then calculate the duplicate probability $P_0((a,b) \in M \mid \gamma)$, which will also be higher as the true value, because of the larger $m(\gamma)$ value. This means all pairs that are ranked very low (below a threshold Θ) after this first estimation, will be even lower ranked with the true values, which allows us to easily declare such pairs as definite non-duplicates.

Removing these non-duplicates from M'_0 results in a new and better potential duplicate set M'_1. On this set we can now recalculate the values for $m(\gamma)$ and $P(M)$ based on M'_1 and repeat the process. This can be done iteratively until no more definite non-duplicates are detected.

$$M'_{t+1} = \{(a,b) \in M'_t \mid P_t((a,b) \in M \mid \gamma) > \Theta\} \tag{10}$$

$$P_t((a,b) \in M \mid \gamma) = \frac{m_t(\gamma) * P(M'_t)}{m_t(\gamma) * P(M'_t) + u(\gamma) * P(U)} \tag{11}$$

This iterative process relies on a few assumptions. Definite non-duplicates can only be detected if the probability for a non-duplicate ($u(\gamma)$) is significantly higher than the probability for a duplicate ($m(\gamma)$). This must obviously hold for the true values, as otherwise also humans would not be able to distinguish between duplicates and non-duplicates, but it does not necessarily hold for the estimated values from M'_0. Therefore the ratio of actual duplicates to non-duplicates in M'_0 must be sufficiently high, which demands for a low false match rate in the blocking phase, like the method introduced in [21].

3.3 Using Continuous Random Variables

Existing approaches for the Fellegi-Sunter model either use boolean values for the γ_i variables ("match" or "does not match") or they use discrete values, like "does not match" or "matches with a particular value". However, these approaches ignore the fact that values are not always clearly categorizable as definite match or not match, but there are often a few clear matches, a few clear not matches and a whole range of similarity. The degree of similarity can be defined with distance functions such as edit-distance. Our approach can be applied to arbitrary distance functions.

When using continuous distance measures in our model, we define the $u(\gamma)$ probability to be the ratio between the number of all pairs in U' whose comparison vector is absolutely smaller than γ to the number of all pairs in U'. This relies on the assumption that the probability for a non-duplicate is monotonically decreasing with the increase of the similarity.

$$u(\gamma) = \frac{|\{(a,b) \in U' \mid \forall \gamma_i[a,b] <= \gamma_i\}|}{|U'|} \tag{12}$$

The probability $m(\gamma)$ is defined accordingly to be the ratio between the number of all pairs in M'_t whose comparison vector is absolutely greater than γ to the number of all pairs in M'_t. This relies on the assumption that the probability for a duplicate is monotonically decreasing with the decrease of the similarity.

$$m_t(\gamma) = \frac{|\{(a,b) \in M'_t \mid \forall \gamma_i[a,b] >= \gamma_i\}|}{|M'_t|} \tag{13}$$

The calculation of these probabilities is equivalent to the method described in Section 3.1 and Section 3.2 by replacing the "=" operator with a "<=" and ">=" operator respectively.

3.4 Handling Null Values

Datasets often contain optional attributes, i.e. attributes may contain null values. As such null values cannot be used in distance functions, it is also difficult to define a distance value for the comparison vector, when null values are involved. Therefore we extend the definition of a comparison vector component to be either a continuous distance value, or one of the following three discrete values:

"a=null", "b=null", "a=b=null". For the calculation of the $m(\gamma)$ and $u(\gamma)$ we assume that null values only match other null values, i.e. a null value never matches a continuous distance value.

3.5 Sampling

For larger data sets the calculation of the full U' is not practical as its size is quadratic with the size of the individual data sets. In this case sampling methods must be used to find a subset U'_s. However, sampling only approximates the distribution of γ, which reduces the accuracy of $u_s(\gamma)$ from the sample in opposite to the true $u(\gamma)$. Here especially two cases that happen more often for smaller samples, have an enormous negative impact on $u_s(\gamma)$:

- $u_s(\gamma) = 0$: In a small sample it may happen that a rarely occurring γ can not be observed. This results in $u_s(\gamma)$ to be 0 and $P((a,b) \in M \mid \gamma)$ to be 1 independent of the value of $m(\gamma)$.
- $u_s(\gamma) > u(\gamma)$: It may also happen that a particular γ is observed more often than usual within the sample. This results in an overestimation of $u_s(\gamma)$ and may then also result in an underestimation of $P_t((a,b) \in M \mid \gamma)$. The iteration described in Section 3.2 relies on the assumption that $P_t((a,b) \in M \mid \gamma)$ is always an overestimation and fails if this assumption does not hold.

We address the $u_s(\gamma) = 0$ problem by approximation, i.e. observing this case we try to find a good approximation for $u_s(\gamma)$, which is slightly higher than 0. The approximation that shows the best results is simply the fallback to the independence assumption. We also experimented with other candidates like multi-dimensional linear interpolation and statistic error bounds, but these performed worse and are computationally more costly.

For the $u_s(\gamma) > u(\gamma)$ problem, we need to find a lower bound of $u_s(\gamma)$ that is guaranteed to be smaller than $u(\gamma)$. Here we use statistic error bounds on the value of $u_s(\gamma)$. Therefore every occurrence of a specific γ in U'_s is seen as the observation of a specific random event. The ratio $\pi = x/n$ between the number of all occurrences of such an event (x) to the number of all observations (n) corresponds to the probability $u_s(\gamma)$. The lower bound π_l and the upper bound π_u for the ratio π are given in statistics literature [22] via the following formulas:

$$\pi_l = \frac{x}{x + (n - x + 1) * F_l} \tag{14}$$

$$\pi_u = \frac{(x+1) * F_u}{n - x + (x+1) * F_u} \tag{15}$$

with:

$$F_{l\{df_1=2*(n-x+1),df_2=2*x\}}(\lambda/2)$$
$$F_{u\{df_1=2*(x+1),df_2=2*(n-x)\}}(\lambda/2)$$

Where F denotes the F-distribution, df_1 and df_2 denote the two degrees of freedom for the F-distribution and λ is the probability of error for these bounds, e.g. for a 95% confidence interval λ is 5%.

We use this lower bound on $u_s(\gamma)$ during iteration to guarantee the overestimation of $P_t((a,b) \in M \mid \gamma)$. However, for the final matching estimation of a pair always the truly observed $u_s(\gamma)$ is used.

As even the full U' is only a sample on the set of all possible non-duplicates, these two fallback strategies can also be used without explicit sampling.

4 Evaluation

4.1 Datasets

For an evaluation we have chosen a *Restaurant* and a *Census* data set, which were previously used as benchmarks for duplicate detection, e.g. in [6, 20]. The restaurant dataset contains 864 restaurant names and addresses with 112 duplicates, composed of 533 and 331 restaurants assembled from Fodor's and Zagat's restaurant guides. These individual datasets are duplicate free. The attributes being restaurant name, street address, city and cuisine. Table 1 shows a sample duplicate record from this dataset.

The census data set is a synthetic dataset containing 824 census-like records with 327 duplicates, composed of two duplicate free sets with 449 and 375 records. The attributes being last name, first name, house number and street. Table 2 shows a sample duplicate record from this data set.

Table 1. Sample duplicate records from the *Restaurant* data set

name	address	city	cuisine
uncle nick's	747 ninth ave.	new york city	greek
uncle nick's	747 9th ave. between 50th and 51st sts.	new york	mediterranean

Table 2. Sample duplicate records from the *Census* data set

last name	first name	house number	street
JIMENCZ	WILLPAMINA	S 214	BANK
JIMENEZ	WILHEMENIA	214	BANKS

4.2 Experimental Methodology

The data sets were transformed into an XML format and stored into an XML database. Indices on every single attribute were created inside the database, which corresponds to a sorting on every attribute.

For the blocking phase the multi-pass algorithm as described in [21] is used, which is an optimization of the classical sorted-neighborhood method of Hernandez and Stolfo [2]. It is used with a window distance size of 0.25. On the restaurant dataset the blocking was done on the name and address attribute resulting in 251 potential duplicate pairs with 100% recall and 45% precision, on the census dataset the blocking was done on the last name and first name

attribute resulting in 1524 potential duplicate pairs with 90% recall and 19% precision. These potential duplicates are used as the initial set M'_0 for evaluating our approach.

For comparison of the experimental results precision, recall and F-measures are calculated. These are defined as usual in information retrieval [23]:

$$Precision = \frac{|CorrectlyIdentifiedDuplicates|}{|IdentifiedDuplicates|}$$

$$Recall = \frac{|CorrectlyIdentifiedDuplicates|}{|TrueDuplicates|}$$

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

The precision-recall curves in the figures use interpolated precision values at 20 standard recall levels following the traditional procedure in information retrieval [23]. However, the figures show only the interesting area between the recall levels 0.6 and 1.

All experiments using the bayes formula use a precalculated set of non-duplicates for the determination of $u(\gamma)$. This set either contains all pairs of the individual datasets, which are 196393 pairs for the restaurant and 170701 pairs for the census examples, or a randomly chosen sample of these full sets. These pairs are compared, stored and indexed in advance as described in Section 3.1.

The iteration as described in Section 3.2 uses a threshold of 50%, i.e. all pairs that have a matching probability of less than 50% are removed.

4.3 Comparison of the Approach

We compare our approach (further referred to as the Bayes classifier) on both datasets to a number of baseline experiments:

- Mean: this baseline simply takes the arithmetic unweighted mean of the individual distances as overall result - ignoring attribute relevance.
- Independence: this uses the Bayes formula, but calculating the $m(\gamma)$ and $u(\gamma)$ values under the independence assumption - ignoring attribute dependencies.
- HGM (Hierarchical Graphical Model): this is the unsupervised approach presented in [20] that uses the same datasets for their evaluation. Therefore we simply copied their results for comparison, although they used a different blocking algorithm.
- SVM Jaccard (supervised): this is the best of the fully supervised approaches presented in [6] that uses the same restaurant dataset for their evaluation. We copied their results for the restaurant dataset.

We use the Jaro-distance [3] as distance function in our experiments, HGM uses the SoftTFIDF-distance, SVM uses the Jaccard-distance. A comparison of

these distance functions is presented in [5]. The impact of different distance functions is shortly evaluated in Section 4.4.

The maximum F-measures of these methods on both data sets are shown in table 3. Figure 1 shows the precision and recall curves for the restaurant dataset, Figure 2 the curves for the census dataset.

Table 3. Maximum F-measures for detecting duplicates

Method	Restaurant	Census
HGM (SoftTFIDF)	0.844	0.759
Mean	0.874	0.853
Bayes independent	0.951	0.575
Bayes dependent	0.955	0.853
SVM (supervised)	0.971	-

These results show that the Bayes dependent classifier is always one of the best classifiers, although the Bayes independent classifier for the restaurant and the simple mean classifier for the census dataset perform nearly as good. This can be explained by the observation that the attributes in the restaurant dataset are only little dependent from each other, but have varying relevance, whereas the attributes in the census dataset have higher dependencies, but similar relevance.

The maximum F-measure with our approach is not far away from the best results in [6] (0.971), with the difference that our approach uses unsupervised instead of supervised learning techniques. When using the true U set with our

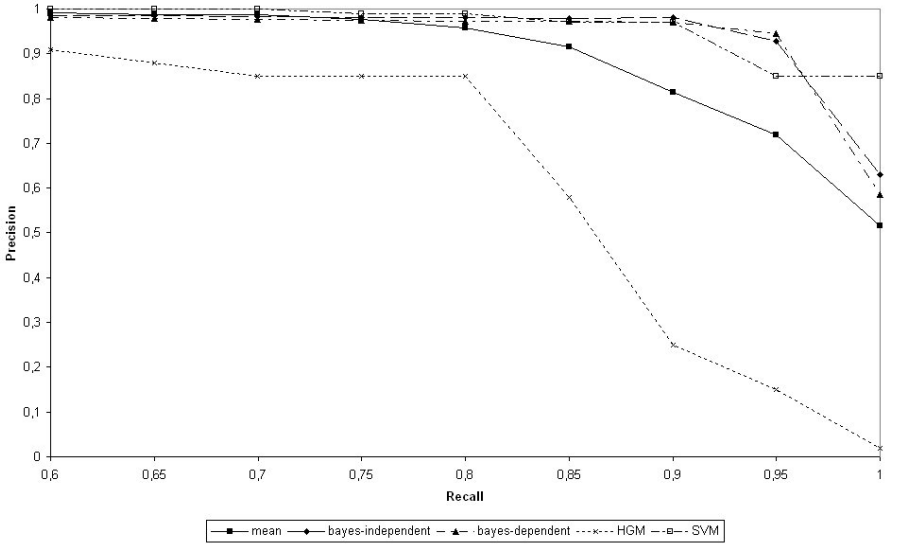


Fig. 1. Precision-Recall for the restaurant dataset

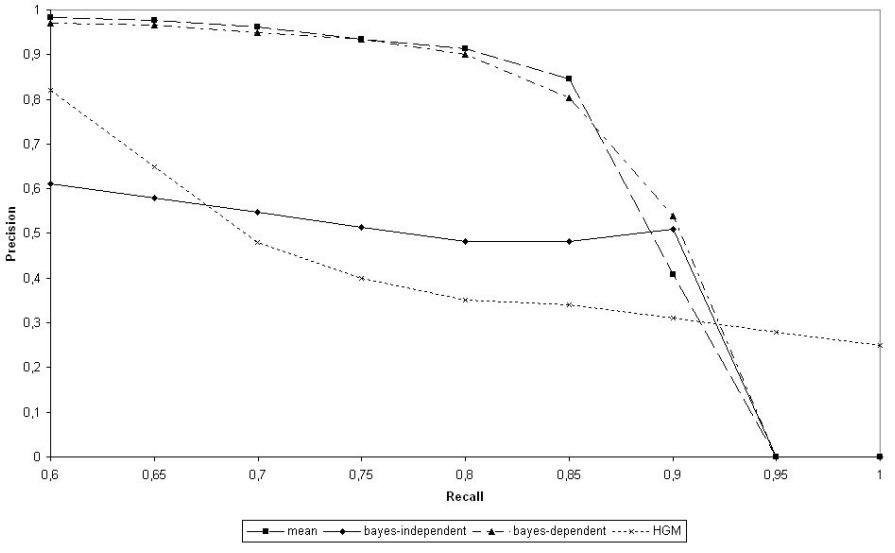


Fig. 2. Precision-Recall for the census dataset

approach, which can be seen as supervised learning, we even reach a maximum F-measure of 0.978.

The higher precision of the HGM method for the census dataset at the 100% recall level is caused by the different blocking algorithm. Ours reaches only around 90% recall, therefore the precision for the 95% and 100% recall levels are 0.

4.4 Impact of Distance Functions

The previous experiments used the Jaro distance for comparison. In order to verify that our approach is working with arbitrary distance functions, we also carried out the experiments with a simple Levenshtein distance [24]. The maximum F-measures can be found in Table 4.

The experiment shows that the selection of a good distance function matters, but still our approach performs better than the baseline experiments. A possible

Table 4. Maximum F-measures using Levenshtein distance

Method	Restaurant	Census
HGM (SoftTFIDF)	0.844	0.759
Mean	0.885	0.778
Bayes independent	0.925	0.565
Bayes dependent	0.922	0.806
SVM (supervised)	0.971	-

interpretation of this impact is that good distance functions are able to more sharply separate the distributions of U and M and give therefore more accurately overall results. This suggest further work in unsupervised finding ideal distance functions.

4.5 Sampling

One disadvantage of our approach is the required set of non-duplicates for determination of $u(\gamma)$. The previous experiments always used the full set of all pairs of the individual datasets. We also experiment with samples of this set, containing 50% or just 10% of the pairs. The small samples show the problem of $u_s(\gamma) = 0$ or $u_s(\gamma) > u(\gamma)$, which are addressed by the approximation fallback and error bounds as described in Section 3.5. The experiment with a 50% sample was done on two randomly chosen seeds, the experiments with a 10% sample was done on three randomly chosen seeds. The maximum F-measures for these experiments can be found in Table 5.

Table 5. Maximum F-measures for sampling

Sampling Method	Restaurant	Census
Full:		
Bayes dependent	0.955 (100%)	0.853 (100%)
50%-1:		
Bayes dependent	0.943 (-1.3%)	0.894 (+4.8%)
+approximation	0.946 (-0.9%)	0.894 (+4.8%)
+error bounds	0.939 (-1.7%)	0.892 (+4.6%)
+both	0.946 (-0.9%)	0.892 (+4.6%)
50%-2:		
Bayes dependent	0.924 (-3.3%)	0.867 (+1.6%)
+approximation	0.932 (-2.4%)	0.867 (+1.6%)
+error bounds	0.929 (-2.7%)	0.877 (+2.8%)
+both	0.936 (-2.0%)	0.877 (+2.8%)
10%-1:		
Bayes dependent	0.912 (-4.5%)	0.743 (-12.9%)
+approximation	0.940 (-1.6%)	0.865 (+1.4%)
+error bounds	0.917 (-4.0%)	0.743 (-12.9%)
+both	0.939 (-1.7%)	0.848 (-0.6%)
10%-2:		
Bayes dependent	0.871 (-8.8%)	0.337 (-60.5%)
+approximation	0.942 (-1.4%)	0.337 (-60.5%)
+error bounds	0.871 (-8.8%)	0.844 (-1.1%)
+both	0.934 (-2.2%)	0.844 (-1.1%)
10%-3:		
Bayes dependent	0.932 (-2.4%)	0.722 (-15.4%)
+approximation	0.960 (+0.5%)	0.722 (-15.4%)
+error bounds	0.932 (-2.4%)	0.850 (-0.4%)
+both	0.960 (+0.5%)	0.850 (-0.4%)

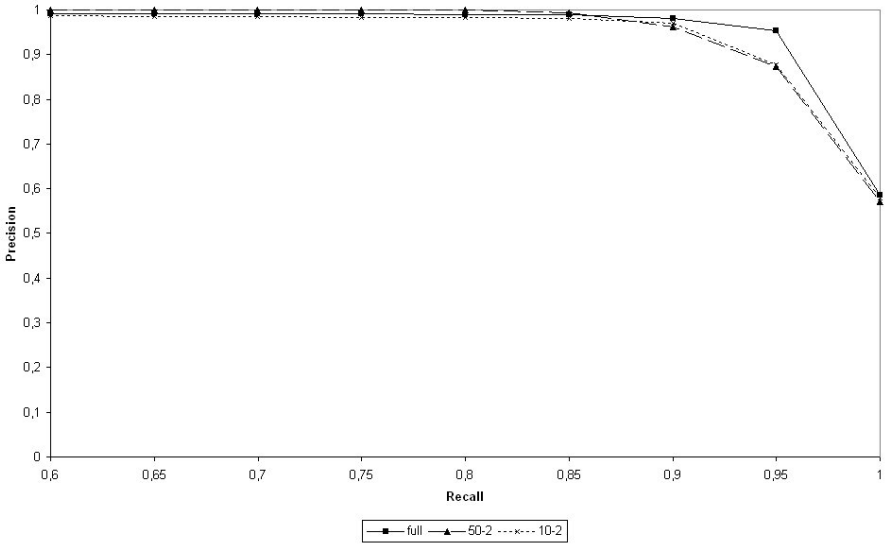


Fig. 3. Precision-Recall for sampling on the restaurant dataset

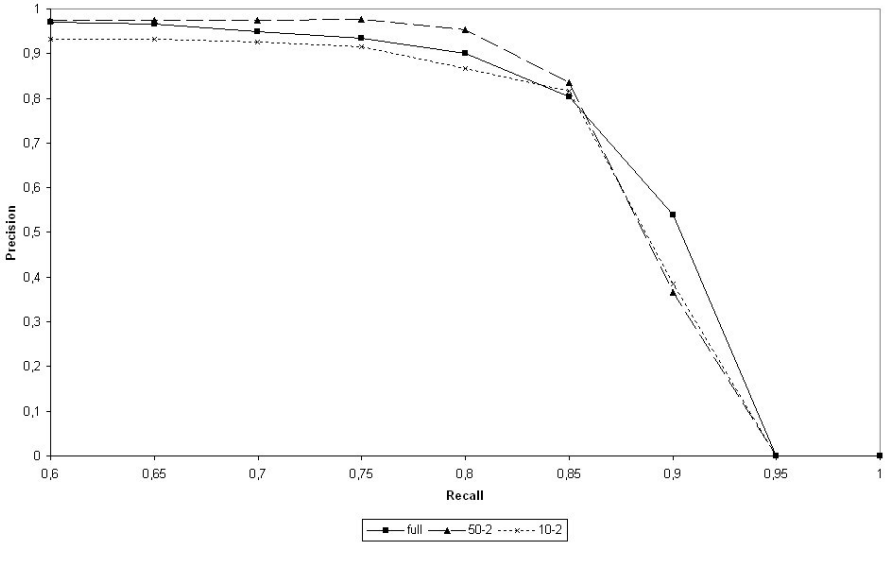


Fig. 4. Precision-Recall for sampling on the census dataset

Figure 3 and Figure 4 show the precision and recall curves for the full set in comparison to the 50% and the 10% sample with the worst F-measures, i.e. for both datasets the 50%-2 and 10%-2 sample.

These results show that even for a 10% sample the minimum F-measure is not more than 2.2%/1.1% less than for the result using the full set, the maximum F-measure might even be higher by accident. It further shows the necessity of both fallback strategies especially for small samples, where often one of it significantly increases the accuracy, thereby the effectiveness of either of them depends on the sample set. Conditions for finding a minimal sample set of U' that still results in a high accuracy is future work.

5 Related Work

In the database community a few knowledge-intensive approaches have been developed. Hernandez and Stolfo [2] present a two phase data cleansing approach based on key expressions for blocking and logic rules for the second phase. The selection of appropriate keys as well as the logic rules are both very knowledge-intensive tasks. Galhardas et al. [1] present a framework for data cleansing based on a set of rules. This framework is very flexible, but requires intensive human interaction and knowledge in order to provide good rules. Monge and Elkan [4] compare instances based on a concatenation of all attributes with the help of distance functions and present a blocking algorithm. A concatenation of all attributes only produces reasonable results if all data sources are complete, i.e. have no null values for attributes and all attributes show the same relevance for duplicate detection.

Fellegi and Sunter [14] from the statistics community present a formal mathematical model for record linkage already in 1969. Their pioneering results are still valid today and our approach is largely based on or related to their results. An introduction to their model is found in Section 2.2. Several work from Winkler and others summarized in [19] show how to use the EM algorithm for unsupervised learning of the Fellegi-Sunter parameters under the independence assumption and generalized for taking dependencies into account, but discretized to boolean random variables ("match", "does not match") or categorical values ("match", "possibly matches", "does not match").

Elfeky et al. [25] claim that probabilistic record linkage models always have the disadvantage to handle only boolean or categorical values and require a training set, which we showed to be wrong. Therefore they propose to use machine learning techniques either based on supervised training of a classifier (e.g. some kind of decision model) or using unsupervised clustering methods like k-means. However, building an appropriate training set is a manual and knowledge-intensive task and simple clustering methods like k-means are only able to identify clusters that are linear separable, which is in general not the case for real world data.

The AI community has proposed various other approaches using supervised learning. Cohen and Richman [8] learn to combine multiple similarity metrics to identify duplicates. Bilenko and Mooney [6] learn in a first step distance metrics for individual fields, and in a second step they learn a combining metric for similarity between records using Support Vector Machines (SVM). Their

results are compared to ours in section 4. Sarawagi and Bhamidipaty [9] present an approach of active learning that reduces the user-overhead for selecting an appropriate learning set.

Ravikumar and Cohen [20] present an unsupervised learning approach based on the Fellegi-Sunter model that takes attribute dependencies into account and use continuous distance measures. We compare our results to them in Section 4. They use a hierarchical latent variable graphical model to model dependencies between the continuous valued individual attributes. They showed that their approach helps reducing overfitting of dependency models. However, overfitting is not an issue in our approach as we always work on the full joint distribution.

A few more recent approaches try to take similarities between related objects in a graph data model into account, which is orthogonal to our approach. Ananthakrishna et al. [10] also compare instances based on the concatenation of all attributes with the same disadvantages as in [4]. Additionally they use a co-occurrence similarity. This co-occurrence similarity checks for co-occurrences in the children sets of the entities. Domingos [13] take similarities between related objects into account by setting up a network with a node for each record pair. This allows to propagate duplicate detection results to related objects. Bhattacharya and Getoor [12] introduce new distance measures that take entity relationships into account. They showed that this can be used for the duplicate detection task. Pasula et al. [11] introduce a generative probability model for the related problem of identity uncertainty based on probabilistic relational networks.

6 Conclusion and Future Work

This paper presented a new probabilistic model for duplicate detection and an iterative approach for determining appropriate parameter values using continuous distance measures and taking attribute dependencies into account. The proposed approach is based on unsupervised learning and domain independent, i.e. it is completely free from user interaction, which makes classical approaches very expensive. The evaluations on two test datasets showed that the approach reaches very high accuracy, outperforms existing unsupervised approaches and is nearly competitive with fully supervised and domain dependent approaches.

The problem of finding a minimal representative sample set for U', might be solved by using a blocking algorithm against U, which would directly select only representative non-duplicates instead of the here presented random selection.

It also showed that the importance of the used distance function is high, therefore we want to further investigate in unsupervised finding ideal distance functions.

An important future work is to extend the approach to object identification, i.e. detecting duplicate objects in a graph data model as opposed to flat record lists. This involves extending the considered 1:1 relationships to 1:n (multi-valued attributes) and n:m relationships. As there exists several related and recent work [10, 11, 12, 13], we hope to be able to incorporate such existing methods into our approach. Another future work is the evaluation of the approach on a large real world dataset.

Acknowledgments

This work is supported by the bmb+f in the SemIPort project.

References

1. Galhardas, H., Florescu, D., Shasha, D., Simon, E.: An extensible framework for data cleaning. In: Proceddings of the 16th International Conference on Data Engineering (ICDE '00). (2000) 312
2. Hernandez, M.A., Stolfo, S.J.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery **2** (1998) 9–37
3. Jaro, M.: Advances in record linkage methodology as applied to matching the 1985 census of tampa. Journal of the American Statistical Society **84** (1989) 414–420
4. Monge, A., Elkan, C.: An efficient domain independent algorithm for detecting approximately duplicate database records. In: In Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery. (1997)
5. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string metrics for matching names and records. In: Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington, DC (2003) 13–18
6. Bilenko, M., Mooney, R.J.: Learning to combine trained distance metrics for duplicate detection in databases. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX (2002)
7. Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 522–532
8. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002), Edmonton, Alberta (2002)
9. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002), Edmonton, Alberta (2002)
10. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: Proceedings of the 28th International Conference on Very Large Data Bases(VLDB '02). (2002)
11. Pasula, H., Marthi, B., Milch, B., Russell, S., Shpitser, I.: Identity uncertainty and citation matching. In: Advances in Neural Information Processing Systems 15, MIT Press (2003)
12. Bhattacharya, I., Getoor, L.: Deduplication and group detection using links. In: Proceedings of the KDD-2004 Workshop on Link Analysis and Group Detection. (2004)
13. Domingos, P., Domingos, P.: Multi-relational record linkage. In: Proceedings of the KDD-2004 Workshop on Multi-Relational Data Mining. (2004) 31–48
14. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association **64** (1969) 1183–1210
15. Newcombe, H.B., Kennedy, J.M., Axford, S.J., James, A.P.: Automatic linkage of vital records. Science **130** (1959) 954–959
16. Winkler, W.E.: Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. In: Proceedings of the Section on Survey Research Methods, American Statistical Association. (1988) 667–671

17. Winkler, W.E.: Improved decision rules in the fellegi-sunter model of record linkage. In: Proceedings of the Section on Survey Research Methods, American Statistical Association. (1993) 274–279
18. Larsen, M.D., Rubin, D.B.: Alternative automated record linkage using mixture models. Journal of the American Statistical Association **79** (2001) 32–41
19. Winkler, W.E.: The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC (1999)
20. Ravikumar, P., Cohen, W.W.: A hierarchical graphical model for record linkage. In: AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence, AUAI Press (2004) 454–461
21. Lehti, P., Fankhauser, P.: A precise blocking method for record linkage. In: Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'05). (2005)
22. Sachs, L. In: Angewandte Statistik. Springer, Berlin (2004) 434–435
23. Baeza-Yates, R., Ribiero-Neto, B. In: Modern Information Retrieval. Addison Wesley (1999) 74–79
24. Levenshtein, V.I.: Binary codes capable of correcting insertions and reversals. Soviet Physics Doklady **10** (1966) 707–710
25. Elfeky, M.G., Verykios, V.S., Elmargarid, A.K.: Tailor: A record linkage toolbox. In: Proceedings of the 18th International Conference on Data Engineering (ICDE'02). (2002)

Efficient Processing of XPath Queries with Structured Overlay Networks*

Gleb Skobeltsyn, Manfred Hauswirth, and Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{gleb.skobeltsyn, manfred.hauswirth, karl.aberer}@epfl.ch

Abstract. Non-trivial search predicates beyond mere equality are at the current focus of P2P research. Structured queries, as an important type of non-trivial search, have been studied extensively mainly for unstructured P2P systems so far. As unstructured P2P systems do not use indexing, structured queries are very easy to implement since they can be treated equally to any other type of query. However, this comes at the expense of very high bandwidth consumption and limitations in terms of guarantees and expressiveness that can be provided. Structured P2P systems are an efficient alternative as they typically offer logarithmic search complexity in the number of peers. Though the use of a distributed index (typically a distributed hash table) makes the implementation of structured queries more efficient, it also introduces considerable complexity, and thus only a few approaches exist so far. In this paper we present a first solution for efficiently supporting structured queries, more specifically, XPath queries, in structured P2P systems. For the moment we focus on supporting queries with descendant axes ("//") and wildcards ("*") and do not address joins. The results presented in this paper provide foundational basic functionalities to be used by higher-level query engines for more efficient, complex query support.

1 Introduction

P2P systems have been very successful as global-scale file-sharing systems. Typically these systems support simple exact and substring queries which suffice in this application domain. To make P2P systems a viable architectural alternative for more technical and database-oriented applications, support for more powerful and expressive queries is required, though. A couple of approaches have been suggested already on top of unstructured P2P systems and are being applied successfully in practice, for example, Edutella [21]. Unstructured P2P systems do not use indexing, but typically some form of constrained flooding, and thus structured queries are very easy to implement, since each peer receiving the query, which can be arbitrarily complex, can locally evaluate it and return its contribution to the overall result set. However, this comes at the expense of very

* The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project BRICKS No 507457.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1243–1260, 2005.

high bandwidth consumption and some intrinsic limitations. For example, completeness of results cannot be guaranteed, query planning is not possible, and joins are nearly impossible to implement efficiently in large-scale settings.

The efficient alternative are structured P2P systems, as they typically offer logarithmic search complexity in the number of participating nodes. Though the use of a distributed index (typically a distributed hash table) makes the implementation of structured queries more efficient, it also introduces considerable complexity in an environment that is as instable and error-prone as large-scale P2P systems. Thus, so far only a few approaches exist, for example the PIER project [14].

In this paper we present a first solution for the efficient support of structured queries, more specifically, XPath queries, in large-scale structured P2P systems. We assume such a P2P system processing queries expressed in a complex XML query language such as XQuery. XQuery uses XPath expressions to locate data fragments by navigating structure trees of XML documents stored in the network. We refer to this functionality as processing of *structured queries*. In this paper we provide an efficient solution for processing XPath queries in structured P2P networks. We do not address query plans or joins, but focus on a foundational indexing strategy that facilitates efficient answering of structured queries, which we refer to as *structural indexing* in the following. We restrict the supported queries to a subset of the XPath language including node tests, the child axes ("/"), the descendant axes ("//") and wildcards ("*") which we will denote as $XPath_{\{*,//\}}$ in the following. Thus, in this paper we describe an indexing strategy for efficient $XPath_{\{*,//\}}$ query answering in a structured P2P network. Our goal was to provide a basic functional building block which can be exploited by a higher-level query engine to efficiently answer structural parts of complex queries in large-scale structured P2P systems. However, we think that the work presented in this paper provides generally applicable concepts which can be generalized to more complete support of XPath predicates and joins.

The paper is organized as follows: Section 2 gives a brief introduction to our P-Grid structured overlay network which we use to evaluate our approach. Our basic indexing strategy is described in Section 3 whose efficiency is then improved through caching as described in Section 4. The complete approach is then evaluated in Section 5 through simulations. Following that, we position our approach in respect to related work in Section 6 and present our conclusions in Section 7.

2 The P-Grid Overlay Network

We use the P-Grid overlay network [1, 3] to evaluate the approach presented in this paper. P-Grid is a structured overlay network based on the so-called distributed hash table (DHT) approach. In DHTs peer identifications and resource keys are hashed into one key space. By this mapping responsibilities for partitions of the key space can be assigned to peers, i.e., which peer is responsible for answering queries for what partition. To ensure that each partition of the

key space is reachable from any peer, each peer maintains a routing table. The routing table of a peer is constructed such that it holds peers with exponentially increasing distance in the key space from its own position in the key space. This technique basically builds a small-world graph [16], which enables search in $O(\log N)$ steps. Basically all systems referred to as DHTs are based on variants of this approach and only differ in respect to fixed (e.g., P-Grid, Pastry [25]) vs. variable key space partitioning (e.g., Chord [27]), the topology of the key space (ring, interval, torus, etc.), and how routing information is maintained (redundant entries, dealing with network dynamics and failures, etc.).

Without constraining general applicability we use binary keys in P-Grid. This is not a fundamental limitation as a generalization of the P-Grid system to k-ary structures is natural, and exists. P-Grid peers refer to a common underlying binary trie structure in order to organize their routing tables as opposed to other topologies, such as rings (Chord), multi-dimensional spaces (CAN [24]), or hypercubes (HyperCuP). Tries are a generalization of trees. A trie is a tree for storing strings in which there is one node for every common prefix. The strings are stored in extra leaf nodes. In the following we will use the terms trie and tree conterminously.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$, also called the *key space partition*. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $key(d)$ is calculated using an order-preserving hash function, i.e., $\forall s_1, s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$, which is pre-requisite for efficient range querying as information is being clustered. $key(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain references $\sigma(p)$ to peers with the same path, i.e., their replicas, and use epidemic algorithms to maintain replica consistency. Figure 1 shows a simple example of a P-Grid tree. Note that, while the network uses a tree/trie abstraction, the system is in fact hierarchy-less, and all peers reside at the leaf nodes. This avoids hot-spots and single-points-of-failures.

P-Grid supports a set of basic operations: *Retrieve(key)* for searching a certain key and retrieving the associated data item, *Insert(key, value)* for storing new data items, *Update(key, value)* for updating a data item, and *Delete(key)* for deleting a data item. Since P-Grid uses a binary tree, *Retrieve(key)* is of

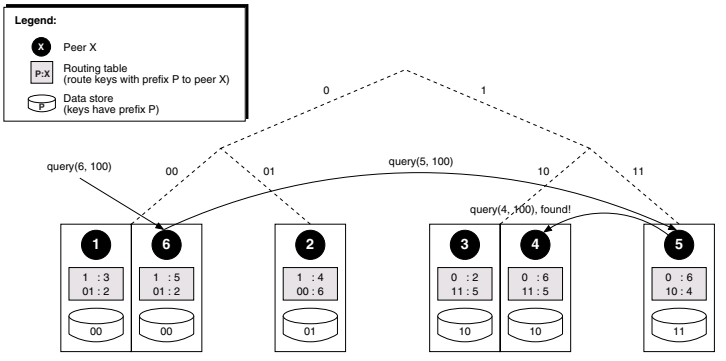


Fig. 1. P-Grid overlay network

complexity $O(\log |\Pi|)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, in [2] it is shown that due to the randomized choice of routing references from the complimentary subtree, the expected search cost remains logarithmic ($0.5 \log N$), independently of how the P-Grid is structured. The intuition why this works is that in search operations keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid's order-preserving hashing may lead to non-uniform key distributions.

The basic search algorithm is shown in Algorithm 1.

Algorithm 1. Search in P-Grid: $Retrieve(key, p)$

```
1: if π(p) ⊆ key or π(p) ⊃ key then
2:    return(d ∈ δ(p)|key(d) = key);
3: else
4:    determine l such that π(key, l) = π(p, l)‾;
5:    r = randomly selected element from ρ(p, l);
6:    Retrieve(key, r);
7: end if
```

p in the algorithm denotes the peer that currently processes the request. The algorithm always terminates successfully, if the P-Grid is complete (ensured by the construction algorithm) and at least one peer in each partition is reachable (ensured through redundant routing table entries and replication). Due to the definition of ρ and $Retrieve(key, p)$ it will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $Retrieve(key, p)$ the length of the common prefix of $\pi(p)$ and key increases at least by one and therefore the algorithm always terminates.

Insert(key, value) and *Delete(key)* are based on P-Grid's more general update functionality [10], *Update(key, value)*, which provides probabilistic guarantees for consistency and is efficient even in highly unreliable, replicated environments, i.e., $O(\log|\Pi| + replication\ factor)$. An insert operation is executed in two logical phases: First an arbitrary peer responsible for the key-space to which the key belongs is located (*Retrieve(key)*) and then the found peer notifies its replicas about the inserted *key* using a light-weight hybrid push-and-pull gossiping mechanism. Deleting and updating a data item works alike.

3 Basic Index

The goal of structural indexing is to provide efficient means to find a peer or a set of peers, that store pointers to XML documents or fragments containing the path(s) matching the queried expression. As we target large-scale distributed XML repositories, we try to minimize the messaging costs, measured in overlay hops, required to answer the query. The intuition of our approach is to use standard database techniques for suffix indexing applied to XML path expressions. Instead of symbols, the set of XML element tags is used as the alphabet.

Given an XML path P consisting of m element tags, $P = l_1/l_2/l_3/\ldots/l_m$, we store m data items in the P-Grid network using the following subpaths (suffixes) as application keys:

- $sp_1 = l_1/l_2/\ldots/l_m$
- $sp_2 = l_2/\ldots/l_m$
  $\vdots$
- $sp_m = l_m$

The key of each data item is generated using P-Grid's prefix-preserving hash function: $key_i = h(sp_i)$. The insertion of the m data items requires $O(m \log N)$ overlay hops. Each data item stores the original XML path to enable local processing and a URI to the XML source document/fragment. We refer to this index as *basic index* in the following.

For example, for the path "*store/book/title*", the following data items (we represent them in a form of {key,data} pairs) will be created:

- $\{h(\text{“}store/book/title\text{”}), (\text{“}store/book/title/\text{”}, URI)\}$
- $\{h(\text{“}book/title\text{”}), (\text{“}store/book/title/\text{”}, URI)\}$
- $\{h(\text{“}title\text{”}), (\text{“}store/book/title/\text{”}, URI)\}$

Any peer in the overlay network can submit an $XPath_{\{*,//\}}$ query. To support wildcards ("*") we consider them as a particular case of descendant axes ("//"). They are converted into "//" and are used only at the local lookup stage as a filtering condition. I.e., our strategy is to preprocess a query replacing all "*" by "//", for example, "$A/*/B$" → "$A//B$", answer the transformed query using our distributed index and filter the result set by applying the original

query to it, thus we obtain the intended semantics of "*". In this paper we concentrate on general indexing strategy and do not address possible optimizations on this issue.

Let q_B denote the longest sequence of element tags divided by child axes ("/") only, which we will call the *longest subpath of a query* in the following. For example, for the query "$A//C/D//F$", $q_B =$ "C/D".

When a query is submitted to a peer, the peer generates a query message that contains the path expression and the address of the originating peer and starts the basic structural querying algorithm as shown in Algorithm 2.

Algorithm 2. Querying using basic index: $AnswerQuery(query, p)$

```
1: compute q_B of query;
2: key = h(q_B)
3: if π(p) ⊆ key then
4:    return(d ∈ δ(p) | isAnswer(d, query) = true);
5: else if π(p) ⊃ key then
6:    ShowerBroadcast(query, length(key), p);
7: else
8:    determine l such that π(key, l) = π(p, l)‾;
9:    r = randomly selected element from ρ(p, l);
10:   AnswerQuery(query, r);
11: end if
```

The function $AnswerQuery(query, p)$ extends $Retrieve(key, p)$ described in Algorithm 1 for answering the $XPath_{\{*,//\}}$ $query$ using the basic index. First the search key is computed by hashing the query's longest subpath q_B. Then we check whether the currently processing peer is the only one responsible for key. If yes, the routing is finished and the result set is returned (line 4). Function $isAnswer(d, query)$ examines if the data item d is a correct answer for $query$. Alternatively, if routing is finished at one of the peers from the sub-trie defined by key (line 5) [1], all peers from this sub-trie could store relevant data items and have to be queried. To do this, we use a variant of the broadcasting algorithm (line 6) for answering range queries described in [11] as shown in Algorithm 3, where the range is defined by key prefix. I.e., we query all peers for which $key \subset \pi(p)$.

The algorithm starts at an arbitrary peer from the sub-trie, and the query is forwarded to the other partitions in the trie using this peer's routing table. The process is recursive, and since the query is split in multiple queries which appear to trickle down to all the key-space partitions in the range, we call it the *shower algorithm*.

[1] I.e., the key is a proper substring of the peer's path ($\pi(p) \supset key$), which means that all bits of the key have been resolved and the query has reached a sub-trie, in which several peers may store data belonging to the query's answer set, and all have to be checked for possible answers (this is ensured by P-Grid's clustering property).

Algorithm 3. $ShowerBroadcast(query, l_{current}, p)$

```
1: for l = l_current to length(π(p)) do
2:    r = randomly selected element from ρ(p, l);
3:    ShowerBroadcast(query, l + 1, r);
4: end for
5: return(d ∈ δ(p) | isAnswer(d, query) = true);
```

With basic indexing the expected cost (in terms of messages) of answering a single query is $O(L) + O(S) - 1$, where L is the cost of locating any peer in the sub-trie and S is the shower algorithm's messaging cost. The expected value of L is a length of the sub-trie's prefix. The intuition for this value is that it is analogous to the search cost in a tree-structured overlay of size 2^L. The expected value of L is $N/2^L$, which refers to the number of peers in the sub-trie. The latency remains $O(\log N)$ because the shower algorithm works in a parallel fashion.

To illustrate how a query is answered using the basic index, assume the $query$ = "$A//C/D//E$" is submitted at some peer p. Following Algorithm 2 the peer responsible for h("C/D") is located. Assume there is a sub-trie defined by the prefix h("C/D") as it is depicted in Figure 2. The shower broadcast is executed and every peer in the sub-trie performs a local lookup for $query$ and sends the result to the originating peer p.

4 Caching Strategy

The basic index is efficient in finding all documents matching an $XPath_{\{*,//\}}$ query expression based on the longest sequence of element tags (q_B). It performs well with queries containing a relatively long $h(q_B)$, such that the number and the size of shower broadcasts is not excessive. However, the search cost might be substantially higher for queries, which require large broadcasts, i.e., $h(q_B)$ is short. For example, queries like "A//B" are answered by looking up the peer responsible for h("A") and then a relatively expensive broadcast depending on the data in the overlay may have to follow. The search would be more efficient if knowledge about the second element tag "B" would be employed as well. In this section we introduce a caching strategy to address this issue, which allows us to reduce the number of broadcasts, and thus, decrease the average cost of answering a query.

Each peer which receives a query determines if it belongs to one of the following types:

1. Queries that can be answered locally, i.e., $\pi(p) \subseteq h(q_B)$. For example the path "A/B/C//E" at the peer responsible for h("A/B").
2. Queries that require additional broadcasts, i.e., $\pi(p) \supset h(q_B)$, but contain only one subpath, $query = q_B$. For example, the path "A" at the peer responsible for h("A/B"). In this case matching index items are stored on

all the peers responsible for $h(\text{“}A\text{”})$. As queries of this type may be very expensive, for example “//”, they could be disabled in the configuration or only return part of the overall answer set to constrain costs.

3. Queries that require an additional broadcast, $\pi(p) \supset h(q_B)$, but include at least one descendant axis (“//”) or wildcard (“*”), i.e. $q_B \neq q$. For example the query “A//C//E” at the peer responsible for $h(\text{“}A/C\text{”})$. The result set for such queries can be cached locally and accessed later without performing a shower broadcast.

Type 1 queries are inexpensive and thus work well with basic indexing. *Type 2 queries* are so general that they return undesirably large result sets and the system may want to block or constrain them. The most relevant type of queries whose costs should be minimized are thus *type 3 queries* which we will address in the following. For simplifying the presentation we assume that only one peer is responsible for a given query and have resources to cache results. We can assume that storage space is relatively cheap as the “expensive” resource in overlay networks is network bandwidth. However, each peer is entitled to arbitrarily limit the size of its cache at will.

4.1 Answering a Query

As a first step Algorithm 2 is modified by changing the routing and adding cache handling. If we sort the subpaths of an $XPath_{\{*,//\}}$ query by their length in descending order, we can “rewrite” the original query as $q_C = concat(P_{l_1}, P_{l_2}, ..., P_{l_k})$, where P_{l_i} is the i-st longest subpath. We will use q_C for routing purposes instead of q_B, which gives us the benefit that we use the whole query for generating the routing key. The modified querying algorithm is shown in Algorithm 4.

Algorithm 4. Querying using basic index extended with cache: $AnswerQueryWithCache(query, p)$

```
1: compute q_C of the query;
2: key_C = h(q_C)
3: compute q_B of the query;
4: key_B = h(q_B)
5: if π(p) ⊆ key_B then
6:    return(d ∈ δ(p) | isAnswer(d, query) = true);
7: else if (π(p) ⊃ key_B) and (ifCached(query) = false) then
8:    ShowerBroadcast(query, length(key_B), p);
9: else if π(p) ⊆ key_C then
10:   return(d ∈ cache(p) | isAnswer(d, query) = true);
11: else
12:   determine l such that π(key_C, l) = π(p, l)‾;
13:   r = randomly selected element from ρ(p, l);
14:   AnswerQueryWithCache(query, r);
15: end if
```

In line 1 we compute q_C which is used for routing (line 12) to the peer (probably) storing a cached result set. Since P-Grid uses a prefix-preserving hash function and $q_B \subseteq q_C$ (q_B is always the first subpath of q_C), this peer is located in the $key_B = h(q_B)$ sub-trie.

Similarly to the basic index's search algorithm we check whether the currently processing peer is the only one responsible for key_B (line 5). If yes, the result set is returned (line 6). If the routing reached one of the peers from the sub-trie defined by key_B, we execute the shower broadcast (line 8) to answer the query as introduced in the previous section, but only if the query has not already been cached (line 7). Section 4.2 explains how the function $ifCached(query)$ works. If the query is cached, the routing proceeds until the peer responsible for key_C is reached. This peer answers the query by looking up a cached result set (line 10).

4.2 Cache Maintenance

Each peer runs a cache manager, which is responsible for cache maintenance. Two functions $createCache(query)$ and $deleteCache(query)$ are available, where $query$ is any query the peer is responsible for. In the following we explain how these functions work. How the cache manager decides if a query is worth caching or not will be described in 4.3.

To cache a query a peer determines a sub-trie's prefix by hashing q_B and collects a result set for the query by executing a special version of the shower broadcast algorithm. The only difference with regard to the $ShowerBroadcast$ listed in Algorithm 3 is that for cache consistency reasons all the peers in the broadcast sub-trie add the query expression to their *lists of cached queries* L_{CQ}. Thus, in case the P-Grid is updated, i.e. data items are inserted, modified or deleted, any peer from the sub-trie can contact the peer(s) that cache relevant queries, to inform them of the change so they can keep their cache consistent. This operation needs $O(\log N)$ messages per cache entry. The function $ifCached(query)$ (line 7, Algorithm 4) looks up the (locally maintained) L_{CQ} list to determine if the query is cached. This solution requires additional storage space which can be significantly decreased by the use of Bloom filters. Similarly, the cache deletion operation requires updates of all L_{CQ} lists.

When a data item is inserted, updated or deleted, all relevant cache entries are updated respectively. A peer looks up the cached queries list and sends update messages to all the peers caching the relevant queries. Each cache update requires a message to be routed with an expected cost of $0.5 \log N$. If we denote as $C(path)$ the number of cached queries that have $path$ as an answer, the update cost can be estimated as $O(\log N) + O(C(path) * 0.5 \log N)$.

4.3 What to Cache?

The cache manager analyzes the benefits of caching for each candidate query the peer is responsible for. To do so, it estimates the overall messaging cost for the query with and without caching. The decision to cache the query result or to delete the existing cache entries is based on comparing these two values.

If the query is cached, each search operation for that query saves a shower broadcast (the shower broadcast requires $s-1$ messages where s is the number of peers in the trie). On the other hand each update operation for any data item related to the query will cost additional $O(\log N)$ messages to update the cache. Knowing the approximate ratio of search/update operations (obtained by local monitoring) the peer can make an adaptive decision on caching of a particular query.

The query is considered to be profitable to cache if:

$$UpdateCost * UpdateRate(subtrie) < SearchCost(subtrie) * SearchRate(query)$$

where

- *subtrie* is the prefix of the q_B sub-trie, i.e., the basic index's shower broadcast sub-trie;
- *UpdateCost* is the cost of one update, which is equal to the routing cost, i.e., $O(\log N)$;
- $UpdateRate(subtrie)$ is the average update rate in the given sub-trie;
- $SearchCost(subtrie)$ is the number of peers in the sub-trie to be contacted to answer the shower broadcast; and
- $SearchRate(query)$ is the search rate for the given query.

To estimate these values each peer collects statistics. For *SearchRate* the peer's local knowledge is sufficient, whereas *UpdateCost* and *SearchCost* values have to be gathered from the neighbors. To do so, we can periodically flood the network or better employ the much more efficient algorithm described in [4]. This algorithm gossips the information about the tree structure among all the peers in the network. Each peer maintains an approximate number of peers in each sub-trie it belongs to (as many values as the peer's prefix length). The values are exchanged via local interactions between peers and a piggyback mechanism avoids sending additional messages. The same idea is used to gossip the *UpdateRate* in every sub-trie a peer belongs to.

4.4 Example

An example illustrating the application of caching is shown in Figure 2.

Note, that in Figure 2 each element tag is represented by one capital letter and we omit child axes ("/") to simplify the presentation. The numbers 1–4 written in brackets next to the arrows correspond to the following steps:

1. The cache manager at the peer II decides to cache a result set for the query Q = "$A//C/D//E$". The shower broadcast to the peers responsible for h("C/D") is initiated to fill up the cache with all data items matching the query. It reaches the peers I, III and IV. They add Q to their lists of cached queries.
2. Peers III and IV send back the matching items. The shower broadcast reaches peer V, which also adds Q to its list of cached queries. 4 messages were sent to execute the shower broadcast in sub-trie h("C/D").

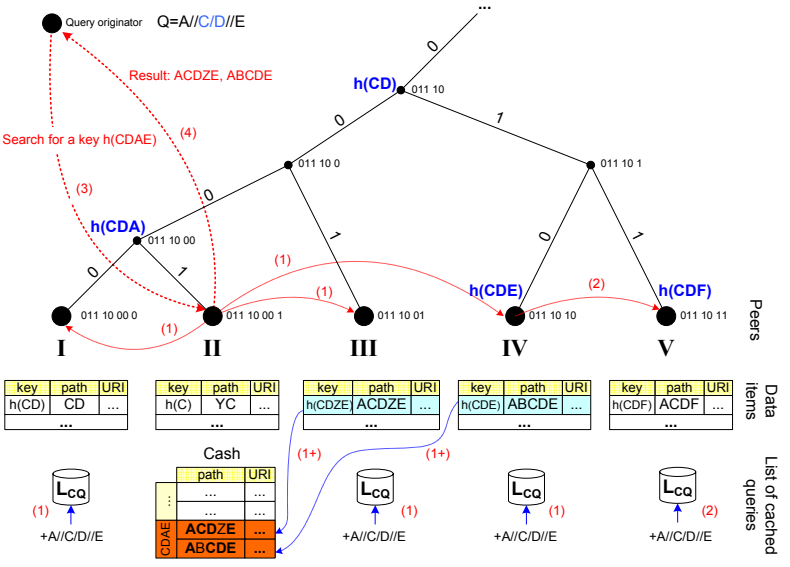


Fig. 2. Caching strategy example

3. Assume, the query $Q =$ "$A//C/D//E$" is submitted at the originating peer. The search message is routed to the peer II ($O(\log N)$), which can answer a query locally by looking up its cache. The broadcast has to be executed every time to answer the query Q if it is not cached.
4. The answer is sent back to the originating peer.

Assume now, a new path "A/C/D/E" is indexed. One of the four (see Section 3) generated data items with the key $h($"$C/D/E$"$)$ is added to the peer V. It checks the list of cached queries and finds query $Q =$ "$A//C/D//E$" to be concerned by this change. Peer V sends a cache update message to the peer responsible for Q, i.e., to Peer II, which ensures cache consistency.

5 Simulations

To justify our approach and its efficiency, we implemented a simulator of a distributed XML storage, based on the P-Grid overlay network. The simulator is written in Java and stores all data locally in a relational database. As the simulation results in this section meet our theoretical expectations we will in a next step implement our approach on top of our P-Grid implementation [22] and test it on PlanetLab.

As input data for our experiments, we use about 50 XML documents (mainly from [28]) from which we extracted a *path collection* of more than 1000 unique paths. Based on each path in the collection we generated four additional paths by randomly distorting the element tags. Using the resulted path collection (about

5000 paths) we generate a P-Grid network by inserting a corresponding number of data items per each path (about 20000 data items overall). P-Grid networks of different sizes can be obtained by limiting the maximum number of data items a peer can store.

For our experiments we generated different *query collections* by randomly removing some element tags from the paths in the path collection. A parameter t specifies query construction and ensures percentage of type 3 ("cachable") queries in the collection.

To emulate the querying process we generated a *query load* of 10000 queries by applying different distributions on the query collection. In the following experiments an average search cost value for given parameters is computed by processing all queries in the query load.

In the first experiment we assume that all possibly "cacheable" queries are in fact being cached. We vary the network size and measure the average cost of answering one query. The query load is uniformly distributed and different t parameters are used. In Figure 3 the first four curves show the average search cost for $t = 0$, 0.5, 0.75 and 1 respectively. Obviously, the more queries are being cached, the lower the search cost becomes. The fifth curve shows the cost of locating at least one peer responsible for the query, i.e., the search cost without shower broadcasts. Evidently, the two last curves coincide because if all queries are cached no shower broadcasts are required.

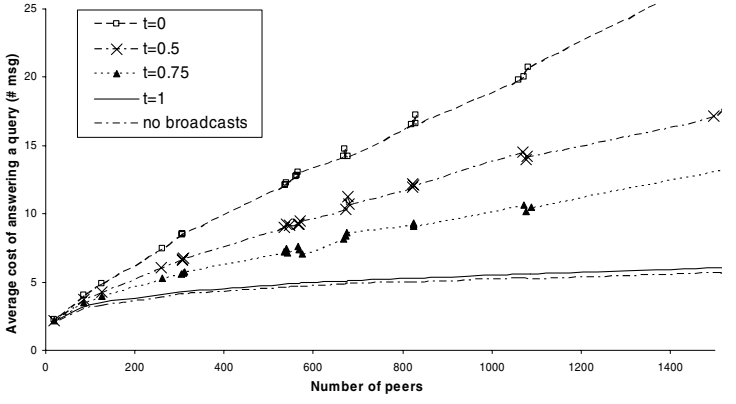


Fig. 3. Average number of messages required to answer a query depending on the network size, t denotes the fraction of "cacheable" queries

However, query load does not necessary follow a uniform distribution. Instead, a Zipfian distribution is more realistic as shown in Figure 4. In the experiment we fixed the network size to 1000 peers, $t = 0.5$ and vary the cache size. The first curve shows the constant search cost if caching is disabled. The other three curves correspond to the different parameters of the Zipf distribution of the query load and show how our approach performs under these conditions.

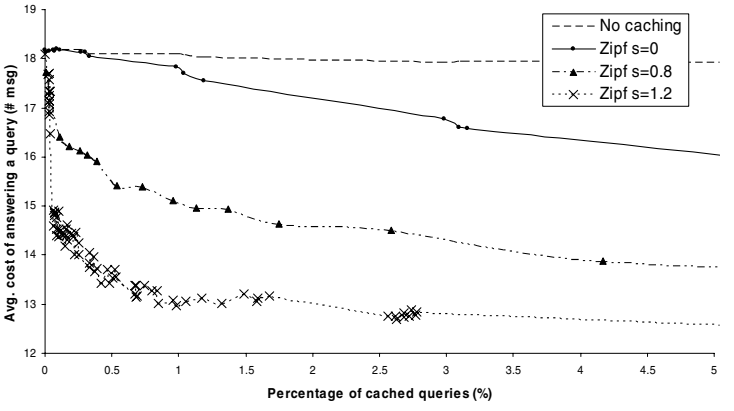


Fig. 4. Average number of messages required to answer a query in the network of 1000 peers depending on the fraction of cached queries

However, the benefits we gain from caching for querying, come at the price of increasing the update costs. To perform one update operation, for example, to insert a new path containing m element tags, we have to contact all the peers responsible for all the subpaths ($O(m \log N)$). We also have to update all relevant cache entries ($O(\log N)$ per cache entry). Figure 5 shows the average update costs depending on size of the network.

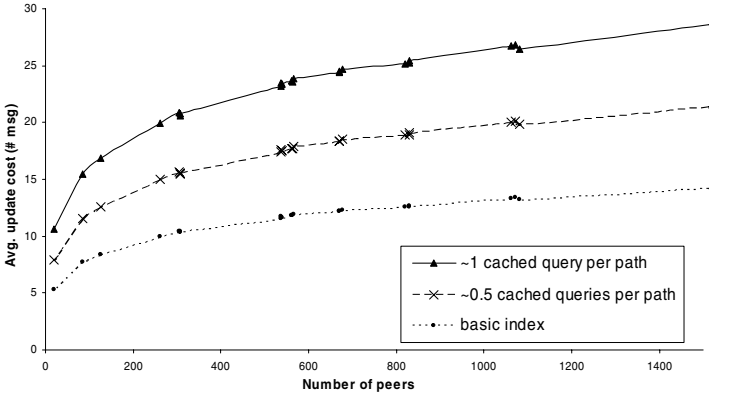


Fig. 5. Average update cost depending on the network size, t denotes the percentage of "cacheable" queries

In Section 4.3 we described the strategy for minimizing the overall messaging costs. In the last experiment we show that for a given state of the system this minimum can be achieved by choosing what queries to cache. In Figure 6 we show that for the given fixed parameters (1000 peers, $t = 0.5$, Zipf $s = 1.2$,

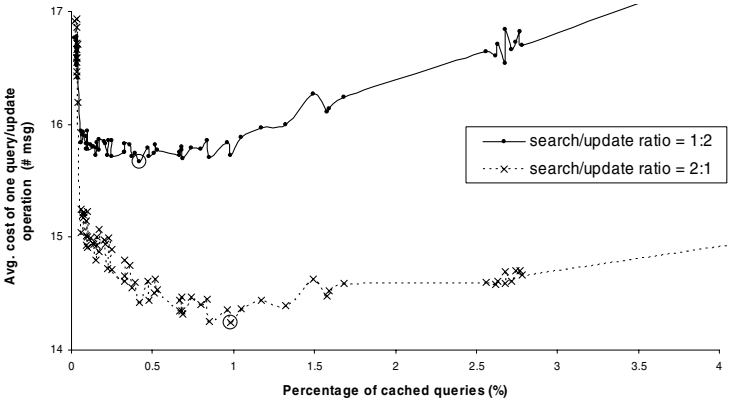


Fig. 6. Average number of messages (query + update) depending on the fraction of cached queries

average number of element tags in the path = 2.5) the overall messaging cost can be minimized. We show two curves for search/update ratios of 1:2 and 2:1. In these cases the minimal messaging costs are achieved if about 0.5% and 1.0% of the queries are being cached.

Evidently, if the search/update ratio is high (more searches than updates) the minimum moves to the right (more queries are to be cached). In contrast, if the update ratio is relatively high, the minimum moves to the left (up to 0, where caching is not profitable anymore). Hence, (1) the higher the search/update ratio is, the more queries should be cached and (2) our solution is adaptive to the current system state and minimizes the overall messaging costs.

The simulations show that the basic index strategy is sufficient for building a P2P XML storage with support for answering structured queries. The introduction of caching decreases the messaging costs. Depending on the characteristics of the query load the benefits from caching vary.

6 Related Work

Many approaches exist that deal with querying of XML data in a local setting. Most of them try to improve the query-answering performance by designing an indexing structure with respect to local data processing. Examples of such index structures include DataGuides [13], T-indexes [20], the Index Fabric [7], the Apex approach [6] and others. However, these approaches are not designed to support a large-scale distributed XML storage.

On the other hand, peer-to-peer networks yield a practical solution for storing huge amounts of data. Thus, a number of approaches exist that try to leverage a P2P network for building a large-scale distributed data warehouse. The important properties of such systems are:

– The flexibility of the querying mechanism (e.g. query language).
– The messaging and maintenance costs.

The use of routing indices [8] facilitates the construction of a P2P network based on content. In such content-based overlay networks peers are linked, if they keep similar data, i.e., each peer maintains summaries of the information stored at its neighbors. While searching, a peer uses the summaries to determine whom to forward a query to. The idea of clustering peers with semantically close content is exploited in [9]. The approach presented in [17] proposes using multi-level bloom filters to summarize hierarchical data, i.e., similarity of peers' content is based on the similarity of their filters. In [23] the authors use histograms as routing indexes. A decentralized procedure for clustering of peers based on their histogram distances is proposed. The content-based approaches could efficiently solve the problem of answering structured queries, though lack of structure affects the result set quality and significantly increases the search cost for large-scale networks.

The Edutella project [21] is a P2P system based on a super-peer architecture, where super-peers are arranged in a hypercube topology. This topology guarantees that each node is queried exactly once for each query, which presumes powerful querying facilities including structured queries, but does not scale well.

Leveraging DHTs to support structured queries decreases the communication costs and improves scalability, but requires more complicated query mechanisms. The approach presented in [12] indexes XML paths in a Chord-based DHT by using tag names as keys. A peer responsible for an XML tag stores and maintains a data summary with all possible unique paths leading to the tag. Thus, only one tag of a query is used to locate the responsible peer. Although ensuring high search speed, the approach introduces considerable overhead for popular tags, when the data summary is large. Our solution for this case is to distribute the processing among the peers in a subtrie. The paper also addresses answering branching XQuery expressions by joining the result sets obtained from different peers. A similar mechanism can be employed for our approach.

[5] also uses a Chord network, but follows a different technique. Path fragments are stored with the information about the super- and child-fragments. Having located a peer responsible for a path fragment, it resolves the query by navigating to the peers responsible for the descendant fragments. Additional information has to be stored and maintained to enable this navigation, which causes additional maintenance costs. For some types of queries the search operation may be rather expensive due to the additional navigation.

Some approaches also employ caching of query results in a P2P network to improve the search efficiency. For example, [18] proposes a new Range Addressable Network architecture that facilitates range query lookups by storing the query results in a cache. In [26] the authors leverage the CAN P2P network to address a similar problem. In both cases queries are limited to integer intervals. The ranges themselves are hashed, which makes simple key search operation highly inefficient.

The PIER project [14, 15] utilizes a DHT to implement a distributed relational query engine bringing database query processing facilities into a P2P environment. In contrast, our approach solves the particular problem of answering structured XPath queries, which is not addressed by PIER. However, many of query processing mechanisms (join, aggregation, etc.) proposed in PIER can be also employed for building a DHT-based large-scale distributed XML storage with powerful query capabilities. The paper [19] leverages the PIER for building a file-sharing P2P system for answering multi-keyword queries. The authors suggest using flooding mechanisms to answer popular queries, and use DHT's indexing techniques for rare queries.

7 Conclusions

In this paper we presented the efficient solution for indexing structural information in a structured overlay network used as distributed P2P storage of XML documents. We based the approach on the P-Grid structured overlay network, however, the solution can be ported to similar tree-based DHTs. We demonstrated the efficiency (low search latency and low bandwidth consumption) of our approach via simulations and also showed that our proposed caching strategy chooses the optimal strategy for minimizing messaging costs.

We envision that the presented solution can be used in a P2P XML querying engine for answering structural (sub)queries. Such a system could be an alternative to the solutions based on the unstructured P2P networks (e.g., Edutella [21]), but more scalable due to the considerably reduced messaging costs. As a next step, we plan to extend the system to support more general XPath queries.

References

1. Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS'01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 179–194, London, UK, 2001. Springer-Verlag.
2. Karl Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *WDAS'02: Proceedings of the 4th Workshop on Distributed Data and Structures*, 2002.
3. Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-Grid: A Self-organizing Structured P2P System. *SIGMOD Record*, 32(3), 2003.
4. Keno Albrecht, Ruedi Arnold, Michael Gahwiler, and Roger Wattenhofer. Join and Leave in Peer-to-Peer Systems: The Steady State Statistics Service Approach. Technical Report 411, ETH Zurich, 2003.
5. Angela Bonifati, Ugo Matrangolo, Alfredo Cuzzocrea, and Mayank Jain. Xpath lookup queries in p2p networks. In *WIDM'04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, pages 48–55, New York, NY, USA, 2004. ACM Press.
6. Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. Apex: an adaptive path index for xml data. In *SIGMOD'02: Proceedings of the ACM SIGMOD 2002 International Conference on Management of Data*, pages 121–132, New York, NY, USA, 2002. ACM Press.

7. Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *VLDB'01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 341–350, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
8. Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS'02: Proceedings of the 28th Int. Conference on Distributed Computing Systems*, July 2002.
9. Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. Technical report, Computer Science Department, Stanford University, 2002.
10. Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *ICDCS'03: Proceedings of the International Conference on Distributed Computing Systems*, 2003.
11. Anwitaman Datta, Manfred Hauswirth, Roman Schmidt, Renault John, and Karl Aberer. Range queries in trie-structured overlays. In *P2P'05: Proceedings of the 5th International Conference on Peer-to-Peer Computing*, August 2005. `http://lsirpeople.epfl.ch/rschmidt/papers/Datta05RangeQueries.pdf`.
12. Leonidas Galanis, Yuan Wang, Shawn R. Jeffery, and David J. DeWitt. Locating data sources in large distributed systems. In *VLDB'03: Proceedings of the 29th International Conference on Very Large Data Bases*, pages 874–885, 2003.
13. Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97: Proceedings of the 23th International Conference on Very Large Data Bases*, pages 436–445, 1997.
14. M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *IPTPS'02: Proceedings for the 1st International Workshop on Peer-to-Peer Systems*, 2002.
15. Ryan Huebsch, Brent Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica, and Aydan R. Yumerefendi. The architecture of pier: An internet-scale query processor. In *CIDR'05: Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2005.
16. Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *STOC'00: Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
17. Georgia Koloniari and Evaggelia Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT'04: Proceedings of 9th International Conference on Extending Database Technology*, pages 29–47, 2004.
18. Anshul Kothari, Divyakant Agrawal, Abhishek Gupta, and Subhash Suri. Range addressable network: A p2p cache architecture for data ranges. In *P2P'03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 14–22, 2003.
19. Boon Thau Loo, Ryan Huebsch, Joseph M. Hellerstein, Scott Shenker, and Ion Stoica. Enhancing p2p file-sharing with an internet-scale query processor. In *VLDB'04: Proceedings of the 30th International Conference on Very Large Data Bases*, August 2004.
20. Tova Milo and Dan Suciu. Index structures for path expressions. In *ICDT'99: Proceeding of the 7th International Conference on Database Theory*, pages 277–295, London, UK, 1999. Springer-Verlag.

21. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: a p2p networking infrastructure based on rdf. In *WWW'02: Proceedings of the eleventh international conference on World Wide Web*, pages 604–615, New York, NY, USA, 2002. ACM Press.
22. http://www.p-grid.org.
23. Yannis Petrakis, Georgia Koloniari, and Evaggelia Pitoura. On using histograms as routing indexes in peer-to-peer systems. In *DBISP2P'04: Proceedings of the Second International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pages 16–30, 2004.
24. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM'01: Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 161–172, 2001.
25. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM'01: Proceedings of the 18th International Conference on Distributed Systems Platforms*, pages 329–350, 2001.
26. Ozgur D. Sahin, Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. A peer-to-peer framework for caching range queries. In *ICDE'04: Proceedings of the 20th International Conference on Data Engineering*, pages 165–176, 2004.
27. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01: Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160. ACM Press, 2001.
28. http://www.cs.washington.edu/research/xmldatasets/.

Community Based Ranking in Peer-to-Peer Networks

Christoph Tempich[1], Alexander Löser[2], and Jörg Heizmann[1]

[1] AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
{tempich, johi}@aifb.uni-karlsruhe.de
[2] CIS, University of Technology Berlin, Einsteinufer 17, 10587 Berlin, Germany
aloeser@cs.tu-berlin.de

Abstract. We address the problem of efficiently ranking the best peers w.r.t. a query with multiple, equally weighted predicates – conjunctive queries – in short-cut overlay networks. This problem occurs when routing queries in unstructured peer-to-peer networks, such as in peer-to-peer information retrieval applications. Requirements for this task include, e.g., full autonomy of peers as well as full control over own resources. Therefore prominent resource location and query routing schemes such as distributed hash tables can not be applied in this setting. In order to tackle these requirements we combine a new resource location and query routing approach that exploits social metaphors of topical experts and experts' experts with standard IR measures for ranking peers based on collection-wide information. The approach has been fully tested in simulation runs for skewed data distributions and network churn and has been partially implemented in the Bibster system [1].

1 Introduction

Finding relevant information from a heterogeneous and distributed set of information resources is a longstanding problem in computer science. Requirements for this task include, for example full autonomy of peers as well as full control over own resources. For this reason prominent resource location and query routing solutions such as distributed hash tables can not be applied in this setting.

Studies of social networks show that the challenge of finding relevant information may be reduced to asking the 'right' people. 'The right people' generally are the ones who either have the desired piece of information and can directly provide the relevant content or the ones who can recommend 'the right people'. Milgram's [20] and Kleinberg's [17] experiments illustrated that people with only local knowledge of the network (i.e. their immediate acquaintances) are quite successful in constructing acquaintance chains of short length leading to 'small world' networks. We observe that such mechanisms in social networks work although

- people may not always be available to respond to requests
- people may shift their interests and attention
- people may not have exactly the 'right' knowledge w.r.t. a particular information request, but only knowledge which is *semantically close* to it.

[1] http://bibster.semanticweb.org

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1261–1278, 2005.

This means that the real-world social networks, as opposed to theoretical network models, are *highly dynamic* w.r.t. peer availability and topic expertise. Therefore real world peer-to-peer scenarios need a concept of *semantic similarity* in order to efficiently answer the information needs of the peers, i.e. to realistically determine 'the right person'. Starting from requirements of semantic search in the setting of distributed, autonomous information sources, in which documents are annotated using Thesaurus-like or Description Logic-like ontologies, we have applied these observations and conceived the INGA algorithm. In this novel peer-to-peer algorithm each peer plays the role of a person in a social network. The novel design principle of INGA lies in the dynamic adaptation of the network topology, adaptation which is driven by the history of successful or semantically similar queries. We realize this in that we bound the local shortcut indices storing semantically labeled shortcuts, and a dynamic shortcut selection strategy. It forwards queries to a community of peers that are likely to best answer them. Shortcuts connect peers that share similar interests and thus spontaneously form semantic communities. Based on local shortcut indexes we select the k-best peers for each query to forward it depending exclusively on collection wide information (CWI).

Contributions and Paper Organisation: Former techniques used gossiping to disseminate a summary of the documents to update local indexes of a peer, an approach with severe scalability short comings. To avoid such drawbacks we create local shortcut indices by observing query and answers. We present new shortcut creation and deletion strategies that cluster peers within semantic communities. Rather than disseminating a query across the network we route queries within such communities to only the best matching peers. Our peer selection is bases on an IR-ranking strategy [9]. We modify this strategy so that it is able to select the top-k peers from a local shortcut index for conjunctive queries. Our extensive simulations using ontologies for Description Logics and thesaurus information show that our approach selects peers and forwards queries with high efficiency even in a highly dynamic setting and with bounded indices.

Our paper is organized as follows: In the next section we review related work, mainly information retrieval techniques and work on shortcut networks. We describe the social metaphors, our infrastructure to maintain the index and our query and results model in Section 3. Section 4 shows the index structure and update strategy for each type of shortcut. Section 5 presents our dynamic routing model. Section 6 describes our simulation methodology and the results of our simulations. Section 7 summarizes the results and points to future work.

2 Related Work

Our approach combines information retrieval (IR) techniques with central indices and work on shortcut networks. We provide a brief overview of both areas in order to situate our contribution. Prior research on distributed information retrieval and meta search engines has addressed the ranking of data sources and the reconciliation of search results from different sources. GlOSS [13] and CORI [4] are among the most prominent distributed IR systems, but neither of them aimed at very large-scale, highly dynamic, self-organizing P2P environments, which were not an issue at the time these systems were developed.

Top-k queries [10] delivering a well-defined set of k best answers according to a user-provided, probably weighted compensation function, have shown their broad applicability, e.g. in content-based retrieval in multimedia collections or digital libraries. Algorithms for top-k retrieval [19, 11] in databases generally try to minimize the number of database objects that have to be accessed before being able to return a correct result set of the k best matching objects. Previous work in distributed top-k retrieval, however does not cover all the issues of high dynamics and autonomy which are unavoidable in a P2P network. Feasible information retrieval solutions need to cope with the following problems:

- Ranked Retrieval Model: In this setting the ranked retrieval models should be adopted in such a way that the models avoid flooding the entire network when selecting the top-k peers for a query with a conjunction of multiple predicates.
- Network Churn: Local index structures should provide all necessary information about what documents are available and should adapt to changing local documents of remote peers or peers joining or leaving the network.
- Collection-Wide Information: Queries should be answered only using collection-wide information, e.g. stored in local shortcut indices. Without constantly disseminating this information a peer should ideally restrict collecting information it is interested in in its index.

In the context of peer-to-peer networks only very few authors have explored retrieval algorithms which are based on collection-wide information for ranking peers in volatile networks. PlanetP [9] concentrates on peer-to-peer communities in unstructured networks with sizes up to ten thousand peers. They introduce two data structures for searching and ranking which create a replicated global index using gossiping algorithms. Each peer maintains an inverted index of its documents and spreads the term-to-peer index across the network. Inspired by the simple TFxIDF metric and based on the replicated index a simple ranking algorithm using the inverse peer frequency is implemented. However, by using gossiping to disseminate Bloom filters the system's scalability is severely limited. In section 5 we show how to apply the ranking strategy to our local shortcut indices to reduce the communication overhead for the dissemination of a peer's summary .

Local index information was first introduced by [7] to improve the efficiency of Gnutella routing indices. This indexing strategy locally stores information about specific queries and about peers which were successfully queried in the past. [22] first considers the semantics of the query to exploit interest-based locality in a static network. They use shortcuts that are generated after each successful query and are used to further requests, hence they are comparable to content provider shortcuts, which we introduce next. Their search strategy differs from ours, since they only follow a shortcut if it exactly matches a query, else they use a flooding approach. To update the index they use a LRU strategy, while we utilize also semantic similarity. In a similar way, [3] uses a local routing index for content provider shortcuts for the specific scenario of top k retrieval in P2P networks. Local indices are maintained in a static super-peer network. Their index policy considers temporal locality, each index entry has a certain time to live after which the shortcut has to be re-established for the next query on that topic. However, the emphasis in the approach is on appropriate topologies for overlay

networks. The paper develops efficient routing methods among static super-peers in a hypercube topology, while we consider dynamic networks without super-peers.

3 System Architecture

Our peer selection strategies described in section 4 can be integrated into any unstructured P2P network system. However, for evaluation purposes we used the SWAP infrastructure [15], which provides all standard peer-to-peer functionality such as information sharing, searching and publishing of resources.

3.1 Social Metaphors

In INGA, facts are stored and managed locally on each peer, thus constituting the 'topical knowledge' of the peer. A peer responds to a query by providing an answer matching the query or by forwarding it to whom it deems to be the most appropriate peers for this task. For the purpose of determining the most appropriate peers, each peer maintains a *personal semantic shortcut index*. The index is created and maintained in our highly dynamic setting in a lazy manner, i.e. by analyzing the queries that are initiated by users of the peer-to-peer network and that happen to pass through the peer. The personal semantic shortcut index maintained at each peer reflects that a peer may play the following four different roles for the other peers in the network (roles are listed in decreasing order of utility):

- The best peers to query are always those that already have answered it or a semantically similar one in the past successfully. We call such peers *content providers*.
- If no content providers are known, peers that have *issued semantically similar queries* in the past are queried. The assumption is that this peer has been successful in getting matching answers and now we can directly learn from it about suitable content providers. We call such peers *recommenders*.
- If we do not know either of the aforementioned items we query peers that have established a good social network to other ones over a variety of general domains. Such peers form a *bootstrapping network*.
- If we fail to discover any of the above we return to the default layer of neighboring peers. To avoid over-fitting to peers already known we occasionally select random peers for a query. We call this the *default network*.

From a local perspective, each peer maintains in its index information about some peers, about what roles these peers play for which topic and how useful they were in the past. From a global perspective, each of the four roles results in a network layer of peers that is independent from the other layers.

3.2 Building Blocks

We assume that each peer provides a unique peer identifier (PID). Similar to file sharing networks each peer may publish all resources from its *local content database*, so that

other peers can discover them by their requests (this also applies to resources downloaded from other peers). Information is wrapped as RDF statements and stored in an RDF repository[2]. Additionally to local meta data (e.g. *Robert Meersman isOrganizerOf ODBASE2005*) each resource is assigned one or more topics (such as *ODBASE2005 isTypeOf OTMConference*) and hierarchical information about the topics is stored *(OTM subTopicOf Conference)*. The topics a peer stores resources for are subsequently referred to as the peers own topics. Note, that our algorithm does not require a shared topic hierarchy, though this restriction provides certain advantages. In particular the ranking of shortcuts depends partly on the availability of a shared hierarchy. Therefore we use in all our experiments a shared hierarchy. For successful queries (own queries or those of other peers), which returned at least one match, the *shortcut management* extracts information about answering and forwarding peers to create, update or remove shortcuts in the *local shortcut index*. The *routing logic* selects 'most suitable' peers to forward a query to, for all own queries or queries forwarded from remote peers. The selection depends on the knowledge a peer has already acquired for the specific query and the similarity between the query and locally stored shortcuts.

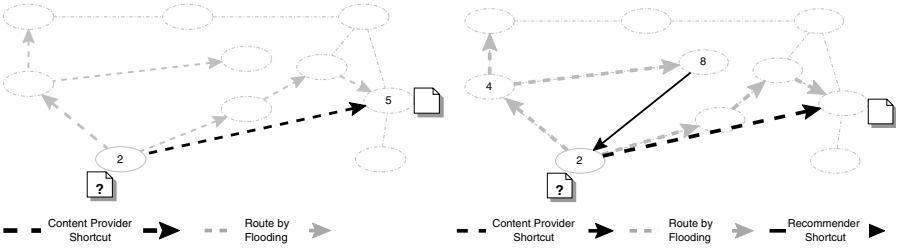
3.3 Query and Result Messages

We use a simple query message model which is similar to the structure of a Gnutella query message[12]. Each query message is a quadruple: $QM(q, b, mp, qid)$ where q is a SERQL query. In this paper we present ranking techniques for queries with multiple predicates such as: *Select all resources that belong to the topic semantic web and to the topic p2p*. In the Semantic Web context we formalize this query using common topic hierarchies, such as the Open Directory: *Find any resource with the topics /computer/web/semanticweb* $\wedge$ */computer/distributed/p2p* . More generally: q is a set of n predicates $q = (p_1...p_n)$. where each predicate is defined in a common ontology, b is the bootstrapping capability of the querying peer to allow the creation of bootstrapping shortcuts, mp the message path for each query message containing the unique PIDs of all peers, which have already received the query, to avoid duplicated query messages, and qid a unique query ID to ensure that a peer does not respond to a query it has already answered. Unique query IDs in INGA are computed by using a random number generator that has sufficiently high probability of generating unique numbers. A result message is a tuple: $RM(r, mp, qid)$ where r represents the answer to the query. We just consider results which exactly match the query. Besides, the message path mp is copied to the answer message to allow the creation of recommender and content provider shortcuts.

4 Building and Maintenance of the Index

Each peer is connected to a set of other peers in the network via uni-directional shortcuts. Hence, each peer can locally select all other peers it wants to be linked to. Following the social metaphors in section 1, we generally distinguish between the following types of shortcuts:

[2] `http://www.openrdf.org/`



(a) Content provider shortcut creation (b) Recommender shortcut creation

Fig. 1. Topic specific shortcut creation

4.1 Content Provider and Recommender Shortcuts

Content Provider Layer: The design of the content provider shortcut overlay departs from existing work as published by [22, 23] and exploits the simple, yet powerful principle of interest-based locality. When a peer joins the system, it may not have any information about the interest of other peers. It first attempts to receive answers for its queries by exploiting lower layers of the INGA peer network, e.g. by flooding. The lookup returns a set of peers that store documents for the topic of the query. These peers are potential candidates to be added to the content provider shortcut list. Each time the querying peer receives an answer from a remote peer, content provider shortcuts *sc* for each query topic to new remote peers are added to the list in the form: *sc(*$topic_i$*, pid, query hits,'c', update)*, where $topic_i$ is one of the topics taken from the query message and *pid* is the unique identifier of the answering peer. *Query hits* total the number of returned statements from a remote peer for a particular topic. *'c'* is the type of content provider shortcuts and *update* is the time, when the shortcut was created or the last time, when the shortcut was used successfully. Subsequent queries of the local peer or of a remote peer are matched against the topic column of the content provider shortcut list. If a peer cannot find suitable shortcuts in the list, it issues a lookup through lower layers, and repeats the process for adding new shortcuts. For example consider Figure 1(a) and Table 1: Peer 2 discovers shortcuts for the conjunctive query $/computer/web/semanticweb \wedge /computer/distributed/p2p$ by flooding the default network with a maximum number of three hops (TTL) and creates content provider shortcuts to peer 5.

Recommender Layer: To foster the learning process of recommender shortcuts, especially for new peers in the network, we consider the incoming queries that are routed through a peer. Similar to a content provider shortcut a recommender shortcut *sc(*$topic_i$*,pid,query hits maxsim,rp, update)* is created for each topic in the query. The

Table 1. Content Provider Creation

PID	Topic	Query Hits	Type	Update
5	/computer/web/semanticweb	300	C	2005:31:05:16:37:34
5	/computer/distributed/p2p	300	C	2005:31:05:16:37:34

Table 2. Recommender Shortcut Creation

PID	Topic	Query Hits	Type	Update
2	/computer/web/semanticweb	1	R	2005:31:05:16:37:34
2	/computer/distributed/p2p	1	R	2005:31:05:16:37:34

PID of a shortcut is extracted from the query message as the PID of the querying peer. Since we will not get any information about the number of results retrieved for the query, we set the number of query hits to 1. Finally, *r* indicates the type of shortcut for passive recommender shortcut and *update* is the time, when the shortcut was created or the last time, when the shortcut was used successfully. For example consider again Figure 1(b). Peer 2 issues the query $/computer/web/semanticweb \wedge /computer/distributed/p2p$. Peer 8 creates a shortcut to peer 2 since this query was routed through peer 8 as shown in table 2.

Content Provider and Recommender Index: We assume that each peer may only store a limited amount of shortcuts, hence it only knows a limited set of topic specific neighbors it can route a query to. If the local index size is reached a peer has to decide which shortcut should be deleted from the index. For each shortcut in the index we compute a rank based on the following types of localities:

Semantic locality. We measure the maximum semantic similarity $maxsim$ between the topic of a shortcut and the topics represented by the local content of a peer according to equation 1. Hence, we retain a shortcut about topic t to a remote peer, if t is close to our own interests. We define the similarity function $sim : t_1 \times t_2 \rightarrow [0; 1]$ between two terms in the same topic hierarchy, according to [18]:

$$sim_{Topic}(t_1, t_2) = \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } t_1 \neq t_2 \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

where l is the length of the shortest path between t_1 and t_2 in the graph spanned by the sub-topic relation and h is the minimum level in the topic hierarchy of either t_1 or t_2. α and β are parameters scaling the contribution of shortest path length l and depth h, respectively. Based on the benchmark data set given in [18], we chose $\alpha = 0.2$ and $\beta = 0.6$ as optimal values.

LRU locality. To adapt content and interests changes we use the LRU replacement policy [2]. Shortcuts that have been used recently receive a higher rank. Each local shortcut is marked with a timestamp indicating when it was created. The timestamp will be updated, if the shortcut will be used successfully by the local peer. There is thus an 'oldest' and 'latest' shortcut. The value $update \in [0..1]$ is normalized by difference between the shortcut's timestamp and the 'oldest' time stamp divided by the difference between the 'latest' and the 'oldest'.

Community locality. We measure how close a shortcut leads us to a document. Content provider shortcuts, marked with a c, provide a one hop distance, therefore we set $type = 1$. Recommender shortcuts, marked with a r require at least two hops to reach a peer with relevant documents. In this case we set $type = 0.5$.

We weight the localities using a weighted moving average and compute the index relevance according to equation 2.

$$relevance = \frac{a * maxsim + b * type + c * update}{a + b + c} \quad (2)$$

Shortcuts with the highest relevance are ranked at the top of the index, while peers with a lower relevance are deleted from the index.

4.2 Bootstrapping Shortcuts

Bootstrapping shortcuts link to peers that have established many shortcuts for different query topics to many remote peers. We determine the bootstrapping capability by analyzing the in-degree and out-degree of a peer. We use the out-degree as a measure of how successful a peer discovers other peers by querying. To weigh the out-degree we measure the amount of distinct sources a peer receives queries from. We use the in-degree as a measure, that such a peer may share prestigious shortcuts with a high availability. By routing a query along bootstrapping shortcuts, we foster the probability to find a matching shortcut for a query and avoid the drawbacks of having to select peers randomly, e.g., by flooding.

Discovery and Update: Each incoming query that is stored in our index includes the bootstrapping information of the querying peer. While a peer is online it continuously updates its content/recommender index based on incoming queries and stores additional bootstrapping shortcuts in the form *sc(pid, bo)*, where *pid* is the PID of the querying peer and *bo* its bootstrapping capability. Once an initial set of bootstrapping nodes is found, a peer may route its queries to the nodes with the highest bo value. The bo value is calculated using equation 3

$$Bo = (1 + |outdegree|) \times (1 + |indegree|) \quad (3)$$

where *out-degree* is the number of distinct remote peers one knows. To compute an approximation of the *in-degree* without any central server we count the number of distinct peers that send a query via one's peer. To do this from the message path of indexed recommender shortcuts we examine the pen-ultimate peers. The number of distinct pen-ultimate peers denotes one's in-degree. To avoid zero values we limited the minimum for both values to 1.

4.3 Default Network Shortcuts

When a new peer enters the network, it has not yet stored any specific shortcuts in its index. Default network shortcuts connect each peer p to a set of other peers (p's neighbors) chosen at random, as in typical Gnutella-like networks (e.g., using rendezvous techniques).

5 Dynamic Shortcut Selection

The basic principle of shortcuts consists of dynamically adapting the topology of the P2P network so that the peers that share common interests spontaneously form well-

connected semantic communities. It has been shown that users are generally interested in only some specific types of content[8]. Therefore being part of a community that shares common interests is likely to increase search efficiency and query success rate. To build semantic communities, for each query INGA executes the following steps:

Across the network: Selecting top-k peers: Whenever a peer receives a query message, it first extracts meta-information about the querying peer and updates its index if needed. Then our forwarding strategy is invoked to select a set of *k* peers which appear most promising to answer the query successfully. Finally, the original query message is forwarded to these *k* peers.
Across the network: Answering Queries: When a peer receives a query, it will try to answer the query using local content. We only return non-empty, exact results and send them directly to the querying peer. If the maximum number of hops is not yet reached, the query is forwarded to a set of peers selected as above.
Locally: Receiving Results: A querying peer analyzes the message path of result item and the number of results to create or update local content provider and recommender shortcuts.

Community based top-k rank: A naive approach would route a query only to a peer that matches *all* predicates of the query using a simple exact match paradigm. Too specific query predicates under the exact match paradigm often lead to empty result sets. Therefore the notion of best matches and relative importance of predicates can be a good alternative to satisfy a user's information needs independently of the individual peer instances. To rank peers we use a measure called the *inverse peer frequency* (IPF), based on TFxIDF, a popular method for assigning term weights. This measure was first introduced by [9] in the PlanetP system to rank peers efficiently for multiple query terms and local indices that are built via gossiping bloom filter summaries. However to create local indices we use social metaphors and interest based locality. We calculate the rank R for a peer p for a query q using formula 4, where N represents the number of distinct peers in one's shortcut index, N_i represents the number of peers providing documents for topic t and q_i^p the query hits q per topic t of each peer N_i.

$$R_p(q) = \sum_{i=1}^{t} q_i^p * \log(1 + \frac{N}{N_i}) \tag{4}$$

Intuitively, this scheme gives peers that contain all terms in a query the highest ranking.

Dynamic peer selection algorithm: The task of the INGA shortcut selection algorithm *Dynamic* is to determine best matching candidates to which a query should be forwarded. We rely on forwarding strategies, depending on the local knowledge for the topic of the query a peer has acquired yet in its index:

- We only forward a query via its *k best matching* shortcuts.
- We try to select content and recommender shortcuts before selecting bootstrapping and default network shortcuts.
- To avoid overfitting queries are also randomly forwarded to selected remote peers.

Algorithm 1. Dynamic

Require: **Query** q, **int** k,
Ensure: $TTL_q < maxTTL$
1: $s \leftarrow$ *TopIPF(q,Content/RecommenderShortcuts,*(k))
2: **if** $(|s| < k)$ **then**
3: $s \leftarrow s$ + *TopBoot(BootstrappingShortcuts,*$(k - |s|)$)
4: **end if**
5: $s \leftarrow$ *RandomFill(s,defaultNetworkShortcuts,f,k)*
6: **Return** s.

In step 1 of algorithm *Dynamic* method *TopIPF* browses through the index of all content and recommender shortcuts and identifies the k peers with the highest $R_p(q)$. If less then *k* peers are found we select peers from the top bootstrapping shortcuts (step 3) in subroutine *TopBoot*. It works similarly to *TopIPF*, but selects the peers with highest bootstrapping capability from the index. It also avoids overlapping peers within the set of selected shortcuts. Finally, in subroutine *RandomFill* we fill up the remaining peers randomly from the default network and return the set of selected peers. The algorithm's task is twofold: if the other subroutines fail to discover k peers for a query, it fills up remaining peers until k is reached. The second task of the algorithm is to contribute with some randomly chosen peers to the selected set of k peers to avoid overfitting of the selection process as known from simulated annealing techniques [16]. The *Dynamic* algorithm terminates if the query has reached its maximum number of hops.

6 Experimental Evaluation

Our approach is partially implemented in the Bibster system, which was evaluated in a real world case study [14]. During the case study a maximum of 50 researchers were simultaneously online. This number is too small to evaluate the scalability of a peer-to-peer routing algorithm. Although we have collected information about the content and the queries the peers have shared and submitted, again the amount of available information was too small to test our algorithm for a larger number of peers. Therefore, we have opted for simulating a peer-to-peer network with more than 1.000 peers and to generate data sets considering different types of ontologies and content distributions.[3] In particular, we have created two synthetic data sets to evaluate our approach. The data sets are distributed among the peers. From the data set we further generated candidate queries which can be submitted by the peers. Before we present the results of our evaluation the distribution parameters to generate the data sets are described.

6.1 Simulation Setup

Content: We have generated two ontologies with instances. The generation of the ontologies is based on the observation in [24] and [5]. In [24] it was observed that on-

[3] According to our knowledge this is the first approach to create a synthetic data set for semantics based peer-to-peer systems. To create the data set we combined results from different analysis regarding distribution parameters for ontologies, content, content distribution and queries. The data set is available online at `http://ontoware.org/projects/swapsim/`.

Table 3. Parameter setting for ontology types

	Thesaurus-like		Description logic-like	
	No.	Distribution	No.	Distribution
$NoOfClasses$	1.000		100	
$NoSubClasses$	$\overline{\ln(1000)} = 7$	Zipf(1.1)	$\overline{\ln(100)} = 5$	Zipf(1.1)
$TotalNoProperties$	357		213	
$NoProperties$	0–5	Zipf(2.5)	0–7	Zipf(0.9)
$NoOfInstances$	200.000	Zipf(1)	200.000	Zipf(1)
$TotalNoPropertyInstances$	35.700	Zipf(1)	21.300	Zipf(1)

tologies publicly available[4] follow broadly three different design characteristics while the distinguishing factors are the number of subclasses of a class, the number of restrictions and the number of properties. The three types of ontologies can be described as (1) Thesaurus-like, (2) Database-like and (3) Description Logic-like structures. The first one defines a large hierarchy with many classes and few restrictions and properties per class. Database-like ontologies are characterized by defining a medium number of primitive classes and equal number of restrictions per class. Finally, Description Logic-like ontologies have a high number of restrictions and properties per class and only a small number of primitive classes.

We have created two synthetic ontologies representing the two extreme types of ontologies namely a Thesaurus-like and a Description Logic-like ontology. The parameters to create the ontologies were set according to the observations made in [24]. Unfortunately, in [24] no information about instance data is available. Therefore, we use the content distribution model of [5] to generate instances of the ontology. The distribution of content in our Bibster case study is comparable to the generated content distribution.

For the Thesaurus-like ontology we generated $NoOfClasses = 1.000$ classes[5]. Each class was assigned a popularity following a Zipf distribution with parameter set to 1. The popularity index is later used to generate the ranges of properties and instances. The classes were then arranged in a sub-class of hierarchy. The maximum depth of the hierarchy was set to $MaxDepth = \ln(NoOfClasses)$. Each class could have a maximum of $NoSubClasses = \ln(NoOfClasses)$ and a minimum of $NoSubClasses = 0$. The number of sub-classes for each class follows a Zipf distribution with parameter set to 1.1. The sub-classes are chosen randomly from the classes not already modelled in the hierarchy.

Similar to the hierarchy generation each class can have a maximum of $NoProperties = 5$ and a minimum of $NoProperties = 0$ number of properties. The number of properties per class, or the number of properties a class is domain for, follows a Zipf distribution with a skew parameter set to 1.1. The ranges of the properties were selected from all classes according to their popularity. In total, the ontology contains $TotalNoProperties = 357$ properties.

[4] The analysis was based on the ontologies available in the DAML ontology library.

[5] Our algorithm was first conceived for topics organized in a topic hierarchy. In order to apply it to the generated ontologies we interpret classes as topics and instances as documents.

We have instantiated the classes with $NoOfInstances = 200.000$ instances taking into account the popularity of the classes. We have further instantiated the properties with $TotalNoPropertyInstances = 100 * TotalNoProperties$ properties. Equally to the instantiation of the classes a property carries a popularity rank following a Zipf distribution with parameter set to 1. After selecting a property according to its popularity, we instantiated it using the instances of its domain and range. We selected the instances according to a Zipf distribution with parameter set to 1.

The parameters used for the Thesaurus-like and Description Logic-like ontologies are summarized in Table 3.

Content Distribution: In order to distribute the content over the peers we adopt the model presented in [8]. Generally, users are interested in a subset of the content available in a peer-to-peer network. Furthermore, they are only interested in a limited number of topics (e.g., databases, Description Logic). Moreover, they are more interested in some classes while less in others. A peer can be interested in and have content for $ClassesOfInterest = \ln NoOfClasses * 2$ classes. The number of classes a peer is interested in is chosen randomly from a uniform distribution. The classes it is interested in are selected randomly from the available classes taking into account their popularity. As observed in [1] not all users share the same amount of data. Most of them do not contribute anything to the available knowledge while a small proportion makes two thirds of the knowledge available. Based on the study in [1] we assigned the following storage capacity to the peers in the network: 70% of the peers do not share any instances (free-riders); 20% share 100 instances or less; 7% share between 101 and 1000 instances; finally, 3% of the peers share between 1001 and 2000 instances (actual storage capacities are chosen uniformly at random). A peer sharing an instance knows all its properties and the type of the range instance.

Query Set: In order to test our algorithm we have generated two different types of queries. All queries ask for instances satisfying a varying number of constraints. The first query type instantiates the blueprint $(instance; isTypeOf; class) \wedge (instance; property; instance2) \wedge (instance2; isTypeOf; class2)$. We thus query for all $instance$s of a certain $class$ with the constraint that the instance has one particular $property$ pointing to another instance. The range of the $property$ determines the type of $instance2$. The second query type extends the first by adding a constraint on the properties and instantiates the blueprint $(instance; isTypeOf; class) \wedge (instance; property; instance2) \wedge (instance2; isTypeOf; class2) \wedge (instance; property2; instance3) \wedge (instance3; isTypeOf; class3)$. We have summarized the number of distinct queries for the two ontology types in table 4. We have only generated queries to which at least one answer exists.

Table 4. Number of queries for the ontology types

Query Type	Description	Ontology Type	
		Thesaurus-like	Description logic-like
QT1	1 class and 1 property	2194	8493
QT2	1 class and 2 properties	1087	8502

Queries are chosen randomly from all available queries. We choose a uniform query distribution instead of a ZIPF-distribution, which is typically observed in file sharing networks [21]. This simulates the worst case scenario, in which we do not take advantage of frequent queries for popular topics.

Gnutella style network: The simulation is initialized with a network topology which resembles the small world properties of file sharing networks[6]. We simulated 1024 peers. In the simulation, peers were chosen randomly and they were given a randomly selected query to question the remote peers in the network. The peers decide on the basis of their local shortcut which remote peers to send the query to. Each peer uses INGA to select up to $pmax = 2$ peers to send the query to. Each query was forwarded until the maximum number of hops $hmax = 6$ was reached.

Volatile network and interest shifts: We implemented the dynamic network model observed for Gnutella networks of [21]: 60% of the peers have a availability of less then 20%, while 20% of the peers are available between 20 and 60% and 20 % are available more then 60%. Hence, only a small fraction of peers is available more than half of the simulation time, while the majority of the peers is only online for a fraction of the simulation time. Users' interest may change over time, e.g. to account for different search goals. To simulate changing interests, after 15 queries, equal to approx. 15.000 queries over all peers, each peer queries a complete different set of topics.

Evaluation Measures: We measure the search efficiency using the following metrics:

- **Recall** is a standard measure in information retrieval. In our setting, it describes the proportion between all relevant documents in peer network and the retrieved ones.
- **Messages** represent the required search costs per query that can be used to indirectly justify the system scalability.
- **Message Gain** To compare the recall per message we introduce the message gain, i.e. the proportion of messages with respect to the achieved recall. We define the message gain formally as

$$messagegain = \frac{Recall}{|Messages|} \tag{5}$$

6.2 Evaluation Results

We have evaluated our approach with several parameter settings. To analyze the influencing variables. If not stated otherwise all simulations were performed using the Thesaurus-like ontology with the second Query Type. We used the Bootstrapping layer and the *TfxIPF* metric to combine the query hits for the shortcuts. Each peer can maintain a shortcut index with 40 shortcuts.

[6] We used the Colt library `http://nicewww.cern.ch/~hoschek/colt/`

Shortcut indices enable efficient peer selection: It was already shown in [23] that shortcuts enable efficient query routing in unstructured peer-to-peer networks for single class queries. In figure 2(a) we have plotted the results comparing the recall for different selection strategies. We refer to the ranking based on equation 4 as *TfxIPF*. After a warm up phase of 2.000 queries, or approximately two queries per peer, we constantly reach around 75% of the available content. Not all peers are always online, thus we have plotted the maximum available content as *Online Available* in the graph. We compare our selection strategy to other available combination functions known in the database community and the *naive* approach. The naive approach represents the baseline for all algorithms likewise. In this approach we use only the default layer to select peers as in Gnutella. From the known peers on the default layer, we randomly select two remote peers to send and forward a query to.

The selection function described in [6] uses an equation similar to 6 to combine query hits in a distributed document retrieval scenario. We refer to this strategy as *Multiply*.

$$R_p(q) = \prod_{i=1}^{t} q_i^p \tag{6}$$

However, besides the query terms contained in the queries [6] also uses the most frequent words contained in the retrieved documents to create the index. This is not possible in our application. Furthermore, the total number of queries and total number of query hits are considered in the ranking function. These are optimizations which we have not considered so far, in order to be able to compare the different combination functions.

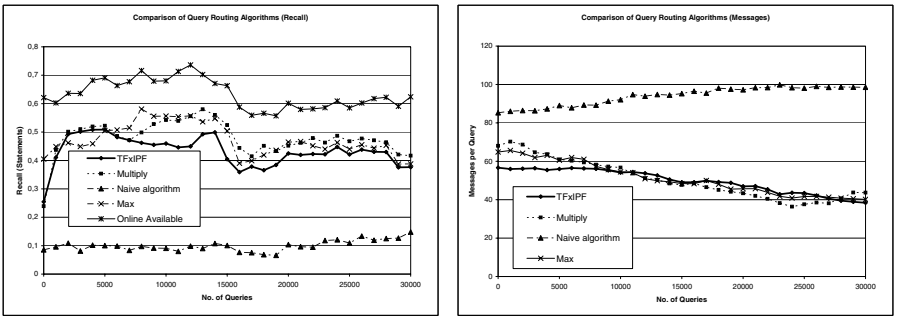
Additionally, we have evaluated equation 7 to select remote peers. The *Max* evaluation function selects the peer which has delivered the most statements for one of the query terms contained in the query.

$$R_p(q) = max(q_i^p) \tag{7}$$

Intriguingly, all three evaluation functions result in almost the same recall values. This is due to the high proportion of content delivered by a relatively small number of peers in typical peer-to-peer systems. We thus can conclude that shortcuts deliver an efficient yet simple way to boost peer selection quality.

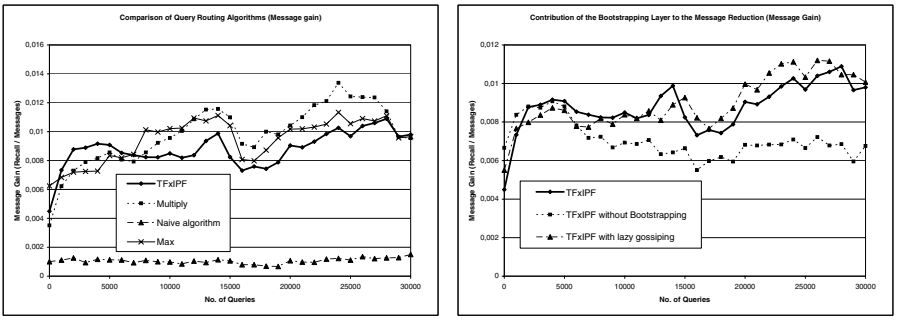
So far we have only considered the recall to evaluate our approach. In figure 2(b) we have plotted the number of messages produced to achieve this recall. The number of messages produced by the *Naive* approach slightly increases over time. Due to the high network churn the peers have to discover new remote peers since the available ones assigned in the setup phase are offline. Thus, the chances to send a query to a remote peer which has not received the query yet increase over time.

In contrast to the observation made for the *Naive* approaches the number of messages produced based on the shortcut selection decreases significantly. The number of messages decreases because different remote peers forward a query to the same peers. Each peer treats a query only once, thus double postings decrease the number of messages used. In figure 3(a) we have combined the two measurers for recall and messages and plotted the message gain. As expected the message gain increases because the recall stays at the same level and the number of messages decreases.



(a) Recall (b) Messages

Fig. 2. Comparison Related Approaches: Dynamic Network 1024 Peers, 6 Hops, k=2

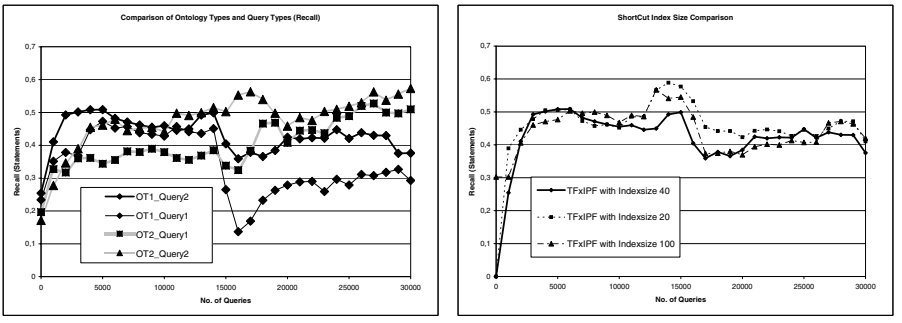


(a) Related Approaches (b) Layer Contribution

Fig. 3. Comparison of Message Gain: Dynamic Network 1024 Peers, 6 Hops, k=2

Bootstrapping decreases messages: In figure 3(b) we compare the performance of our algorithm with and without the bootstrapping layer. The bootstrapping metric favors remote peers which are well-connected and receive a high number of queries. In case a peer has no information about any of the classes used in the query it will forward the query to the best bootstrapping peers or - in case bootstrapping is not used - to randomly selected peers found on the network layer. As the bootstrapping capabilities of the peers are disseminated across the network, more peers select the same remote peers and thus the number of messages decreases. The recall is almost not affected by this decrease and thus the message gain increases.

Gossiping does not increase the message gain: Gossiping is an established method to boost peer selection performance. We have implemented a lazy gossiping algorithm for comparison reasons. Each time a peer does not receive an answer to a query it sends around a discovery message to its neighbors on the network layer. The discovery message returns all classes a remote peer has content for. The result is treated like an incoming query, thus we create a number of recommender shortcuts for the returned



(a) Ontology Types (b) Index Size

Fig. 4. Influencing variables: Dynamic Network 1024 Peers, 6 Hops, k=2

classes. We observe in figure 3(b) that gossiping is not advantageous in comparison to our standard selection algorithm.

Performance of INGA is independent from the underlying ontology type: In figure 4(a) we compare the recall of INGA in case of changing Ontology Types and varying queries. We observe that the recall increases quickly using our approach independent of the underlying ontology and query type. However, when the queries change after 15.000 queries the recall for Query Type 1 and the Thesaurus-like ontology decrease strongly. We conclude that the high number of properties for the Description-logic style ontologies facilitates the quick emergence of an efficient shortcut network.

Performance of INGA is robust against varying index sizes: In figure 4(b) we have compared the recall of INGA with a varying index size. The size of the index determines the required resource allocation for routing purposes at the local peer. We have varied the index size starting from 20 shortcuts, to 40 and finally 100 shortcuts per peer. The examination of the results reveals, that INGA is very robust against the change of the index size.

7 Summary and Outlook

In this paper we present the first semantic query routing algorithm for a completely distributed, dynamic and unstructured peer-to-peer system. Our routing algorithm is based on the creation of local shortcuts and the maintenance of a small shortcut index. The shortcuts memorize the number of instances a remote peer has answered for a conjunctive query. While the local maintenance of the shortcut index allows for complete control over the usage of the shortcuts, no summary of the peers' content must be disseminated to other peers. In our extensive evaluations we show that our shortcut indices are efficient for peer selection in dynamic peer-to-peer networks.

We have introduced a novel metric to characterize well-connected peers, viz. bootstrapping peers. They maintain a high number of shortcuts and have many incoming

connections. We provide evidence that bootstrapping is a good metric to reduce the number of messages. We have compared different metrics to combine local shortcuts and shown that the right index strategy and shortcut creation strategy are more important for efficient routing than right combination metrics. A small index size is sufficient to guarantee a high performance. Intriguingly, our comparisons show that routing based on shortcuts is as efficient as gossiping algorithms.

Our evaluations are based on the first available synthetic data set to setup the knowledge base of the peers. We were able to demonstrate that our approach is equally suitable for Description Logic-like ontologies as well as Thesaurus-like ontologies.

With the full support of conjunctive queries our algorithms are perfectly suitable for application around the social semantic desktop and other applications in which local control about available information is important. Current research in keyword based distributed document retrieval suggests that our shortcut update strategies can be further improved in the future. As more instantiated ontologies become available an evaluation of our approach with a real data set is desirable. An interesting additional problem is the generalization of our approach for a network with individual semantics on each peer. Peers within the same community may share their facts and possibly agree on a common set of semantics. Such a community search engine would enable flexible and efficient wide area knowledge sharing applications without the maintenance of central indexing servers or a static semantic structure.

Acknowledgement. Research reported in this paper has been partially financed by EU in the IST project SEKT (IST-2003-506826). Alexander Löser was generously funded by the German Research Society, Berlin-Brandenburg School in Distributed Information Systems (DFG grant GRK 316/3).

References

1. E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oktober 2000.
2. A. V. Aho, P. J. Denning, and J. D. Ullman. Principles of optimal page replacement. *J. ACM*, 18(1):80–93, 1971.
3. W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *21st International Conference on Data Engineering (ICDE)*, Tokyo, Japan, 2005.
4. J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28, New York, NY, USA, 1995. ACM Press.
5. V. Cholvi, P. Felber, and E. Biersack. Efficient search in unstructured peer-to-peer networks. *European Transactions on Telecommunications: Special Issue on P2P Networking and P2P Services*, 15(6):535–548, November 2004.
6. B. Cooper. Guiding queries to information sources with InfoBeacons. In *ACM/IFIP/USENIX 5th International Middleware Conference*, Toronto, 2004.
7. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *International Conference on Distributed Computing Systems*, july 2002.
8. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University, 2002.

9. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Intern. Symp. on High-Performance Distributed Computing*, Seattle, USA, 2003.
10. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.
11. U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Intern. Conf. on Very Large Databases*, Cairo, Egypt, 2000.
12. The gnutella developer forum, http://rfc-gnutella.sourceforge.net, 2004.
13. L. Gravano and H. García-Molina. Generalizing GlOSS to vector-space databases and broker hierarchies. In *International Conference on Very Large Databases, VLDB*, pages 78–89, 1995.
14. P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004*, volume 3298 of *LNCS*, pages 122–136. Springer, NOV 2004.
15. P. Haase et al. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proc. of the 3rd International Semantic Web Conference, Japan*. Springer, 2004.
16. S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. pages 606–615, 1987.
17. J. Kleinberg. Navigation in a small world. *Nature*, 406, 2000.
18. Y. Li, Z. Bandar, and D. McLean. An Approach for messuring semantic similarity between words using semantic multiple information sources. In *IEEE Transactions on Knowledge and Data Engineering*, volume 15, 2003.
19. A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.
20. S. Milgram. The small world problem. *Psychology Today*, 67(1), 1967.
21. S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. *Multimedia Systems*, 9(2), 2003.
22. K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest Based Locality in Peer-to-Peer System. In *Infocom*. IEEE, 2003.
23. C. Tempich, S. Staab, and A. Wranik. REMINDIN:Semantic Query Routing in Peer-to-Peer Networks based on Social Metaphers. In *Proceedings of the 13th WWW Conference New York*. ACM, 2004.
24. C. Tempich and R. Volz. Towards a benchmark for Semantic Web reasoners - an analysis of the DAML ontology library. In Y. Sure, editor, *Proceedings of Evaluation of Ontology-based Tools (EON2003) at 2nd International Semantic Web Conference (ISWC 2003)*, pages 4–15, OCT 2003.

Ontology–Based Representation and Query of Colour Descriptions from Botanical Documents

Shenghui Wang and Jeff Z. Pan(*)

School of Computer Science, University of Manchester,UK
{wangs, pan}@cs.man.ac.uk

Abstract. A proper representation of the semantics of colour descriptions is necessary when colour information needs to be processed semantically, such as in the area of information integration. Semantics–based methods can help information from different sources to be integrated more easily and make final results more accurate and understandable among different sources. This paper introduces an ontology–based representation of the semantics of colour descriptions. By using a quantitative colour model and a Description Logic (DL) with datatype support, the semantics of a single colour term can be represented. Multiple such terms are then combined to generate the semantics of a complex colour description, which is interpreted by using morpho–syntactic rules derived from the analysis of colour descriptions in botanical documents. A colour reasoner, interacting with the FaCT-DG DL reasoner, can support colour–related queries and give domain–oriented results.

1 Introduction

Ontologies are currently perceived as an appropriate modelling structure for representing such knowledge as taxonomic knowledge in botany or zoology. Ideally, an ontology captures a shared understanding of certain aspects of a domain. More specifically, an ontology provides a common *vocabulary*, including important concepts, properties and their definitions, and *constraints* regarding the intended meaning of the vocabulary, sometimes referred to as background assumptions. In this paper, we use a multi–parameter ontology to capture the semantics of colour descriptions from botanical documents and to represent them in a plant ontology using the OWL-Eu [1] ontology language. OWL-Eu is an ontology language which extends the W3C standard OWL DL [2] with customised datatypes. A colour reasoner is proposed that interacts with the FaCT-DG DL reasoner [3] to answer colour–related queries which are useful in botanical practice.

Colours play an important role in the identification of plant species. A complete list of species containing those plants that have flowers of the requested colour, can be very helpful to botanists in identifying a plant sample in nature. Colour descriptions of the same species are found in many different *floras*,[1] and

(*) This work is partially supported by the FP6 Network of Excellence EU project Knowledge Web (IST-2004-507842).

[1] A flora is a treatise describing the plants of a region or time.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1279–1295, 2005.

are therefore treated as parallel sources. For instance, the species *Origanum vulgare* (Marjoram) has at least four colour descriptions of its flowers from four floras:

- "violet–purple", in *Flora of the British Isles* [4],
- "white or purplish–red", in *Flora Europaea* [5],
- "purple–red to pale pink", in *Gray's Manual of Botany* [6],
- "reddish–purple, rarely white", in *New Flora of the British Isles* [7].

It has been demonstrated in [8] that extracting and integrating parallel information from different sources can produce more accurate and complete results. Some current projects [9, 10] attempt to store knowledge extracted from natural language documents in electronic forms. These projects generally allow keyword-based queries but do not support a formal representation of the semantics.

In this paper, we present an ontology-based approach [11, 12] to tackle this problem. Information extraction techniques are used to get proper colour descriptions from botanical documents. In order to decompose the semantics of colour descriptions, we propose a quantitative model based on the HSL (Hue Saturation Lightness) colour model. By using a parser based on our BNF syntax, we can quantify complex colour descriptions more precisely; for instance, we support adjective modifiers, ranges, conjunction or disjunction relations indicated by natural language constructions. Based on the semantics of colour descriptions, we can generate an ontology to model such complex colour information in our project. Such an ontology provides a foundation for information integration and domain-oriented query answering.

The semantics of information are important for both extraction and integration. However, existing information integration systems [13] do not process information directly based on their semantics. One obvious reason for this is that there is a deep gap between linguistic and logical semantics [14]. As we will show later in the paper, customised datatypes are crucial to capture the semantics of the quantitative model. This suggests that we can not use the Semantic Web standard ontology language OWL DL for our purpose, since OWL DL does not support customised datatypes. Instead, we use the OWL-Eu ontology language, which is a datatype extension of OWL DL. In our ontology, we can represent complex colour descriptions as OWL-Eu class descriptions. Therefore, we can make use of the subsumption checking reasoning service provided by the FaCT-DG DL reasoner to check if a colour description is more general than another one. A colour reasoner, interacting with the FaCT-DG reasoner, is implemented to answer colour-related species identification queries and return more useful results for practical botanical purposes.

The rest of the paper is structured as follows. Section 2 introduces some technical background knowledge of multi–parameter colour models and the OWL–Eu ontology language. Section 3 provides the morpho–syntactic rules of building complex colour descriptions and their influences in generating final semantics. Section 4 describes how the semantics of colour descriptions are represented in the OWL-Eu language. Section 5 introduces a domain–oriented usage of such a

multi–parameter representation, i.e. colour–related species query. Section 6 gives primary experimental results of representation and several queries based on it. Finally, Section 8 concludes this paper and discusses some of our future work.

2 Technical Background

2.1 The Colour Model

Several colour representations using a multi–parameter colour space (CIE XYZ, L*a*b*, L*u*v*, RGB, CMYK, YIQ, HSV, HSL, etc.) are used in computer graphics and image processing. Colours are quantified as points (or regions) in those spaces. Linguistically naming the physically represented colours has been thoroughly investigated [15].

The psychologically based HSL (Hue Saturation Lightness) model is more accurate than machine–oriented colour models, such as the RGB (Red Green Blue) model, in colour notation, and is second only to natural language [16]. The HSL model was therefore chosen to model basic colour terms parametrically. Its colour space is a double cone, as shown in Figure 1.

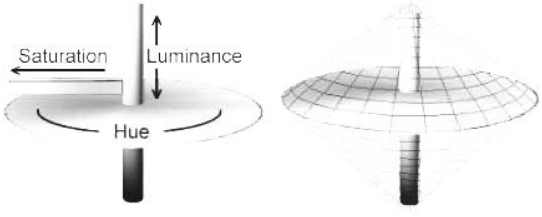


Fig. 1. HSL Colour Model

In the HSL model, a colour is represented by the following three parameters:

- *Hue* is a measure of the colour tint. In fact, it is a circle ranging from 0 (red) to 100 (red again), passing through 16 (yellow), 33 (green), 50 (cyan), 66 (blue) and 83 (magenta).
- *Saturation* is a measure of the amount of colour present. A saturation of 0 is a total absence of colour (i.e. black, grey or white), a saturation of 100 is a pure colour tint.
- *Lightness* (also Luminance or Luminosity) is the brightness of a colour. A lightness of 0 is black, and 100 is white, between 0 and 100 are shades of grey. A lightness of 50 is used for generating a pure colour.

Each basic colour term corresponds to a small space in the double cone whose centre is the particular point representing its HSL value, that is, instead of a point, a colour term is represented by a cuboid space, defined by a range

triplet (hueRange, saturationRange, lightnessRange). For instance, "purple" is normally defined as the HSL point (83, 50, 25), but is represented in our ontology as the region (78–88, 45–55, 20–30), adding a certain range to each parameter.[2]

2.2 OWL DL and Its Datatype Extension OWL-Eu

The OWL Web Ontology Language [2] is a W3C recommendation for expressing ontologies in the Semantic Web. OWL DL is a key sub–language of OWL. Datatype support [18, 19] is one of the most useful features that OWL is expected to provide, and has brought extensive discussions in the RDF–Logic mailing list [20] and Semantic Web Best Practices mailing list [21]. Although OWL provides considerable expressive power to the Semantic Web, the OWL datatype formalism (or simply *OWL datatyping*) is much too weak for many applications. In particular, OWL datatyping does not provide a general framework for customised datatypes, such as XML Schema user–defined datatypes.

To solve the problem, Pan and Horrocks [1] proposed OWL-Eu, a small but necessary extension to OWL DL. OWL-Eu supports customised datatypes through unary datatype expressions (or simply datatype expressions) based on unary datatype groups. OWL-Eu extends OWL DL by extending datatype expressions with OWL data ranges.[3] Let $\mathcal{G}$ be a unary datatype group. The set of $\mathcal{G}$*–datatype expressions*, $\mathbf{Dexp}(\mathcal{G})$, is inductively defined in abstract syntax as follows [1]:

1. *atomic expressions*: if u is a datatype URIref, then $u \in \mathbf{Dexp}(\mathcal{G})$;
2. *relativised negated expressions*: if u is a datatype URIref, then $\mathtt{not}(u) \in \mathbf{Dexp}(\mathcal{G})$;
3. *enumerated datatypes*: if $l_1, \ldots, l_n$ are literals, then $\mathtt{oneOf}(l_1, \ldots, l_n) \in \mathbf{Dexp}(\mathcal{G})$;
4. *conjunctive expressions*: if $\{E_1, \ldots, E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\mathtt{and}(E_1, \ldots, E_n) \in \mathbf{Dexp}(\mathcal{G})$;
5. *disjunctive expressions*: if $\{E_1, \ldots, E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\mathtt{or}(E_1, \ldots, E_n) \in \mathbf{Dexp}(\mathcal{G})$.

Uniform Resource Identifiers (URIs) are short strings that identify Web resources [22]. A *URI reference* (or URIref) is a URI, together with an optional fragment identifier at the end. In OWL, URIrefs are used as symbols for classes, properties and datatypes, etc.

For example, the following XML Schema user–defined datatype

```
<simpleType name = "HueRange">
  <restriction base = "xsd:integer">
    <minInclusive value = "0"/>
    <maxInclusive value = "100"/>
  </restriction>
</simpleType>
```

can be represented by the following conjunctive datatype expression:

$$\mathtt{and}(\text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThanOrEqualTo100}),$$

[2] Referring to the NBS/ISCC Color System [17], giving a 100-point hue scale, each major hue places at the middle of its 10–point spread, or at division 5.

[3] This is the *only* extension OWL-Eu brings to OWL DL.

Table 1. Colour description patterns and their relative frequencies of occurrence, where X, Y and Z each represent a single colour term or a simpler colour phrase, A is a degree adjective and P is a probability adverb

Colour description pattern	Frequency of occurrence	Example
X	25.5%	"orange"
A X	36.5%	"pale blue"
X to Y (to Z...)	25.9%	"white to pink to red to purple"
X–Y	19.9%	"rose–pink"
X+ish(–)Y	13.2%	"reddish–purple"
X(, Y) or Z	6.5%	"white or violet
X(, Y), P Z	6.4%	"reddish–purple, rarely white"
X/Y	4.6%	"pink/white"
X, Y	2.8%	"lavender, white–pink
X(, Y), and Z	2.3%	"white and green"

where xsdx:integerLessThanOrEqualTo100 is the URIrefs for the user–defined datatype $\leq_{100}$. Readers are referred to [23] for more details of OWL abstract syntax and semantics.

Similarly to an OWL DL ontology, an OWL-Eu ontology typically contains a set of class axioms, property axioms and individual axioms. FaCT-DG, a datatype group extension of the FaCT DL reasoner, supports TBox reasoning on OWL-Eu ontologies without nominals.

3 NL Processing

From a closer observation of the real data in floras, we find that colour descriptions are mostly compound phrases so that they can cover the variations of plant individuals in the field, such as the example shown in Section 1. Complex colour descriptions are built from multiple basic colour terms by certain morpho–syntactic rules. In order to be represented correctly, a complex colour description has to be analysed using the same rules.

We carry out a morpho–syntactic analysis on 227 colour descriptions of 170 species from five floras.[4] Different types of phrases and their relative frequencies of occurrence in the data set are summarised in Table 1 (page 1283). Table 2 (page 1284) gives the corresponding BNF syntax of these phrases for colour description. As shown in Table 1, most patterns describe colour ranges that are built from several atomic colour phrases, such as "blue", "blue–purple" or "bright yellow".

There are two steps in our text processing. Firstly, we construct the following atomic colour phrases as basic colour spaces.

[4] They are *Flora of the British Isles* [4], *Flora Europaea* [5], *The New Britton and Brown Illustrated Flora of the Northeastern United States and Adjacent Canada* [6], *New Flora of the British Isles* [7] and *Gray's Manual of Botany* [24].

Table 2. BNF syntax of colour descriptions

$< Cterm >::= red \mid yellow \mid green \mid \ldots$
$< Dmodifier >::= strong \mid pale \mid bright \mid deep \mid dull \mid light \mid dark \mid \ldots$
$< Pmodifier >::= usually \mid often \mid sometimes \mid occasionally \mid rarely \mid never \mid \ldots$
$< Cphrase >::= < Cterm >$
$\mid < Cterm > [ish][-\mid\ ] < Cterm >$
$\mid < Cphrase > - < Cphrase >$
$\mid < Dmodifier > < Cterm >$
$< Cdescription >::= < Cphrase >$
$\mid < Cphrase > \{\ to\ < Cphrase >\}$
$\mid < Cphrase >,\ < Cphrase >$
$\mid < Cphrase > / < Cphrase >$
$\mid < Cphrase > \{, < Cphrase >\}\ or\ < Cphrase >$
$\mid < Cphrase > \{, < Cphrase >\}\ and\ < Cphrase >$
$\mid < Cphrase > \{, < Cphrase >\},\ < Pmodifier > < Cphrase >$

Table 3. Meanings of adjective modifiers and their corresponding operations on a colour space

Adjective	Meaning[a]	Operation[b]
strong	high in chroma	satRange + 20
pale	deficient in chroma	satRange - 20, ligRange + 20
bright	of high saturation or brilliance	satRange + 20, ligRange + 20
deep	high in saturation and low in lightness	satRange + 20, ligRange - 20
dull	low in saturation and low in lightness	satRange - 20, ligRange - 20
light	medium in saturation and high in lightness	satRange - 20, ligRange + 20
dark	of low or very low lightness	ligRange - 20

[a] referring to Merriam–Webster online dictionary.

[b] Referring to the specifications from Colour Naming System (CNS) [25], saturation and lightness are each divided into 5 levels, which causes a range/ranges to change by 20 (100/5).

X : This is a single colour space, i.e. (hueRange, satRange, ligRange).

A X : We need to modify the space of X according to the meaning of A, as shown in Table 3 (page 1284). For example, "light blue" is represented as (61–71, 70–80, 65–75) where "blue" is (61–71, 90–100, 45–55).

X–Y : This represents an intermediate colour between the two colours X and Y [25]. For example, "blue–purple" is generated from the halfway colour between "blue" (66, 100, 50) and "purple" (83, 50, 25), that is, the colour with HSL value of (75, 75, 38), the hue, saturation and lightness of which are calculated by the following formulae (similar calculation for saturation and lightness) and finally represented by the range triple (70–80, 70–80, 33–43).

$$Hue_{X-Y} = \frac{Hue_X + Hue_Y}{2}$$

Xish–Y : This denotes a quarterway value between the two colours [25], closer to the latter colour term. For instance, "reddish–purple" means it is basically purple (83, 50, 25) but reflecting a quarterway deviation to red (100, 100, 50), so the final hue range for "reddish–purple" is (87, 63, 34) calculated by the following formula (similar formulas for saturation and lightness), which is finally (82–92, 58–68, 29–39).

$$Hue_{X_{ish}-Y} = Hue_Y + \frac{Hue_X - Hue_Y}{4}$$

Secondly, we build up combined colour spaces based on basic ones. Specifically, combined colour spaces are built up by a colour reasoner, according to the following morpho–syntactic rules:

1. If basic colour terms are connected by one or more "to"s, the final colour space should be the whole range from the first colour to the last one. For instance, if light blue is (66, 100, 70) and purple is (83, 50, 25), "light blue to purple" should be the whole range (66–83, 50–100, 25–70), which contains any colour in between.
   Note that special care is needed for ranges starting or ending with a grey colour, such as "white to purple". In the HSL model, colours ranging from white, through different levels of grey, to black have no hue and saturation values. For instance, the HSL value of "white" is (0, 0, 100), while "red" also has a hue value of 0. A special rule for building such kind of ranges has to be followed; that is, a range from colour A (0, 0, l_a) to colour B (h_b, s_b, l_b) should be ($\overline{h_b - 5}$–$\overline{h_b + 5}$, 0–s_b, l_a–l_b). For example, "white to purple" should be represented by the triple (78–88, 0–50, 25–100).
2. If basic colour terms are connected by any of these symbols: "or", "and", comma (",") or slash ("/"), they are treated as separate colour spaces; that is, they are disjoint from each other. For instance, "white, lilac or yellow" means that the colour of this flower could be either white or lilac or yellow, not a colour in between.
   Notice that "and" is treated as a disjunction symbol because, in floras, it normally means several colours can be found in the same species, instead of indicating a range by normal logical conjunction. For instance, flowers of species *Rumex crispus* (Curled Dock) are described as "red and green", which means flowers that either red or green may occur in the same plant, but it does not mean that one flower is both red and green.

By using a parser based on our BNF syntax, we can generate an OWL-Eu ontology to model complex colour information.

4 Representation of Colour Descriptions in OWL–Eu

Based on the morpho–syntactic rules introduced in Section 3, we can decompose the semantics of colour descriptions into several quantifiable components, which

can be represented as DL datatype expressions. In this section, we will show how to use the OWL-Eu ontology language to represent the semantics of a colour description.

The fragment of our plant ontology $\mathcal{O}_C$ contains Colour as a primitive class. Important primitive classes in $\mathcal{O}_C$ include

```
Class(Species), Class(Flower), Class(Colour);
```

important object properties in $\mathcal{O}_C$ include

```
ObjectProperty(hasPart), ObjectProperty(hasColour);
```

important datatype properties in $\mathcal{O}_C$ include

```
DatatypeProperty(hasHue Functional
 range(and(xsd:nonNegativeInteger, xsdx:integerLessThanOrEqualTo100))),
DatatypeProperty(hasSaturation Functional
 range(and(xsd:nonNegativeInteger, xsdx:integerLessThanOrEqualTo100))),
DatatypeProperty(hasLightness Functional
 range(and(xsd:nonNegativeInteger, xsdx:integerLessThanOrEqualTo100))),
```

which are all functional properties. A *functional* datatype property relates an object with at most one data value. Note that the datatype expression

$$\mathsf{and}(\text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThanOrEqualTo100})$$

is used as the range of the above datatype properties.

Based on the above primitive classes and properties, we can define specific colours, such as Purple, as OWL-Eu defined classes (indicated by the keyword "complete") .

```
Class(Purple complete Colour
  restriction(hasHue someValuesFrom
                                (and(xsdx:integerGreaterThanOrEqualTo78,
                                 xsdx:integerLessThanOrEqualTo88)))
  restriction(hasSaturation someValuesFrom
                                (and(xsdx:integerGreaterThanOrEqualTo47,
                                 xsdx:integerLessThanOrEqualTo52)))
  restriction(hasLightness someValuesFrom
                                (and(xsdx:integerGreaterThanOrEqualTo20,
                                 xsdx:integerLessThanOrEqualTo30))))
```

In the above class definition, datatype expressions are used to restrict the values of the datatype properties *hasHue*, *hasSaturation* and *hasLightness*. Note that not only colour terms but also complex colour descriptions can be represented in OWL-Eu classes, as long as they can be transformed into proper colour subspaces with constraints on their hue, saturation and lightness.

As colour descriptions are represented by OWL-Eu classes, we can use the subsumption checking service provided by the FaCT-DG reasoner to check if one colour description is more general than another. Namely, if ColourA is subsumed

by ColourB, we say that ColourB is more general than ColourA. The formal representation of colour descriptions makes it possible to express a query about a range of colours, such as to retrieve all species which have "bright rose–pink" or "light blue to purple" flowers, with the help of the FaCT-DG DL reasoner.

5 Domain-Oriented Queries

The flower colour of an individual plant is an important distinguishing feature for identifying which species it belongs to. The species identification that botanists are interested in can be written as a query: "Given a certain colour, tell me all the possible species whose flowers have such a colour." We would like to point out that, from a botanical point of view, one has to take the variations between individuals in nature into account. In other words, botanists rarely use colour as a strict criterion. It is more appropriate to answer such species identification queries in an fuzzy manner, that is, returning a list which contains all species that could match the query. This kind of query, which is particularly suitable for domain interests, is called domain-oriented queries.

We can answer species identification queries based on subsumption queries that are supported by the FaCT-DG DL reasoner. For example, if our plant ontology contains the following class axioms:

```
Class(SpeciesA restriction(hasPart someValueFrom(FlowerA)))
Class(FlowerA restriction(hasColour someValueFrom(ColourA)))
Class(SpeciesB restriction(hasPart someValueFrom(FlowerB))
Class(FlowerB restriction(hasColour someValueFrom(ColourB)))
```

and if from the definitions of ColourA and ColourB we can conclude that ColourA is subsumed by ColourB, when we ask our DL reasoner whether the above ontology entails that SpeciesA is subsumed by SpeciesB, the reasoner will return "yes". By using this kind of subsumption query, we can, for example, conclude that a species having "golden–yellow" flowers is subsumed by a more general species which has "yellow" flowers, which again is subsumed by another species which has "red to orange to yellow" flowers. Therefore, if one asks "Which species might have yellow flowers?", our colour reasoner will return all these three species.

For species identification, this hierarchical subsumption matching is very useful for shortening the possible species list. After classification reasoning, we have already had three different levels of matchings: **Exact** matching ($\mathsf{Class}_{\mathsf{RealSpecies}} \equiv \mathsf{Class}_{\mathsf{QuerySpecies}}$), **PlugIn** matching ($\mathsf{Class}_{\mathsf{RealSpecies}} \sqsubseteq \mathsf{Class}_{\mathsf{QuerySpecies}}$) and **Subsume** matching ($\mathsf{Class}_{\mathsf{RealSpecies}} \sqsupseteq \mathsf{Class}_{\mathsf{QuerySpecies}}$) [26, 3]. Actually there is another possible species list, which is not covered by the above three kinds of matchings, that is, **Intersecting** matching ($\neg(\mathsf{Class}_{\mathsf{RealSpecies}} \sqcap \mathsf{Class}_{\mathsf{QuerySpecies}} \sqsubseteq \bot)$). For example, if a species has "greenish–yellow" flowers, it would also be possible to find in the field an individual which has "yellow" flowers. Although this latter list has a lower probability to contain the correct answers, it is still helpful from botanical point of view.

We have implemented a colour reasoner to reduce our domain problems into standard Description Logics reasoning problems. In fact, it interacts with the

Table 4. Query results of species having "yellow" flowers (partial)

Species	Flower colour	Matching type
Amsinckia menziessi	yellow	Exact matching
Ranunculus acris	golden–yellow	PlugIn matching
Barbarea vulgaris	yellow to pale yellow	Subsume matching
Eucalyptus globulus	creamy–white to yellow	Subsume matching
Anaphalis margaritacea	white, yellow to red	Subsume matching
Castilleja wightii	yellow–orange–apricot–red	Subsume matching
Tropaeolum majus	yellow to orange to red	Subsume matching
Myrica californica	green to red to brown	Subsume matching
Trillium chloropetalum	dark red to greenish–white	Subsume matching
Artemesia californica	whiteish–yellow	Intersection matching
Rumex acetosella	reddish–yellow	Intersection matching
Rhodiola sherriffii	greenish–yellow	Intersection matching
Lasthenia californica	yellow–orange	Intersection matching
Mimulas aurantiacus	orange	Intersection matching
Artemesia douglasiana	whiteish–green to whiteish–yellow	Intersection matching
Eschscholzia californica	deep orange to pale yellow	Intersection matching

FaCT-DG reasoner, in order to answer domain-oriented queries. First of all, the colour in a query is represented by an OWL-Eu class Q with datatype constraints about its hue, saturation and lightness. Secondly, the colour reasoner calculates the complete set of colours $complete_Q$ which satisfies the above four levels of matching. Specifically, $complete_Q$ consists of the following four sets.

- $equiv_Q$: all elements are equivalent to the class Q, such as "yellow";
- sub_Q: all elements are subsumed by the class Q, such as "golden–yellow";
- $super_Q$: all elements subsume the class Q, such as "yellow to orange to red";
- $intersection_Q$: all elements intersect with the class Q, such as "greenish–yellow".

Note that the first two contain answers with 100% confidence, while the latter two contain those with less confidence. Thirdly, in order to find all species that have flowers whose colour satisfies the query, the colour reasoner interacts with the Fact-DG reasoner to return those species which have flowers whose colour is contained in $complete_Q$ set.

6 Experiments and Discussions

In this section, we will present some of the experiments, in terms of species identification queries, we did with our plant ontology.

We chose 100 colour terms, which are commonly used in floras, as basic colour terms. For each basic term, we obtained its RGB value by referring to the X11 Colour Names,[5] converted it into the corresponding HSL value and finally defined it as ranges in hue, saturation and lightness (as described in Section 4).

[5] http://en.wikipedia.org/wiki/X11_Color_Names

Table 5. Query results of species having "light blue" flowers (partial)

Species	Flower colour	Matching type
Aster chilensis	light blue	Exact matching
Ceanothus thyrsiflorus	pale blue to bright blue	Subsume matching
Heliotropium curassavicum	white to bluish	Subsume matching
Linum bienne	pale blue to lavender	Subsume matching
Phacelia ramosissima	white, sometimes pale blue	Subsume matching
Viola adunca	light blue to purple	Subsume matching
Triteleia laxa	blue to violet	Intersection matching
Vinca major	light–to–dark blue–violet	Intersection matching
Dichelostemma congestum	pink to blue	Intersection matching

By using the method described in Section 3 on the same data set we used for linguistical analysis, each colour description was transformed into a small colour space. A simple plant ontology, mentioned in Section 4, was constructed using the OWL-Eu language. This ontology contains 170 species classes, each species has a flower part which has a colour property. The colour property is represented by a datatype expression. For example, species *Viola adunca* has "light blue to purple" flowers.

```
Class(Viola_adunca complete Species
  restriction(hasPart someValuesFrom(Viola_adunca_flower))),
Class(Viola_adunca_flower complete Flower
  restriction(hasColour someValuesFrom(Viola_adunca_flower_colour))),
Class(Viola_adunca_flower_colour complete Colour
  restriction(hasHue someValuesFrom
                              (and(xsdx:integerGreaterThanOrEqualTo66,
                               xsdx:integerLessThanOrEqualTo83)))
  restriction(hasSaturation someValuesFrom
                              (and(xsdx:integerGreaterThanOrEqualTo50,
                               xsdx:integerLessThanOrEqualTo100)))
  restriction(hasLightness someValuesFrom
                              (and(xsdx:integerGreaterThanOrEqualTo25,
                               xsdx:integerLessThanOrEqualTo70))))
```

In the species identification queries, we used colour descriptions with different levels of complexity (as shown in Table 1 on page 1283). Some of the results are presented in Tables 4, 5 and 6, in the order of complexity of colours: "yellow", "light blue", "light blue to purple".

From Table 4, all species which could have "yellow" flowers have been queried out. In the result, we can find the species whose flowers are explicitly described as "yellow" or some variation thereof. We also find that those species which have "orange", "green to red to brown" or "dark red to greenish–white" flowers are also returned by our colour reasoner, as the results of subsumption or intersection

Table 6. Query results of species having "light blue to purple" flowers (partial)

Species	Flower colour	Matching type
Viola adunca	light blue to purple	Exact matching
Aster chilensis	pale blue	PlugIn matching
Scoliopus bigelovii	purple	PlugIn matching
Linum bienne	pale blue to lavender	PlugIn matching
Myosotis latifolia	blue	PlugIn matching
Eriodictyon californicum	light purple/lavender	PlugIn matching
Ceanothus thyrsiflorus	pale blue to bright blue	PlugIn matching
Iris douglasiana	pale lavender to blue to purple	PlugIn matching
Delphinium variegatum	royal blue to purple	PlugIn matching
Verbena lasiostachys	blue–purple	PlugIn matching
Cirsium occidentale	red–purple	Intersection matching
Cirsium vulgare	reddish–purple	Intersection matching
Epilobium cilatum	white to pink to red to purple	Intersection matching
Heliotropium curassavicum	white to bluish	Intersection matching
Lupinus eximus	blue to purple, sometimes lavender	Intersection matching
Polygala californica	rose–pink/purple	Intersection matching
Solanum	white to white–lavender to pink/blue	Intersection matching
Vinca major	light–to–dark blue–violet	Intersection matching
Dichelostemma congestum	pink to blue	Intersection matching
Sisyrinchium bellum	strong blue–purple	Intersection matching
Stachys bullata	light purple to pink to white	Intersection matching
Triteleia laxa	blue to violet	Intersection matching

matching. This is consistent with their real meanings, because in our colour model "yellow" is contained by or intersects with them.

We can query in a specific manner, such as to find species which have "light blue" flowers but excluding those with "dark blue" flowers (see Table 5); or in a more general style, such as to query all species which could have flowers ranging from "light blue to purple" (see Table 6). All of these owe to our quantitative model which makes it possible to compare and reason with classes at a semantic level.

As stated in Section 5, the resulting list is from four different levels of matching, which gives a complete list for species identification. We can also specify to stop at certain levels of matching to get results with different confidences, such as only return those species which fully satisfy the query.

The semantics of a colour term or a complex colour description is decomposed and represented by a group of ranges in multiple numerical parameters, which is a small subspace in a multi–dimensional space. Numerical representation makes it easy to build ranges between colours, but a further observation shows that this is not as obvious as we thought. For example, there could be different ways of interpreting the meaning of "light blue *to* purple" (see Fig. 2):

- light blue to purple directly (area B),
- light blue to blue then to purple,
- light blue to light purple then to purple,
- the whole rectangle.

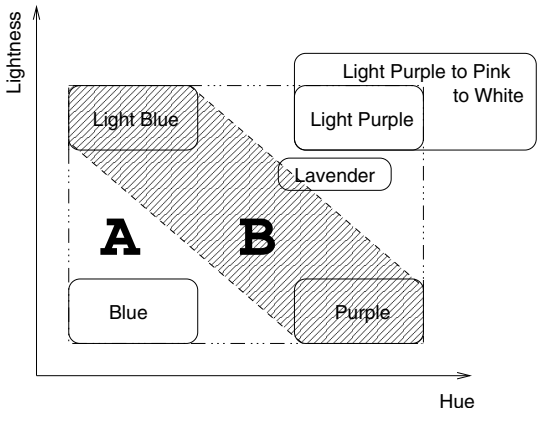


Fig. 2. Range between "light blue" and "purple"

In our experiment (see Table 6), we used the last option (the whole rectangle). We plan to extend our work and to allow the users to pick up one of the above options when they query with the keyword "to".

7 Related Work

Automatically integrating information from a variety of sources has become a necessary feature for many information systems [13]. Compared to structured or semi-structured data sources, information in natural language documents is more cumbersome to access [27]. Our work focuses mainly on parallel information extraction and integration from homogeneous monolingual (English) documents.

Information Extraction (IE) [28] is a common Natural Language Processing (NLP) technique which can extract information or knowledge from documents. Ontologies, containing various semantical expressions of domain knowledge, have recently been adopted in many IE systems [29, 30, 31]. Semantics embedded in ontologies can boost the performance of IE in terms of precision and recall [32]. Since they can be shared by different sources, ontologies also play an important role in the area of information integration [13, 33, 27]. Logic reasoning, as supported by ontology languages, is also introduced naturally into the extraction and integration process [34, 35, 32].

The work in this paper is similar to some research in computational linguistics, such as Lexical Decomposition [36], which attempts to break the meanings of words down to more basic categories. Instead of certain qualitative criteria, our method attempts to give precise semantics to the classes in an ontology by using multiple basic and quantifiable classes. This makes comparing and reasoning with classes easier.

The quantitative semantic model can produce more useful results for real domain purposes. Specifically, in botanical domain, many current plant databases

can only support keyword-based query, such as the ActKey [9] provided by the floras project,[6] ePIC project [10], etc. They rely heavily on the occurrence of keywords, which are normally very general and therefore less robust. As demonstrated in Section 6, our method can compute on their real semantics instead of pure keyword matching, which supports more flexible-styled queries and gets more useful results.

8 Conclusion and Outlook

This paper introduces an ontology–based approach to capture the semantics of colour descriptions and to support colour-related species identification queries. It turns out that, even in a limited domain, representing the semantics of colour descriptions is not a trivial problem. Based on a multi–parameter semantic model for colour descriptions and certain morpho–syntactic rules, we have implemented an NLP parser which translates complex colour descriptions into quantifiable logic representations. More importantly, a colour reasoner is implemented to carry out queries for real botanical applications by interacting with the FaCT-DG DL reasoner.

We have shown that our approach outperforms text–based approaches by providing more precise integration results. Firstly, our quantifiable model makes it possible to reason and query on a semantic level. Relations between colour descriptions are more tractable. For example, yellow is between red and green in terms of hue, lilac is lighter than purple although they have the same hue. Furthermore, based on the support of adjective modifiers and ranges, we can query in a detailed manner, such as "light blue", which excludes pure blue and dark blue. We can also query on a fuzzy manner, such as "light blue to purple", as required for particular domain purposes.

Interestingly, our work also provides a use case for the OWL-Eu ontology language. OWL-Eu extends OWL DL with user-defined datatypes, which are needed to represent the ranges for hue, saturation and lightness used in colour descriptions, not to mention the degree adjective described in Table 3.

Encouraged by the existing results, we plan to further extend our work on ontology-based species identification queries. Firstly, as suggested in Section 6, a future version of our colour reasoner should provide several options so as to allow users to decide their intended meaning of the 'to" keyword. Technically, this requires the use of not only unary but also n-ary datatype expressions as constrains on datatype properties *hasHue*, *hasSaturation* and *hasLightness*. To capture these constraints, we need to use the OWL-E [37, 3] ontology language, which is the n-ary extension of OWL-Eu.

Taking advantage of the quantitative representation, similarities between different descriptions from different authors can be measured as the distances between their corresponding colour subspaces. As long as an appropriate distance measurement is chosen, such distances can easily tell us how different two descriptions are and to what extend they overlap with each other. How to define

[6] http://www.efloras.org/

the distance and how to use it for integration is an interesting work to explore in the future.

Another future work is to represent the probabilistic information in the ontology. There are many descriptions with adverbs of quantification, such as "sometimes", "rarely", "often", etc., which also indicate the probability of certain colours. Because current ontology languages do not support the annotation of classes with probabilities, the probability aspect is ignored in the text processing which affects the precision of integration. However, there are several attempts to extend DL languages with fuzzy expressions [38, 39, 40], which, in the future, may be used to enable our logic representation to capture more of the original semantics implied by natural language.

Furthermore, our approach is applicable in other similar areas, such as the representation of leaf shapes, which is another key feature of identifying species. We have started to experiment with a quantitative model generated by a Super-Shape formula [41]. We expect that our method can also produce similar results in this case.

References

1. Pan, J.Z., Horrocks, I.: OWL-Eu: Adding Customised Datatypes into OWL. In: Proc. of Second European Semantic Web Conference (ESWC 2005). (2005)
2. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., eds., L.A.S.: OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/ (2004)
3. Pan, J.Z.: Description Logics: Reasoning Support for the Semantic Web. PhD thesis, School of Computer Science, The University of Manchester (2004)
4. Clapham, A., Tutin, T., Moore., D.: Flora of the British Isles. Cambridge University Press (1987)
5. Tutin, T.G., Heywood, V.H., Burges, N.A., Valentine, D.H., Moore(eds), D.M.: Flora Europaea. Cambridge University Press (1993)
6. Gleason, H.: The New Britton and Brown Illustrated Flora of the Northeastern United States and Adjacent Canada. Hafner Publishing Company, New York (1963)
7. Stace, C.: New Flora of the British Isles. Cambridge University Press (1997)
8. Wood, M.M., Lydon, S.J., Tablan, V., Maynard, D., Cunningham, H.: Using parallel texts to improve recall in ie. In: Proceedings of Recent Advances in Natural Language Processing (RANLP-2003), Borovetz, Bulgaria (2003) 505–512
9. : Actkey. (Web Page `http://flora.huh.harvard.edu:8080/actkey`)
10. Royal Botanic Gardens, K.: electronic plant information centre. (Published on the Internet `http://www.kew.org/epic/`)
11. Wood, M., Lydon, S., Tablan, V., Maynard, D., Cunningham, H.: Populating a database from parallel texts using ontology-based information extraction. In Meziane, F., Métais, E., eds.: Proceedings of Natural Language Processing and Information Systems, 9th International Conference on Applications of Natural Languages to Information Systems, Springer (2004) 254–264
12. Wood, M., Wang, S.: Motivation for "ontology" in parallel-text information extraction. In: Proceedings of ECAI-2004 Workshop on Ontology Learning and Population (ECAI-OLP), Poster, Valencia, Spain (2004)

13. Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Huebner, S.: Ontology-based integration of information - a survey of existing approaches. In: Proceedings of the IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA (2001) 108–117
14. Dik, S.C.: Coordination: Its implications for the theory of general linguistics. North-Holland, Amsterdam (1968)
15. Lammens, J.M.: A computational model of color perception and color naming. Ph.D. thesis, State University of New York (1994)
16. Berk, T., Brownston, L., Kaufman, A.: A human factors study of color notation systems for computer graphics. Communications of the ACM **25** (1982) 547–550
17. U.S. Department of Commerce, National Bureau of Standards: Color: Universal Language and Dictionary of Names. NBS Special Publication 440. U.S. Government Printing Office, Washington D.C. (1976) (S.D. Catalog No. C13.10:440).
18. Pan, J.Z., Horrocks, I.: Extending Datatype Support in Web Ontology Reasoning. In: Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002). (2002) 1067–1081
19. Pan, J.Z., Horrocks, I.: Web Ontology Reasoning with Datatype Groups. In: Proc. of the 2nd International Semantic Web Conference (ISWC2003). (2003)
20. : http://lists.w3.org/archives/public/www-rdf-logic/. W3C Mailing List (starts from 2001)
21. : http://lists.w3.org/archives/public/public-swbp-wg/. W3C Mailing List (starts from 2004)
22. Group, J.W.U.P.I.: URIs, URLs, and URNs: Clarifications and Recommendations 1.0. URL `http://www.w3.org/TR/uri-clarification/` (2001) W3C Note.
23. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C (2004) W3C Recommendation.
24. Fernald, M.: Gray's Manual of Botany. American Book Company, New York (1950)
25. Berk, T., Brownston, L., Kaufman, A.: A new color-naming system for graphics languages. IEEE Computer Graphics and Applications **2** (1982) 37–44
26. Li, L., Horrocks, I.: A Software Framework For Matchmaking Based on Semantic Web Technology. In: Proc. of the Twelfth International World Wide Web Conference (WWW 2003), ACM (2003) 331–339
27. Williams, D., Poulovassilis, A.: Combining data integration with natural language technology for the semantic web. In: Proc. Workshop on Human Language Technology for the Semantic Web and Web Services, at ISWC'03. (2003)
28. Gaizauskas, R., Wilks, Y.: Information Extraction: Beyond Document Retrieval. In: Computational Linguistics and Chinese Language Processing. Number 2 (1998) 17–60
29. Embley, D., Campbell, D., Liddle, S., Smith, R.: Ontology-based extraction and structuring of information from data-rich unstructured documents. In: Proceedings of International Conference On Information And Knowledge Management, 7, Bethesda, Maryland, USA. (1998)
30. Maedche, A., Neumann, G., Staab, S.: Bootstrapping an ontology-based information extraction system. studies in fuzziness and soft computing. In Szczepaniak, P., Segovia, J., Kacprzyk, J., Zadeh, L.A., eds.: Intelligent Exploration of the Web. Springer, Berlin (2002)
31. Alani, H., Kim, S., Millard, D.E., Weal, M.J., Hall, W., Lewis, P.H., Shadbolt, N.R.: Automatic ontology-based knowledge extraction from web documents. IEEE Intelligent Systems (2003) 14–21

32. Ferrucci, D., Lally, A.: UIMA: an architectural approach to unstructured information processing in the corporate research environment. Journal of Natural Language Engineering **10** (2004) 327–348
33. Goble, C., Stevens, R., Ng, G., Bechhofer, S., Paton, N., Baker, P., Peim, M., Brass, A.: Transparent access to multiple bioinformatics information sources. IBM Systems Journal Special issue on deep computing for the life sciences **40** (2001) 532 – 552
34. Calvanese, D., Giuseppe, D.G., Lenzerini, M.: Description logics for information integration. In Kakas, A., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski. Volume 2408 of Lecture Notes in Computer Science. Springer (2002) 41–60
35. Maier, A., Schnurr, H.P., Sure, Y.: Ontology-based information integration in the automotive industry. In: Proceedings of the 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA, Springer (2003) 897–912 None.
36. Dowty, D.R.: Word Meaning and Montague Grammar. D. Reidel Publishing Co., Dordrecht, Holland (1979)
37. Pan, J.Z.: Reasoning Support for OWL-E (Extended Abstract). In: Proc. of Doctoral Programme in the 2004 International Joint Conference of Automated Reasoning (IJCAR2004). (2004)
38. Tresp, C., Molitor, R.: A description logic for vague knowledge. In: Proceedings of the 13th biennial European Conference on Artificial Intelligence (ECAI'98), Brighton, UK, J. Wiley and Sons (1998) 361–365
39. Straccia, U.: Transforming fuzzy description logics into classical description logics. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04). Number 3229 in Lecture Notes in Computer Science, Lisbon, Portugal, Springer Verlag (2004) 385–399
40. Giorgos Stoilos, Giorgos Stamou, V.T.J.Z.P., Horrock, I.: A Fuzzy Description Logic for Multimedia Knowledge Representation. In: Proc. of the International Workshop on Multimedia and the Semantic Web. (2005) To appear.
41. Gielis, J.: A generic geometric transformation that unifies a wide range of natural and abstract shapes. American Journal of Botany **90** (2003) 333–338

Creating Ontologies for Content Representation—The **OntoSeed** Suite

Elena Paslaru Bontas[1], David Schlangen[2], and Thomas Schrader[3]

[1] Freie Universität Berlin, Institut für Informatik,
AG Netzbasierte Informationssysteme, Takustr. 9, 14195 Berlin, Germany
paslaru@inf.fu-berlin.de
[2] Universität Potsdam, Institut für Linguistik,
Angewandte Computerlinguistik, P.O. Box 601553, 14415 Potsdam, Germany
das@ling.uni-potsdam.de
[3] Institute for Pathology Charitè, Rudolf-Virchow-Haus,
Schumannstr. 20-21, D-10117 Berlin, Germany
thomas.schrader@charite.de

Abstract. Due to the inherent difficulties associated with manual ontology building, knowledge acquisition and reuse are often seen as methods that can make this tedious process easier. In this paper we present an NLP-based method to aid ontology design in a specific setting, namely that of semantic annotation of text. The method uses the World Wide Web in its analysis of the domain-specific documents, eliminating the need for linguistic knowledge and resources, and suggests ways to specify domain ontologies in a "linguistics-friendly" format in order to improve further ontology-based natural language processing tasks such as semantic annotation. We evaluate the method on a corpora in a real-world setting in the medical domain and compare the costs and the benefits of the NLP-based ontology engineering approach against a similar reuse-oriented experiment.

1 Introduction

Ontologies are widely recognized as a key technology to realize the vision of the Semantic Web and Semantic Web applications. In this context, ontology engineering is rapidly becoming a mature discipline which has produced various tools and methodologies for building and managing ontologies. However, even with a clearly defined engineering methodology, building a large ontology remains a challenging, time-consuming and error-prone task, since it forces ontology builders to conceptualize their expert knowledge *explicitly* and to *re-organize* it in typical ontological categories such as concepts, properties and axioms. For this reason, knowledge acquisition and reuse are often seen as ways to make this tedious process more efficient: though both methods cannot *currently* be used to *automatically* generate a domain ontology satisfying a specific set of requirements, they can be used to *guide* or *accelerate* the modeling process.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1296–1313, 2005.

Natural language processing techniques have proven to be particularly useful for these purposes [3, 8, 6, 13, 18, 24]. However, existing systems are still knowledge or resource intensive: they may not require much prior knowledge about the *domain* that is to be modeled, but they require *linguistic* knowledge or resources. In this paper we present a method to aid ontology building—within a certain setting, namely that of semantic annotation of texts—by using NLP techniques to analyze texts from the target domain. These techniques are comparably "knowledge-lean", since as a novel feature they make use of the WWW as a text collection against which the domain texts are compared during analysis; this makes them easy to employ even if no linguistic expertise is available and reduces the engineering costs since it avoids building an application-specific lexicon.

The techniques not only aid the ontology engineer in deciding which concepts to model, but they also suggest ways to specify the ontology in such a way that it fits ideally into further NLP-based processing steps, e.g. the extraction of information from domain-specific texts. Describing these specification issues and giving an example use case of ontologies thus created is the second aim of this paper.

The remainder of this paper is organized as follows: we motivate our approach and discuss previous work in Section 2. Section 3 gives details about our approach to using NLP to aid ontology design, which is evaluated from a technical and application perspective in Section 4. We close with a discussion of the results and an outline of future work in Section 5.

2 Motivation

2.1 Ontology Engineering

Due to the difficulties and costs involved in building an ontology from scratch, ontology engineering methodologies [9] often recommend to rely on available domain-related ontologies or unstructured data like text documents in conjunction with knowledge acquisition techniques, in order to simplify the domain analysis and the conceptualization of the domain knowledge.

In our own experience in a Semantic Web project in the medical domain (see [22, 30] for a longer discussion of this issue, and Section 4.2 below for the project setting), we found that just selecting and extracting relevant sub-ontologies (e.g. from a comprehensive medical ontology like UMLS[1]) was a very time-consuming process. Besides, this approach still resulted in a rather poor domain coverage as determined by the semantic annotation task. The ontology generated in this way could not be involved optimally in NLP-based processes and its acceptance w.r.t. its users was extremely low because of their difficulties in comprehending and evaluating it; this was our motivation to develop the techniques described here.

[1] `http://www.nlm.nih.gov/research/umls`

An alternative to reusing available ontologies or related knowledge sources (e.g. classifications, thesauri) is to employ text documents as an input for the conceptualization process. The most basic way to use texts is to extract *terms* from them, i.e. to determine words that denote domain-specific concepts (e.g. "lymphocyte" in a medical text) as opposed to general concepts (e.g. "telephone" in the same text). While this is often seen as a problem that is more or less solved ([7]; see [15] for a review of methods), the methods employed still rely on the presence of linguistic resources (e.g. corpora of non-domain-specific texts, lexicons; our approach differs in this respect, see below), and in any case are only the first step in a text-based analysis: ideally, the goal is to get a collection of terms that is further structured according to semantic relationships between the terms. There are several systems that go in this direction [3, 8, 6, 13, 18, 24], which however still require the availability of linguistic knowledge and resources, and moreover do not seem to work on all kinds of texts.[2] In general, there is a trade-off between the cost of getting or producing these resources and the simplification these methods offer. Hence our aim was a more modest, but at the present state of the art of the Semantic Web and in the given application scenario [22, 30] a more realistic one: to aid the ontology engineer as far as possible, requiring as little additional resources as possible. Before we come to a description of our approach, however, we briefly review the use of ontologies in NLP, and derive some requirements for "NLP-friendly" ontologies. These requirements are crucial for the development of high quality domain ontologies, which should combine a precise and expressive domain conceptualization with a feasible fitness of use (i.e. in our case, fitness of use in language-related tasks).

2.2 Ontologies in NLP

Ontologies have been used for a long time in many NLP applications, be that machine translation [20], text understanding [14], or dialogue systems (some recent examples are [12, 29]), and are of course central to information-extraction or Semantic Web-related NLP applications [2].

Despite all differences in purpose, a common requirement for an ontology to be considered "linguistics-friendly" (or "NLP-friendly") is that the path from lexical items (e.g. words) to ontology concepts should be as simple as possible.[3] On a more technical level, this requires that access to ontology concepts is given in a standardized form—if access is via names, then they should be in a predictable linguistic form. To give an example of this *not* being the case, the medical ontology UMLS contains concept names in the form "noun, adjective" (e.g. "Asthma, allergic") as well as "adjective noun" (e.g. "Diaphragmatic pleura"), and also concept names that are full phrases or even clauses (e.g. "Idiopathic fibrosing alveolitis chronic form"). Below we describe a method to avoid

[2] These methods rely on relational information implicitly encoded in the use of verbs; one of the domains we tested our approach is marked by a reduced, "telegram"-like text style with an almost complete absence of verbs.

[3] See [1] for a still relevant discussion of these interface issues.

these problems during the ontology engineering process, by making the engineering team aware of the requirements of NLP applications; we also describe the concrete use of an ontology in the task of semantic annotation of text documents.

3 Using the OntoSeed Suite in Ontology Engineering

This section describes the suite of tools we have developed to aid the design of ontologies used in language-related tasks such as semantic annotation.[4] We begin by giving a high-level description of the NLP-aided ontology engineering process, illustrating this with examples from the medical domain and explain the technical realization of the tools.

3.1 Overview and Examples

The OntoSeed suite consists of a number of programs that produce various statistical reports (as described below) given a collection of texts from a certain domain, with the aim to provide guidance for the ontology engineer on which concepts are important in this domain, and on the semantic relationships among these concepts. More specifically, it compiles five lists for each given collection of texts, as follows:

1. a list of nouns (or noun sequences in English texts; we will only write "noun" in the following) occurring in the collection, ranked by their "termhood" (i.e. their relevance for the text domain; see below);
2. nouns grouped by common prefixes and
3. suffixes, thereby automatically detecting compound nouns; and
4. adjectives together with all nouns they modify; and
5. nouns with all adjectives that modify them.

Figures 1 to 3 show excerpts of these files for a collection of German texts from the medical domain of lung pathology (the LungPath-Corpus (see [25]), consisting of 750 reports of around 300 words each; during ontology construction we used a "training-subset" of 400 documents).

As illustrated in Figure 1, terms like "Tumorzelle/tumor cell" or "Lungengewebe/lung tissue" get assigned a relatively high weight by our analysis methods (the highest weight is 112.666), which suggests that these terms denote relevant domain concepts that need to be modeled. Terms related to domain-independent concepts (e.g. terms like "System/system" or "Zeit/time" in Figure 1) tend to be ranked with significantly lower value. Having made the decision to model them, we then look up clusters in which these terms occur, as shown in Figure 2. The overview of the data afforded by ordering phrases in prefix and suffix clusters can be very useful in deciding how to model complex concepts, since there is no general, established way to model them. For example, a noun

[4] The OntoSeed tools are available at `http://nbi.inf.fu-berlin.de/research/swpatho/ontoseed.html`.

Lungenparenchym	96.515
Schnittfläche	90.993
Tumorzelle	90.951
Pleuraerguß	89.234
Entzündung	88.476
Bronchialsekret	87.711
Lungengewebe	84.918
Entzündungsbefund	83.631
....	....
Wert	1.825
System	1.761
Neuß	1.448
Bitte	1.296
Zeit	1.085
Seite	1.018

Fig. 1. Excerpt of the weighted term list (step 1)

phrase like "Tumorzelle/tumor cell" can be modeled as a single concept subclass of `Zelle` (cell), while in other settings it can be advantageous to introduce a property like `Zelle infectedBy Tumor` . The suffix clustering offers valuable information about subclasses or types of a certain concept (in our example in Figure 2 several types of cells). The prefix clustering can be utilized to identify concept parts or properties (e.g. in Figure 2 `Lungengewebe` (lung tissue) or `Lungengefaess` (lung vessel) as parts of the `Lunge` (lung)).

Finally, we look at ways in which the relevant terms are modified by adjectives in the texts, by inspecting the lists shown in Figure 3. These lists give us

B-Zellen	Lunge
Carcinom-Zellen	Lungen-PE
Schleimhautlamellen	Lungenabszeß
Plasmazellen	Lungenarterienembolie
Epitheloidzellen	Lungenbereich
Rundzellen	Lungenbezug
Alveolardeckzellen	Lungenbiopsat
Epithelzellen	Lungenblutung
Plattenepithelzellen	Lungenembolie
Karzinomzellen	Lungenemphysem
Schaumzellen	Lungenerkrankung
Riesenzellen	Lungenfibrose
Tumorzellen	Lungengefäße
Alveolarzellen	**Lungengewebe**
Zylinderzellen	Lungengewebsareal
Becherzellen	Lungengewebsprobe
Herzfehlerzellen	Lungengewebsstücke
Bindegewebszellen	Lungeninfarkt
Entzündungszellen	Lungenkarzinom
Pilzzellen	Lungenlappen

Fig. 2. Excerpt of the prefix (left, step 2) and suffix lists (right, step 3)

Tumorzelle:	92			
beschrieben	1	1%	10	10%
einzeln	1	1%	60	1%
epithelialer	1	1%	1	100%
gelegen	1	1%	16	6%
gross	4	4%	129	3%
klein	1	1%	88	1%
mittelgross	1	1%	6	16%
pas-positive	1	1%	6	16%
spindeligen	2	2%	2	100%
vergroessert	1	1%	9	11%
zahlreich	1	1%	47	2%

gross:	
Absetzungsrand	1
Abtragungsfläche	1
Biopsate	1
Bronchus	2
Lungengewebsprobe	3
Lungenlappen	3
Lungenteilresektat	1
Lungenunterlappen	5
Lymphknoten	1
Nekroseherde	13
Oberlappenresektat	1
Ossifikationen	1
PE	1
Pleuraerguß	4
Raumforderung	1
Rippe	15
Rundherd	1
Stelle	5
Tumor	1
Tumorknoten	10
Tumorzelle	4
Vene	4

Lunge:	85			
link	9	10%	53	16%
recht	7	8%	66	10%
tumorferne	2	2%	2	100%

link:	
Bronchus	7
Hauptbronchus	6
Lunge	9
Lungenlappen	1
Lungenoberlappens	1
Lungenunterlappen	4
Mittellappen	2
Oberlappen	9
Oberlappenbronchus	3
Seite	1
Thoraxseite	3
Unterlappen	4
Unterlappenbronchus	2
Unterlappensegment	1
Unterschenkels	1

Fig. 3. Excerpt of modifier list (steps 4 and 5)

information that can be used in making a decision for one of two ways of modeling the meaning of modifiers: as properties of a concept (e.g. "gross/large" as in "grosse Tumorzelle/large tumor cell"), or as part of a single concept (e.g. "link/left" in `linke Lunge` (left lung)). The decision for either of the modeling alternatives cannot be made automatically, since it depends strongly on the context of the application. However, analyzing a text corpus can support the decision process: modifiers which occur mostly together with particular noun phrases or categories of concepts, respectively, could be candidates for the single concept variant, while those used with a broad range of nouns should usually be modeled as a property. As Figure 3 shows, in our corpus the noun "Tumorzelle/tumor cell" occurs 92 times, 4 times modified with "gross/large" (i.e. approximately 4% of all modifiers). The modifier, on the other hand, occurs 129 times, so the co-occurrences of the two terms are 3% of all its occurrences, which indicates that "gross/large" is a property that is ascribed to many different concepts in the

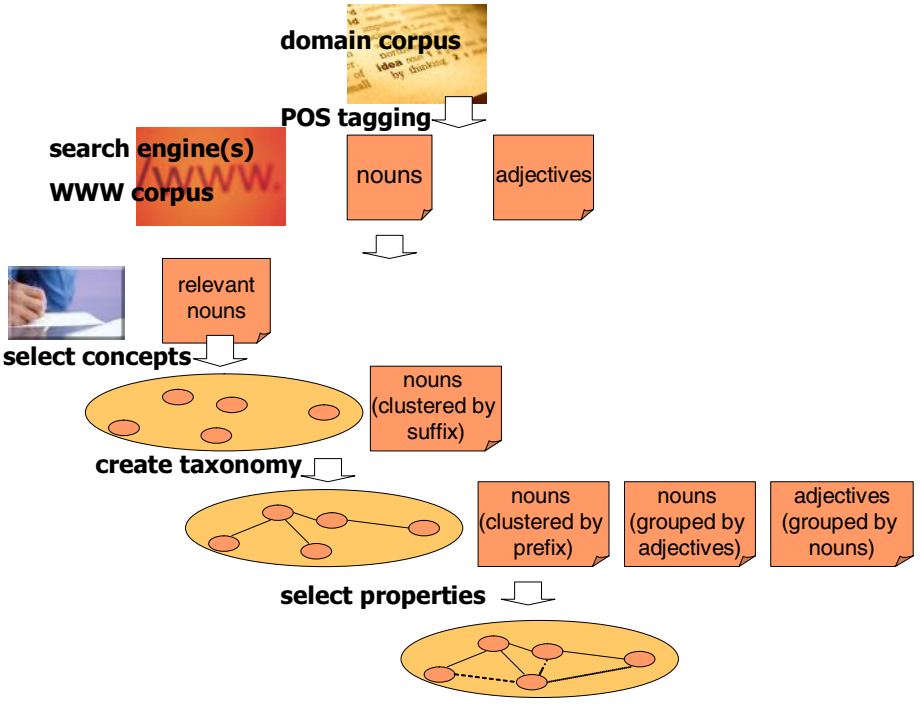


Fig. 4. The OntoSeed process

corpus. In contrast, the modifier "link/left" (the normalized form of "links/left") seems to be specific in the corpus to concepts denoting body organs like `Lunge` (lung) and its parts.[5]

To summarize, the classifications of the noun phrases and their modifiers are used as input to the conceptualization phase of the ontology building process, which is ultimately still performed *manually* (Figure 4). Nevertheless, compared to a fully manual process, preparing the text information in the mentioned form offers important advantages in the following ontology engineering sub-tasks:

- selecting relevant concepts: the ontology engineer uses the list of nouns that are ranked according to their domain specificity as described above and selects relevant concepts and relevant concept names. Domain-specific and therefore potentially ontology-relevant terms are assigned higher rankings in the noun list (see Section 4.1 for the evaluation of the ranking function). First simple concept names from the noun list are identified as being relevant for the ontology scope. Then the ontology engineer uses the prefix and suffix

[5] A possible next step in specifying possible ontology properties could be to consider verbs in correlation with noun phrases. Our tool does not yet include this feature, but see discussion below in Section 5.

clusters to decide which compound concept names should be as well included to the target ontology.
- creating taxonomy: suffix clusters can be used to identify potential subclasses.
- creating properties/relationship: the ontology engineer uses the modifier classification and the generated taxonomy to decide about relevant properties (denoted by adjectives) and about the taxonomy level the corresponding property could be defined. For example in Figure 3 most of the concepts modified by "link/left" are subsumed by `RespiratorySystem` —therefore if the ontology engineer decides to define a property corresponding to this adjective, this property will be assigned the domain `RespiratorySystem` . However since "link/left" occurs in the corpus mostly in correlation with "Lunge/lung" an alternative conceptualization is to introduce the concept `LinkeLunge` (left lung) as a subclass of `Lunge` (lung). Further relationships are induced by the decision to conceptualize relevant compound nouns as two or more related concepts in the ontology. For example if "Tumorzelle/tumor cell" is to be conceptualized in the ontology as `Zelle locationOf Tumor` the relationship `locationOf` should also be included to the ontology. Relationships between concepts (e.g. `locationOf`) are not suggested explicitly; however on the basis of taxonomy which was specified in the previous step OntoSeed is able to identify clusters of compound terms implying a similar relational semantics. For example given the fact that `Lunge` (lung) and `Herz` (heart) are both subsumed by `BodyPart` , the system suggests that the relationship correlating `Lunge` (lung) and `Infarkt` (attack) in the compound noun "Lungeninfarkt/lung attack" is the same as the one in the case of the compound "Herzinfarkt/heart attack", thus simplifying this conceptualization step even when no linguistic knowledge w.r.t. verbs is available.

3.2 OntoSeed and NLP-Friendly Ontologies

It is well accepted that NLP-driven knowledge acquisition on the basis of domain-specific text corpora is a useful approach in aiding ontology building [3, 8, 6, 13, 18, 24]. On the other hand, if the resulting ontology is targeted to language-related tasks such as semantic annotation, these tasks can be performed more efficiently by means of an ontology which is built in a "linguistics-friendly" manner. On the basis of our previous experiences in applying ontologies to medical information systems [30, 22] we identified the following set of operations which can be useful in this context and therefore should be taken into account while conceptualizing the ontology:

- logging modeling decisions: the relationship between extracted terms (resulting from the knowledge acquisition process) and the final modeled concepts should be recorded. For example the term `Klatskin tumor` will be probably modeled as a single concept, while `lung tumor` might be formalized as `tumor hasLocation lung` . These decisions should be encoded in a predefined form for subsequent NLP tasks, so that the lexicon that has to be built for these tasks knows about potential compound noun suffixes.

– naming conventions for ontology primitives: since semantic annotation requires matching text to concept names, it is necessary that the concept names are specified in a uniform, predictable manner.[6] Typically concept names are concatenated expressions—where the first letter of every new word is capitalized— or lists of words separated by delimiters (e.g. `KlatskinTumor` or `Klatskin_Tumor`). Furthermore it is often recommended to denominate relationships in terms of verbs (e.g. `diagnosedBy` , `part_of`) and attributes / properties in terms of adjectives (e.g `left`). If the names become more complex, they should be stored in a format that is easily reproducible, and allows for variations. E.g., should there be a need to have a concept name that contains modifiers ("untypical, outsized lung tumor with heavy side sequences"), the name should be stored in a format where the order of modifiers is predictable (e.g. sorted alphabetically), and the modification is disambiguated (`((lung tumor (with ((side sequences), heavy))), (untypical, outsized))`). NLP-tools (chunk parsers) can help the ontology designer to create these normalized names in addition to the human-readable ones.

We now turn to a description of the technical details of OntoSeed.

3.3 Technical Details

In the first processing step, the only kind of linguistic analysis proper that we employ is performed: determining the part of speech (e.g., "noun", "adjective", etc.) of each word token in the collection. Reliable systems for performing this task are readily available; we use the TreeTagger [26] developed at IMS in Stuttgart, Germany,[7] but other systems could be used as well.

This enables us to extract a list of all occurring nouns (or, for English, noun sequences, i.e., compound nouns; German compound nouns are, as is well known, written as one orthographic word). The "termhood" of each noun is determined by the usual *inverted document frequency* measure (tf.idf), as shown in the formula below—with the added twist, however, of using a WWW-search engine to determine the document frequency in the comparison corpus.[8,9] In the formula, $tf(w)$ stands for the frequency of word w in our collection of texts; $wf(w)$ is the number of documents in the corpus used for comparison, i.e., the number of hits for query w reported by the search engine used— in our experiments, both `www.google.com` (through the API made available by Google inc.) and `www.yahoo.com`. N is the size of the collection, determined in an indirect way

[6] This requirement, for example, is *not* fulfilled in UMLS and other medical ontologies.

[7] Freely available for academic research purposes from `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html`.

[8] See [19] for a textbook description of the family of tf.idf measures.

[9] Using the Web as a corpus in linguistic analysis has become a hot topic recently in computational linguistics (see e.g. a current special issue of *Computational Linguistics* [16]); to our knowledge, the system presented here is the first to use the web in this kind of application.

(as the search engines used do not report the number of pages indexed) by making a query for a high-frequency word such as "the" for English or "der" for German.[10]

$$weight(w) = (1 + \log tf(w)) * (\log \frac{N}{wf(w)})$$

Sorting by this weight results in lists like those shown partially in Figure 1 above; a quantitative description of the effect of this weighing procedure is given in Section 4.1.

In the next step, nouns are clustered, to find common pre- and suffixes. We use a linguistically naïve (since it only looks at strings and ignores morphology), but efficient method for grouping together compound nouns by common parts. This step is performed in two stages: first, preliminary clusters are formed based on a pre- or suffix similarity of three or more letters (i.e., "lung" and "lung pathology" would be grouped into one cluster, but also "prerogative" and "prevention"). These preliminary clusters are then clustered again using a hierarchical clustering algorithm [19], which determines clusters based on maximized pre- or suffix length (see Figure 2 above). The accuracy of the suffix clustering procedure is anew improved by using the Web to eliminate suffixes that do not denominate concepts in the real world, but are simply common endings of the clustered nouns (such as the ending "ight" in "light" or "night" in English or the German ending "tion" in "Reaktion/reaction", "Infektion/infection").

The compilation of the adjective lists (Figure 3) from the tokenized and POS-tagged text collection is straightforward and need not be explained here.

4 Evaluation

This section is dedicated to the evaluation of our approach from a technical and an application-oriented perspective. We first compare the results of our analysis procedure on two different corpora against a naïve baseline assumption (Section 4.1). The whole suite of tools is then evaluated within a real-world application setting in the medical domain. For this purpose we will compare two engineering experiments aiming at developing the same ontology—a OntoSeed-aided engineering approach and reuse-oriented one—in terms of costs and suitability of the outcomes in the target application context (Section 4.2).

4.1 Technical Evaluation

For the technical evaluation of our methods we examined the weighing function described above and the results of the prefix and suffix clustering against human expertise.

[10] This of course is just an approximation, and also the hits reported for normal queries get progressively less exact the more frequent a term is; for our purposes, this is precise enough, since for "web-frequent" terms (where *wf* ranges from 10^3 to 10^6) rough approximations already have the desired effect of pushing the weight down.

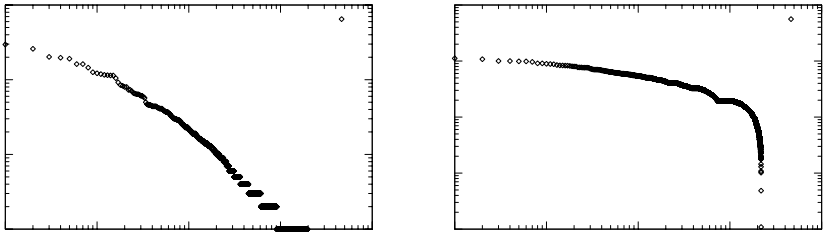


Fig. 5. Rank (x-axis) vs. frequency (left), and rank vs. weight (right); doubly logarithmically

A simple concept of the importance of a term would just treat its position in a frequency list compiled from the corpus as an indication of its "termhood". This ranking, however, is of little discriminatory value, since it does not separate frequent *domain-specific* terms from other frequent terms, and moreover, it does not bring any structure to the data: Figure 5 (left) shows a doubly logarithmic plot of frequency-rank vs. frequency for the LungPath data set; the distribution follows closely the predictions of Zipf's law [31], which roughly says that in a balanced collection of texts there will be a low number of very frequent terms and a high number of very rare terms.

In comparison, after weighing the terms as described above, the distribution looks like Figure 5 (right), again doubly logarithmically rank (this time: rank in weight-distribution) vs. weight. There is a much higher number of roughly similarly weighted terms, a relatively clear cut-off point, and a lower number of low-weight terms. A closer inspection of the weighed list showed that it distributed the terms from the corpus roughly as desired: the percentage of general terms within each 10% chunk of the list (sorted by weight) changed progressively from 5% in the first chunk (i.e., 95% of the terms in the highest ranked 10% denoted domain-specific terms) to 95% in the last chunk (with the lowest weights). We repeated this process (weighing, and manually classifying terms as *domain-specific* or *general*) with another corpus, a collection of 244 texts (approximately 80500 word tokens altogether) describing environmental aspects of

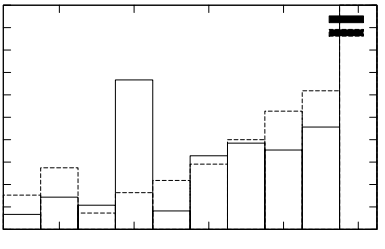


Fig. 6. The ratio of general terms per 10% chunk of weighted term list (highest weight to the left); LungPath corpus (dashed lines) and travel corpus (solid lines)

world countries, and found a similar correlation between weight and "termhood" (the results for both corpora are shown in Figure 6).

In both corpora, however, there was one interesting exception to this trend: a higher than expected number of terms in one 10% chunk in the middle of the weight distribution which were classified as irrelevant by the experts. These turned out to mostly be misspellings of names for general concepts—a kind of "noise" in the data to which the termhood measure is vulnerable (since in the misspelled form they will be both rare in the analyzed collection as well as the comparison corpus, the web, pushing them into the middle ground in terms of their weights). While this is not a dramatic problem, we are working on ways of dealing with it in a principled manner.

Further on, the comparison of the clusters generated as described in Section 3.3 with the results of the human classification revealed an average percentage of approximately 14% of irrelevant suffix/prefix clusters — a satisfactory result given the linguistically naïve algorithms employed.

We now turn to a qualitative evaluation of the usefulness of OntoSeed within a real-world Semantic Web application we are developing for the medical domain.

4.2 Application-Based Evaluation

In order to evaluate the costs and the benefits of the OntoSeed approach, we examined two subsequent semi-automatic ontology engineering experiments which aimed at building an ontology for a Semantic Web application in the domain of lung pathology [30, 22]. The application operates upon an archive of medical reports (the LungPath-Corpus mentioned above) consisting of both textual and image-based data, which are semantically annotated in order to transform them into a valuable resource for diagnosis and teaching, which can be searched in a fast, *content-based* manner [22, 30]. The semantic annotation of the data is realized by linguistically extracting semantic information from medical reports and lists of keywords associated with each of the digital images (both reports and keyword lists are available in textual form). The search is content-based in that it can make use of semantic relationships between search concepts and those occurring in the text. In the same time the medical information system can provide quality assurance mechanisms on the basis of the semantic annotations of the patient records. The annotated patient records are analyzed on-the-fly by the quality assurance component, and potential inconsistencies w.r.t. the background domain ontology are spotted.

Extracting semantic information from the medical text data is realized automatically using LUPUS—Lung Pathology System [25]. LUPUS consists of a NLP component (a robust parser) and a Semantic Web component (a domain ontology represented in OWL, and a Description Logic reasoner), which work closely together, with the domain ontology guiding the information extraction process. The result of the linguistic analysis is a (partial) semantic representation of the content of the textual data in form of an OWL semantic network of instances

of concepts and properties from the domain ontology. This ontology is used in three processing stages in LUPUS, all of which can profit from a good coverage (as ensured by building the ontology bottom-up, supported by OntoSeed) and a "linguistics-friendly" specification (as described above). The most obvious step where NLP and ontology interface is concept lookup: the ontology defines the vocabulary of the semantic representation. Since LUPUS cannot "know" whether a phrase encountered (e.g. "anthrakotischer Lymphknoten/anthracotic lymph node") is modelled as a simple or complex concept (i.e., as a concept `AnthrakotischerLymphknoten` or as a concept `Lymphknoten` having the property `anthrakotisch`) it has to first try the "longest match". For this to work, the system has to be able to construct a form that would be the one contained in the ontology. To stay with this example, an inflected occurrence of these terms, e.g. in "die Form des anthrakotischen Lymphknotens" ("the form of the anthracotic lymph node"), would have to be mapped to a canonical form, which then can be looked up. As mentioned above, in ontologies like UMLS there is no guarantee that a concept name would be in a particular form, if present at all. In a second step, the ontology is used to resolve the meaning of compound nouns and prepositions [25].

During this project we examined two alternatives for the semi-automatic generation of an ontology for lung pathology which suits the application functionality mentioned above. The two experiments were similar in terms of engineering team (and of course application context). In the first one the ontology was compiled on the basis of UMLS, as the largest medical ontology available. The engineering process was focused on the customization of pre-selected UMLS libraries w.r.t. the application requirements and resulted in an ontology of approximately 1200 concepts modeling the anatomy of the lung and lung diseases [22, 21]. Pathology-specific knowledge was found to not be covered by available ontologies to a satisfactory extent and hence was formalized manually. In the second experiment the ontology was generated with the help of the OntoSeed tools as described in Section 3.1.[11]

We compared the efforts invested in the corresponding engineering processes and analyzed the fitness of use of the resulting ontologies, in our case the results these ontologies achieved in semantic annotation tasks. The main advantages of the OntoSeed-aided experiment compared to the UMLS-based one are the significant cost savings in conjunction with the improved fitness of use of the generated ontology.

From a resource point of view, building the first ontology involved four times as many resources than the second approach (5 person-months for the UMLS-based ontology with 1200 concepts vs. 1.25 person-months for the "text-close" ontology of a similar size). We note that the customization of UMLS[12]required

[11] The knowledge-intensive nature and the complexity of the application domain convinced us to not pursue the third possible alternative, building the ontology from scratch.

[12] Customization includes getting familiar with, evaluating and extracting relevant parts of the source ontologies.

over 45% of the overall effort necessary to build the target ontology in the first experiment. Further 15% of the resources were spent on translating the input representation formalisms to OWL. The reuse-oriented approach gave rise to considerable efforts to evaluate and extend the outcomes: approximately 40% of the total engineering effort were necessary for the refinement of the preliminary ontology. The effort distribution for the second experiment was as follows: 7% of the overall effort was invested in the selection of the relevant concepts. Their taxonomical classification required 25% of the resources, while a significant proportion of 52% was spent on the definition of additional semantic relationships. Due to the high degree of familiarity w.r.t. the resulting ontology, the evaluation and refinement phase in the second experiment was performed straight forward with 5% of the total efforts. The OWL implementation necessitated the remaining 11%.

In comparison with a fully manual process the major benefit of OntoSeed according to our experiences would be the pre-compilation of potential domain-specific terms and semantic relationships. The efforts invested in the taxonomical classification of the concepts are comparable to building from scratch, because in both cases the domain experts still needed to align the domain-relevant concepts to a pre-defined upper-level ontology (in our case the Semantic Network core medical ontology from UMLS). The selection of domain-relevant terms was accelerated by the usage of the termhood measure as described above since this avoids the manual processing of the entire domain corpus or the complete evaluation of the corpus vocabulary. The efforts necessary to conceptualize the semantical relationships among domain concepts were reduced by the clustering methods employed to suggest potential subClass and domain-specific relationships. However the OntoSeed approach assumes the availability of domain-narrow text sources and the quality of its results depends on the quality/domain relevance of the corpus.

In order to evaluate the quality of the outcomes (i.e. the ontologies resulted from the experiments mentioned above) we compared their usability within the LUPUS system by setting aside a subset (370 texts) of the LungPath corpus and comparing the number of nouns matched to a concept. Using the ontology created by using OntoSeed (on a different subset of the corpus) as compared to the ontology derived from UMLS resulted in a 10 fold increase in the number of nouns that were matched to an ontology concept—very encouraging results indeed, which indicate that our weighting method indeed captures concepts that are important for the whole domain, i.e. that the results generalize to unseen data. However, this evaluation method does of course not tell us how good the recall is w.r.t. all potentially relevant information, i.e., whether we not still miss relevant concepts—this we could only find out using a manually annotated test corpus, a task which is currently performed. In a preliminary evaluation, domain experts selected the most significant (w.r.t their information content) concepts from an arbitrary set of 50 patient reports. These concepts are most likely to be used as search terms in the envisioned system because of

their high domain relevance (as assigned by human experts). The ontology derived from UMLS contained 40% of these concepts. However, only 8% of them were directly found in the ontology,[13] while the usage of the remaining 32% in the automatic annotation task was practically impossible because of the arbitrary concept terminology used in UMLS. As underlined before UMLS contains concept names in various forms ("noun, adjective", "adjective noun", full phrases—to name only a few). In comparison, the OntoSeed-generated ontology was able to deliver 80% of the selected concepts with an overall rate of 61% directly extracted concepts. In contrast to the UMLS-oriented case, the 19% of the remaining, indirectly recognized concepts could be de facto used in automatic annotation tasks, due to the NLP-friendly nature of the ontology. In the second ontology the concepts were denominated in an homogeneous way and critical modeling decisions were available in a machine-processable format.

The results of the evaluation can of course not be entirely generalized to arbitrary settings. Still, due to the knowledge-intensive character of its processes, medicine is considered a representative use case for Semantic Web technologies [17]. Medicine ontologies have already been developed and used in different application settings: GeneOntology [5], NCI-Ontology [11], LinKBase [4] and finally UMLS. Though their modeling principles or ontological commitments have often been subject of research [28, 23, 27, 10], there is no generally accepted methodology for how these knowledge sources could be *efficiently* embedded in real Semantic Web applications. At the same time, the OntoSeed results could be easily understood by domain experts, enabled a rapid conceptualization of the application domain whose quality could be efficiently evaluated by the ontology users. Though OntoSeed was evaluated in a particular application setting, that of semantically annotating domain-narrow texts using NLP techniques, we strongly believe that the tools and the underlying approach are applicable to various domains and domain specific corpora with similar results. This assumption was in fact confirmed by the technical evaluation of the tools on a second English corpus from the domain of tourism.

5 Conclusions and Future Work

In this paper we presented methods to aid the ontology building process. Starting from a typical setting—the semantic annotation of text documents—we introduced a method that can aid ontology engineers and domain experts in the ontology conceptualization process. We evaluated the analysis method itself on two corpora, with good results, and the whole method within a specific application setting, where it resulted in a significant reduction of effort as compared

[13] Directly extracted concepts are the result of simple string matching on concept names or their synonyms. The indirect extraction procedure assumes that a specific concept available in the text corpus is formalized "indirect" in the ontology i.e. as a set of concepts and semantical relationships; see Section 3.

to adaptation of existing resources. Additionally, the method suggests guidelines for building "linguistics-friendly" ontologies, which perform better in ontology-based NLP tasks like semantic annotation.

As future work, we are investigating to what extent analyzing verbs in domain specific texts can be used to aid ontology building, and ways to extract more taxonomic information from this source (e.g. information about hypnoym (is-a) relations, via the use of the copula (x is a y)), while still being as linguistically knowledge-lean as possible. Second, we are currently implementing a graphical user interface to simplify the usage of the presented tools in ontology engineering processes and in the same time to extend the automatic support provided by the OntoSeed approach. Lastly we will complete the evaluation of the LUPUS system and the benefits of using "NLP-friendly" ontologies for the semantic annotation task in more detail.

Acknowledgements

This work has been partially supported by the EU Network of Excellence "KnowledgeWeb" (FP6-507482). The project "A Semantic Web for Pathology" is funded by the DFG (German Research Foundation). We are also grateful to Google Inc. for making available their API to the public. Thanks to Manfred Stede for valuable comments on a draft of this paper.

References

1. J. A. Bateman. The Theoretical Status of Ontologies in Natural Language Processing. KIT-Report 97, Technische Universität Berlin, May 1992.
2. K. Bontcheva, H. Cunnigham, V. Tablan, D. Maynard, and H. Saggion. Developing Reusable and Robust Language Processing Components for Information Systems using GATE. In *Proceedings of the 3rd International Workshop on Natural Language and Information Systems NLIS02*. IEEE Computer Society Press, 2002.
3. P. Buitelaar, D. Olejnik, and M. Sintek. A Protege Plug-In for Ontology Extraction from Text Based on Linguisitc Analysis. In *Proceedings of the European Semantic Web Symposium ESWS-2004*, 2004.
4. W. Ceusters, B. Smith, and J. Flanagan. Ontology and Medical Terminology: Why Description Logics are Not Enough. In *Proc. Towards An Electronic Patient Record, TEPR2003*, 2003.
5. The Gene Ontology Consortium. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25:25–30, 2000.
6. M. Dittenbach, H. Berger, and D. Merll. Improving Domain Ontologies by Mining Semantics from Text. In *Proceedings of the first Asian-Pacific conference on Conceptual modelling*, pages 91–100. Australian Computer Society, Inc., 2004.
7. P. Drouin. Detection of Domain Specific Terminology Using Corpora Comparison. In *Proceedings of the International Language Resources Conference LREC04*, Lisbon, Portugal, May 2004.

8. D. Faure and Poibeau T. First Experiments of Using Semantic Knowledge Learned by ASIUM for Information Extraction Task Using INTEX. In *Ontology Learning ECAI-2000 Workshop*, 2000.
9. M. Fernández-López and A. Gómez-Pérez. Overview and Analysis of Methodologies for Building Ontologies. *Knowledge Engineering Review*, 17(2):129–156, 2002.
10. A. Gangemi, D. M. Pisanelli, and G. Steve. An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.
11. J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, B. Parsia, and J. Oberthaler. The National Cancer Institute's Thesaurus and Ontology. *Journal of Web Semantics*, 1(1), 2003.
12. I. Gurevych, R. Porzel, E. Slinko, N. Pfleger, J. Alexandersson, and S. Merten. Less is More: Using a Single Knowledge Representation in Dialogue Systems. In *Proceedings of the HLT-NAACL Workshop on Text Meaning*, 2003.
13. U. Hahn and K. Schnattinger. Towards Text Knowledge Engineering. In *Proceedings of the AAAI/IAAI*, pages 524–531, 1998.
14. J. R. Hobbs, W. Croft, T. Davies, D. Edwards, and K. Laws. Commonsense metaphysics and lexical semantics. *Compuational Linguistics*, 13(3–4), 1987.
15. K. Kageura and B. Umino. Methods of Automatic Term Recognition. *Terminology*, 3(2):259–289, 1996.
16. A. Kilgarriff and G. Grefenstette. Introduction to the Special Issue on the Web as Corpus. *Computational Linguistics*, 29(3):333–348, September 2003.
17. KnowledgeWeb European Project. Prototypical Business Use Cases (Deliverable D1.1.2 KnoweldgeWeb FP6-507482), 2004.
18. A. Maedche and S. Staab. Semi-automatic Engineering of Ontologies from Text. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering SEKE2000*, 2000.
19. C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, USA, 1999.
20. S. Nirenburg and V. Raskin. The Subworld Concept Lexicon and the Lexicon Management System. *Computational Linguistics*, 13(3–4), 1987.
21. E. Paslaru Bontas, M. Mochol, and R. Tolksdorf. Case Studies in Ontology Reuse. In *Proceedings of the 5th International Conference on Knowledge Management IKNOW05*, 2005.
22. E. Paslaru Bontas, S. Tietz, R. Tolksdorf, and T. Schrader. Generation and Management of a Medical Ontology in a Semantic Web Retrieval System. In *CoopIS/DOA/ODBASE (1)*, pages 637–653, 2004.
23. D.M. Pisanelli, A. Gangemi, and G. Steve. Ontological Analysis of the UMLS Metathesaurus. *JAMIA*, 5:810 – 814, 1998.
24. M. L. Reinberger and P. Spyns. Discovering Knowledge in Texts for the Learning of DOGMA-inspired Ontologies. In *Proceedings of the Workshop Ontology Learning and Population*, ECAI04, pages 19–24, Valencia, Spain, August 2004.
25. D. Schlangen, M. Stede, and E. Paslaru Bontas. Feeding OWL: Extracting and Representing the Content of Pathology Reports. In *Proceedings of the NLPXML Workshop 2004*, 2004.
26. H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, 1994.

27. S. Schulze-Kremer, B. Smith, and A. Kumar. Revising the UMLS Semantic Network. In *Proceedings of the Medinfo 2004*, 2004.
28. B. Smith, J. Williams, and S. Schulze-Kremer. The Ontology of GeneOntology. In *Proceedings of the AMIA*, 2003.
29. M. Stede and D. Schlangen. Information-Seeking Chat: Dialogues Driven by Topic-Structure. In *Proceedings of Catalog (the 8th Workshop on the Semantics and Pragmatics of Dialogue SemDial04)*, pages 117–124, 2004.
30. R. Tolksdorf and E. Paslaru Bontas. Organizing Knowledge in a Semantic Web for Pathology. In *Proceedings of the NetObjectDays Conference*, 2004.
31. G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Cambridge, MA, USA, 1949.

Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences

Eva Blomqvist

School of Engineering, Jönköping University, P.O. Box 1026, SE-551 11 Sweden
eva.blomqvist@ing.hj.se

Abstract. The main contribution of this paper is an initial method for automatically exploiting ontology design patterns with the aim of further automating the creation of enterprise ontologies in small-scale application contexts. The focus is so far on developing a fully automated construction method, thereby somewhat reducing the requirements on ontology customization and level of detail. In this paper we present an approach how to use knowledge (patterns) from other areas, like data modeling, knowledge reuse, software analysis and software design, to create ontology patterns. These design patterns are then used within our method for automatically matching and pruning them, in accordance with information extracted from existing knowledge sources within the company in question. Though the method still needs some fine-tuning, it has already been used when creating an enterprise ontology for a supplier-company within the automotive industry.

1 Introduction

The area of ontology engineering is developing fast, new methods and tools are introduced continuously. Recent developments involve semi-automatic ontology construction to reduce the time and effort of constructing an ontology, but also to reduce the need of expert "ontology engineers". Especially when considering small-scale application cases the need for reducing the effort and expert requirements is obvious.

One way of reducing this effort is by further facilitating semi-automatic construction of ontologies, but also by introducing reuse in ontology engineering. Patterns have proved to be a fruitful way to handle the problem of reuse in a construction setting. In software engineering it is already the commonly accepted way to build software, by using for example design and architecture patterns. This could also become true in ontology engineering.

Some might question this by stating that reuse of ontologies, and knowledge in general, is a much more complex task and the result needs to be precisely adapted to the users' context. This is true, but it has been proven, for example considering data model patterns and reuse of problem-solving methods, that knowledge reuse is actually possible. The focus of this paper is the use of a specific

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1314–1329, 2005.

kind of patterns, ontology design patterns, for knowledge reuse and automatic ontology construction.

In the next section some definitions are given, e.g. what is meant by ontology, pattern, and ontology patterns, together with a presentation of related work. In Sect. 3 patterns are discussed in more detail, and our reuse approach towards patterns is introduced. The section continues by presenting the method for using ontology design patterns in automatic ontology creation. The approach was validated and used for creating a real-world ontology within a research project in the automotive suppliers domain. The results and experiences from this project are presented in Sect. 4. Finally in Sect. 5 some conclusions are drawn and future research possibilities are presented.

2 Background and Related Work

This section presents some background and definitions together with other research approaches which are related to our own approach.

2.1 Ontologies

Ontology is a popular term today, used in many areas and defined in many different ways. In this paper ontology is defined as:

An ontology is a hierarchically structured set of concepts describing a specific domain of knowledge, that can be used to create a knowledge base. An ontology contains concepts, a subsumption hierarchy, arbitrary relations between concepts, and axioms. It may also contain other constraints and functions.

The definition is a reformulation of what is contained in the most common ontology definitions today. It sets constraints on what can be denoted by "ontology" but even using this definition, ontologies can be used for many different purposes and applications, and they can be constructed and structured in many different ways. One of the most common ways to describe the level of generality of an ontology is by using the structure in Fig. 1 [1].This shows how a general top-level ontology can be specialised into a domain ontology (describing some knowledge domain) or a task ontology (describing a generic task applicable to several domains). Domain and task ontologies can in turn be specialised, and combined, into application ontologies. An application ontology describes a certain task (or tasks) specialised for a specific domain. As will be shown this paper is mostly concerned with domain ontologies within enterprises.

Another categorisation of ontologies can be obtained by classifying them by their intended use [2]. There are three main levels, terminological ontologies, information ontologies and knowledge modelling ontologies, where each level adds further complexity to the ontology structure. Terminological ontologies are used to structure terminology, information ontologies to structure information in general and knowledge modelling ontologies to perform more advanced knowledge management tasks. For example, in a terminological ontology often a simple

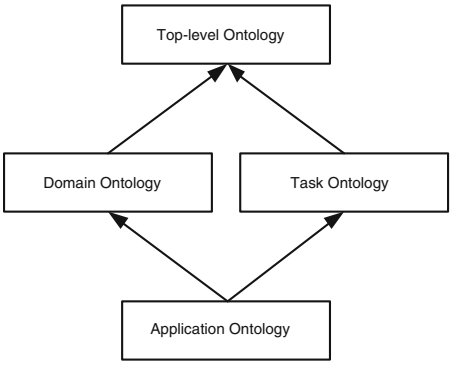


Fig. 1. Different kinds of ontologies according to their level of generality [1]

taxonomy of terms is enough. In an information ontology general relations and simple axioms may also be needed, while in a knowledge modelling ontology more advanced axioms and constraints are very often required, in order to perform reasoning tasks. Our focus is mainly on information ontologies.

A final dimension to be considered when discussing ontologies today is the construction, maintenance and evolution strategy used. Here it is important to distinguish between purely manual methods and semi-automatic (or even fully automatic) methods. It is important to note that most management of ontologies is in some way semi-automatic, since the use of tools for tasks in this area is common practice. The extent of automation, from purely manual to fully automatic, of the different tasks in an ontology life-cycle has an impact on both the type of ontology and the possible patterns needed for its creation and evolution. As stated at the beginning of this paper our aim is to further automate the ontology construction process.

2.2 Patterns in Computer Science

The idea of using patterns in computer science came from the architecture field already in the 70's and 80's. Generally patterns can assist on the following issues [3]:

- Patterns address a recurring design problem that arises in specific design situations, and presents a solution to it.
- Patterns document existing, well-proven design experience.
- Patterns identify and specify abstractions that are above the level of single classes and instances of components.
- Patterns provide a common vocabulary and understanding for design principles.
- Patterns are a means of documenting an architecture.
- Patterns help in constructing complex and heterogeneous architectures.
- Patterns help in managing complexity.

The probably most well-known book in the software pattern community is the book on design patterns [4], but there exist many other kinds of patterns on different levels of abstraction and intended for different usage areas. There are patterns to help data modellers in describing a company's information structures, either for constructing a database or a model used by an application [5]. Further, there exist semantic patterns, used as meta-structures in knowledge engineering, and knowledge patterns for describing implementation independent constructs used in the knowledge base development of artificial intelligence. Also, complete libraries of generic problem-solving methods [6] are present, object system models for reuse in software engineering [7], and many other approaches.

Ontology Patterns. As has been described above, patterns are widely used in many areas of computer science today. The ontology community has not yet adopted the pattern idea on a broader scale. There exist a few patterns for ontologies, and all of them are specialised for some specific kind of ontology (with respect to both level of generality, usage area, and construction method as defined in section 2.1).

For specific ontology languages, patterns are being developed to help engineers construct useful and well-structured ontologies. An example is the OWL-patterns created by the W3C Ontology Engineering Patterns Task Force, within the Semantic Web Best Practices and Deployment Working Group [8]. These patterns describe how to implement certain features in OWL, like the notion of n-ary relations and how to represent classes as property values.

Examples of meta-structures for ontologies, which can be denoted patterns, are semantic patterns, for describing implementation independent logical constructs [9] [10]. These patterns can for example describe notions like locally inverse relations and composition of relations. The implementation of the patterns (as axioms in an ontology) depends very much on which logical formalism is used to construct the ontology. By first using the semantic patterns to describe the meaning of the constructs, the language specific implementation can be postponed.

Similar to software design patterns are the ontology design patterns developed by the Laboratory of Applied Ontology [11] and the design patterns developed for ontologies in molecular biology [12]. These patterns are intended to be used when constructing ontologies, as a good way to structure and implement smaller parts of the ontology. The patterns are quite general and describe for example how to keep track of updates between different partitions of an ontology, or how to structure a terminological hierarchy. That is, they are often application and domain independent.

In our research we have previously attempted to structure the area of ontology patterns [13]. Five levels of patterns can be identified, from the application-level where patterns deal with the usage and interaction of different ontologies within an application down to the syntactic level where patterns are language specific. The other levels are architecture patterns, describing how to arrange the overall structure of the ontology, design patterns, describing how to structure

and implement smaller parts (perhaps modules) of the ontology, and semantic patterns describing the primitives of an ontology in a language independent way.

On all these levels there can be patterns for constructing ontologies for different usages and with different levels of generality, as well as for different construction methods (manual or automatic), as presented in Sec. 2.1. Patterns are not yet developed at all of these levels and with all these aspects in mind. As can be noted when studying the related work presented above, the focus has mainly been on the lower levels (syntactic and semantic patterns). Also some design patterns are present, but mainly for manual use. The focus of this paper will be on design patterns for automatic use. This means that the patterns have to be more specific and formalised than patterns intended to be used manually.

2.3 Semi-automatic Ontology Construction

There are a number of existing semi-automatic approaches when it comes to ontology construction. Semi-automatic ontology construction, sometimes denoted ontology learning, is a quite new field in ontology research. Many researchers have realized that building ontologies from scratch, and by hand so to speak, is too resource demanding and too time consuming. In most cases there are already different knowledge sources that can be incorporated in the ontology engineering process. Such existing knowledge sources can be documents, databases, taxonomies, web sites, applications and other things. The question is how to extract the knowledge incorporated in these sources automatically, or at least semi-automatically, and reformulate it into an ontology.

To solve this problem the researchers in the field have come up with varying solutions. The name "ontology learning" refers to that many solutions build on some learning mechanism originating in the area of machine learning. The approaches have some differences in choice of solutions and both in choice of input and output, but the main differences lie in the aim of the methods. Some are aimed at terminological ontologies only, while others are more aimed at information ontologies on the domain ontology level. It is the latter kind that is most interesting in the scope of this paper.

Many of the semi-automatic approaches rely on basic techniques that were developed for other purposes. The techniques often originated in data or text mining, or in machine learning. Some of the parts present in most systems are term extraction by linguistical analysis and relations extraction by co-occurrence theory (or association rules). Some systems also try to automatically build a concept taxonomy by using some form of concept clustering. This clustering can be based on purely linguistic information and linguistic patterns [14] [15] or on details of the already extracted concepts [16](like instances or attribute values). These systems are the ones that come closest to being fully automatic. Still there is no way to include for example arbitrary relations in these clustering algorithms. The approach suggested in this paper tries instead to start with the demand of a fully automatic system, and realize it by using patterns to construct the ontology from single concepts and relations extracted from texts.

3 Automatic Ontology Construction

In this section we present our approach, first how to create ontology design patterns and then how to use them in an automatic ontology construction process.

3.1 Approach to Creation of Ontology Patterns

One can imagine two general approaches when addressing the problem of creating (or extracting) ontology design patterns. The first one is to take all existing ontologies and derive patterns from them. The other one is to develop criteria of "good design" and construct patterns that reflect these principles. Unfortunately, to derive criteria of how to design ontologies for all situations is almost impossible, so the second approach is not practically feasible.

The first approach is the common way, in for example software engineering, to "discover" patterns using existing structures and experiences. Unfortunately enterprise ontologies are so far quite scarce and often sparsely documented, which makes it hard to extract patterns from them. It would be like extracting software design patterns from actual running systems, instead of from design specifications. This can be done but it is a very complex task.

One "middle course" is to try to draw from knowledge already accumulated in other areas. Many patterns in computer science also describe some kind of knowledge, although they might have a different purpose than constructing ontologies. Our approach is to study patterns from other areas in order to develop ontology patterns, since the focus in many cases is still on enterprises. For our first experiments in the area some sources of pattern-like constructs where chosen, among others sources from data modelling, software engineering, knowledge and software reuse (for example reuse of problem-solving methods).

One of the areas most similar to ontology engineering is the database domain. Database structures are modelled to describe a domain, of course with the intent of storing data efficiently, but they share many properties with ontologies. Especially when, as in our case, ontologies are restricted to information ontologies within enterprises, since one of the largest application areas of databases is of course storing enterprise data. The greatest differences lie in the aim and usage of the structures. While databases are intended to store data and their relations, ontologies are intended to bring meaning and structure to the enterprise's content.

Since we expected to gain from reusing this knowledge, our first attempt was to "translate" the knowledge stored in for example data model patterns into ontology design patterns. Some parts of the data model patterns were left out for these first tests, like for example cardinality constraints. The aspects that were taken into consideration and their chosen mappings to ontologies are shown in Table 1.

To illustrate this mapping Fig. 2 shows a very simple data model pattern describing organisations of different types (the pattern can be found in [17]). This pattern has then been translated into an ontology design pattern, which is depicted (using the tool KAON [18]) in Fig. 3.

Table 1. Chosen mappings between data models and ontologies

Data Model	Ontology
Entity	Concept
Attribute	Relation to "attribute"-concept
Subtypes/Supertypes	Subsumption hierarchy
Relationships	Relations
Mutually exclusive sets	Disjoint concepts

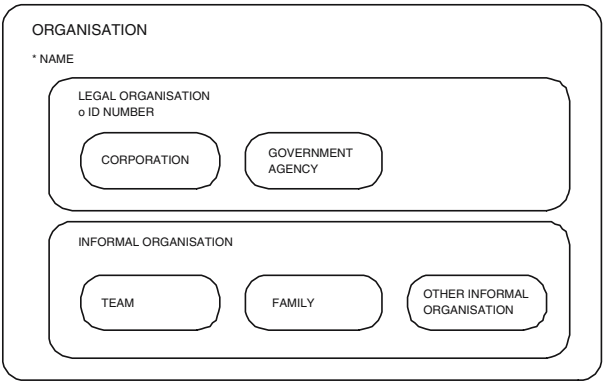


Fig. 2. Data model pattern describing organisations [17]

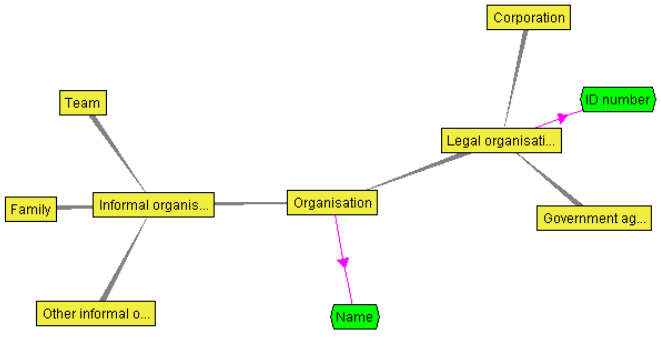


Fig. 3. Resulting ontology design pattern

Another source of patterns is the goal structures of the object system models, presented in [7]. These constructs can be viewed as representing processes within a company, consisting of a set of goals and choices for how to accomplish them. Table 2 shows the mappings chosen to adapt a goal structure for use as an ontology design pattern. Other similar mappings were then determined for the remaining sources but will not be displayed in this paper.

Table 2. Chosen mappings between goal structures and ontologies

Goal Structure	Ontology
Node (goal or method)	Concept
AND-relations	Part-of relations
OR-relations	Choice relation (related to a new concept representing the category of possible choices)
Iterations	Part-of relations

Since the naming of terms in the source areas might not be adequate for ontologies, and ontologies aim at describing a concept and not mainly a term, the resulting ontology design patterns were enriched with label synonyms. The synonyms used were WordNet synsets [19]. This gives the patterns a certain level of abstraction, since the concepts do not simply represent a linguistic term but actually a concept with an option of choosing the concept representation out of all possible synonyms.

At this stage quite simple parts of the ontologies have been considered, for example only very simple axioms have been incorporated in the patterns (like disjointness of concepts). In future experiments more complex axioms should also be developed for the patterns, in order to make the resulting ontology more useful and precise. The level of detail of the patterns is also to be more carefully considered in the future. The instance level is not present since instances can be viewed as a specialisation for a specific case, and this would not be appropriate in an abstract pattern, but where to draw the line between classes and instances is always an open question.

3.2 Method for Automatic Ontology Construction

Since the focus of our research is mainly on patterns, and the more basic parts of semi-automatic ontology creation have already been well researched, our approach uses existing tools to extract concepts and relations. So far very few of these approaches go further than extraction of single terms or relations so this is where our approach is needed.

The method that we propose in this paper is still under refinement but the general idea can be viewed in Fig. 4. The idea is to take the extracted terms and relations, match them against the patterns and depending on the result use parts of the patterns to build the ontology. As a preprocessing step a text corpus is analysed by some term extraction software, which renders a list of possibly relevant terms. This list of terms is the input to our method.

The first step is then to match the list of terms against all the patterns in the library. The method used for this matching is not fixed at this stage of our methodology development. There exist many matching approaches for lexical matching to choose from, and for each such method it also has to be decided

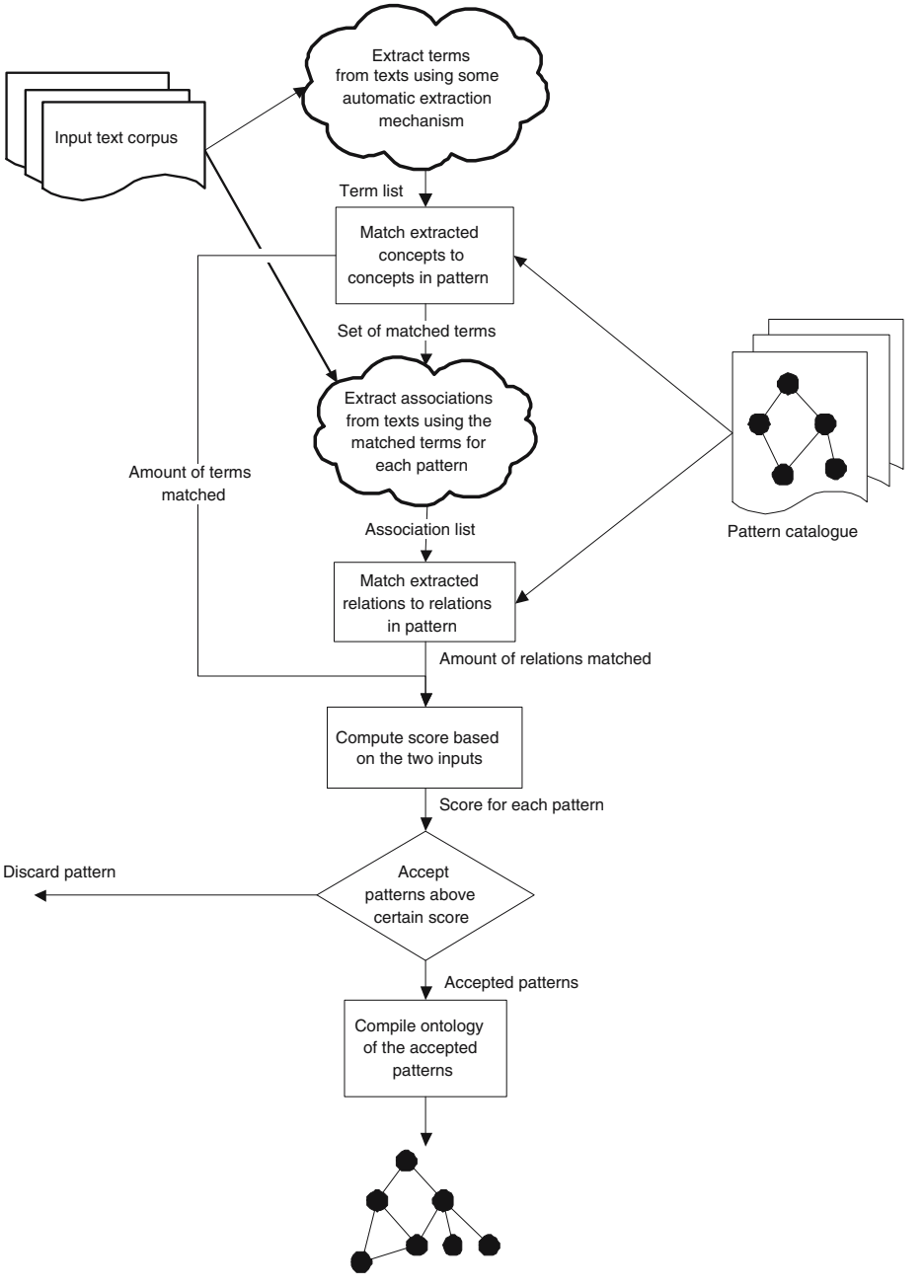


Fig. 4. The basic steps of the proposed method

on what grounds to register an accepted match. In Sect. 4 we show one possible realisation of this matching process, using a string matching tool.

This first step results in two things. First, a score for each pattern representing the amount of terms in the pattern that matches the term-list. Second, a list of the terms in the extracted term-list that were considered to have been a match to the pattern at hand. The list of correctly matched terms is used (for each pattern) to extract possible relations in the pattern also present in the text corpus. This relation extraction is also outside the scope of our research, since tools for this already exist. The output of this should be a list of connected concepts together with their relations. These connected concepts are then compared to the pattern currently in question and a score is computed based on the amount of relations in the pattern that also exist among the extracted relations. The exact nature of the matching is not fixed at this stage of our research, an example procedure is presented in Sect. 4.

Next, the two scores obtained (matched concepts and matched relations) are weighted together to form a "total matching-score" for each pattern. Then a decision is made according to some threshold value, which patterns will be kept and included in the resulting ontology and which will be discarded. Finally, an ontology is built from the accepted patterns. This construction is based on the lists generated at the beginning of the process, the lists of matched concepts and relations. The basic method for building the ontology, by considering one pattern at a time, is depicted in Fig. 5.

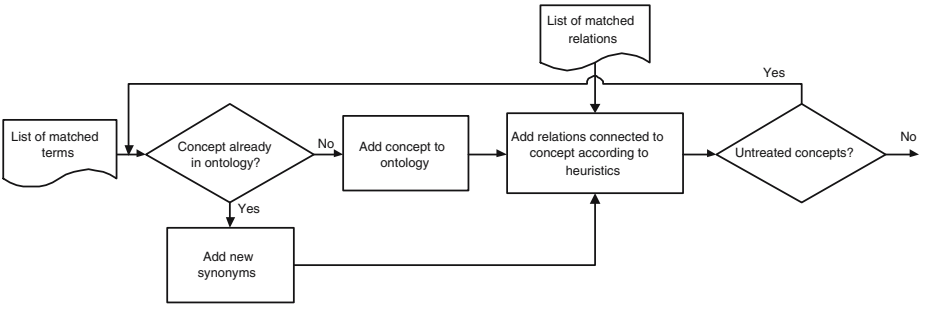


Fig. 5. The basic steps of the ontology building, for each accepted pattern

For each pattern the lists of matches are used as input. Then there is an iterative step which considers all concepts and relations of the pattern. If the concept at hand is already in the ontology (no conflicting synonyms) the concept in the ontology is only enriched with all new synonyms that have been found for the new concept. If the concept is not in the ontology already, it is added along with all matched synonyms.

Relations between concepts are added according to some heuristics. Relations to and from concepts already present in the ontology and matched in the matching process previously are added. Other heuristics, like adding hierarchical relations directly between concepts which in the pattern are separated only

by an intermediate node, or adding a node if more than a certain number of relations lead to it could be possible. The iterative process continues until there are no more concepts or relations of the pattern to consider.

A criticism one might propose is that the patterns might not perfectly reflect what the enterprise actually means by the terms they use, depending on how the matching is done. This is certainly a concern to be evaluated in the future but so far our focus is on creating a fully automatic process. Also if a concept has been matched to the wrong sense intended by the enterprise, this will not have a great impact on the resulting ontology since other parts of the pattern will not be matched and thereby pruned in the later stages of the process. After the process is finished there is also nothing which opposes a manual validation or refinement of the resulting ontology, but this is outside the scope of this paper. If instances are needed in the ontology then a post-processing step is certainly needed.

4 Experiment: Creating an Enterprise Ontology for an Automotive Supplier

In order to validate the approach introduced in Sect. 3, we performed an experiment, which was part of the research project SEMCO. SEMCO aims at introducing semantic technologies into the development process of software-intensive electronic systems in order to improve efficiency when managing variants and versions of software artifacts. The scope of the experiment was to fully automatically construct a selected part of the enterprise ontology for one of the SEMCO project partners, based on a collection of documents from product development. The purpose of the ontology is to support capturing of relations between development processes, organisation structures, product structures and artifacts within the development process.

The ontology is so far limited to describing the requirements engineering process, requirements and specifications with connections to products and parts, organisational concepts and project artifacts. This is due to the limited amount of patterns used in this initial experiment. 25 ontology patterns were developed based on the approach presented in Sect. 3.1. The sources of the 25 patterns used are specified in Table 3.

The text corpus used in the process consisted of software development plans, software development process descriptions, and other similar documents. The extraction of relevant terms and relations from these texts were performed using the tool Text-To-Onto [24] within the KAON tool-suite [18]. This choice was made mainly out of convenience, since this is one of the most mature tools and it is freely available on the KAON website. Any evaluation of the methodology will be conducted with the same prerequisite extraction method, so this is not considered important at this early stage of validation.

In order to keep the experiment on a reasonable scale, to be able to validate the accuracy and efficiency of the method manually, the concepts and patterns were restricted to a relatively low number. Therefore, a frequency threshold of 25

Table 3. Patterns used in the experiment and their original sources

Pattern name	Source
Actions	Analysis pattern [20]
Analysis and modeling	Goal structure [7]
Communication event	Data model [17]
DOLCE Descriptions and Situations	Top-level ontology [21]
Employee and department	Data model [17]
Engineering change	Data model [22]
Information acquisition	Goal structure [7]
Organisation	Data model [17]
Parts	Data model [22]
Party	Data model [17]
Party relationships	Data model [17]
Party roles	Data model [17]
Person	Data model [17]
Planning and scheduling	Goal structure [7]
Positions	Data model [17]
Product	Analysis pattern [20]
Product associations	Data model [17]
Product categories	Data model [17]
Product features	Data model [17]
Requirements	Data model [17]
Requirements analysis	Goal structure [7]
System	Cognitive pattern taxonomy [23]
System analysis	Cognitive pattern taxonomy [23]
System synthesis	Cognitive pattern taxonomy [23]
Validate and test	Goal structure [7]
Work effort	Data model [17]

was set at the concept extraction and used during the experiment. This rendered 190 concepts as the initial input of the construction process.

The matching of the pattern-concepts and their synonyms against the extracted concepts was done using a lexical matching tool. In order to be able to test different matching algorithms and metrics, an existing string matching comparison tool [25] was selected for this task (Secondstring, available at [26]). For simplicity a Jaccard string similarity metric was selected for this initial experiment and the threshold for a match was set to a similarity level of 0.5. When this matching was completed for each pattern a score was computed according to the number of matched concepts. There was also a list of correctly matched concepts which were again used with the Text-To-Onto tool in order to extract relations between those concepts.

When matching relations, all arbitrary relations were assumed to be transitive, since this simplified the matching task when an intermediate concept did not exist in the matched concepts. The score representing the number of correctly matched relations then was weighted together with the score of matched concepts into a total score for each pattern. Since the relations were deemed more important than the lexical matching of concept names, the relation scores were in this experiment given a higher weight than the concept-matching scores.

Most patterns received a quite low score. This was mainly due to the difficulty of extracting relations and thereby also matching relations to the patterns. A quite low threshold score was set after some considerations and manual evaluations of relevancy of the patterns, and also because by accepting quite a few

patterns the properties of the pruning algorithm could be studied more thoroughly. This resulted in 14 accepted pattern out of the original 25.

The 14 accepted patterns were then compiled into an ontology using the method specified in Sec. 3.2. Each pattern was treated separately, one at a time. For each pattern each of its concepts was considered. If a matched concept was not already in the ontology it was included, together with all its matched synonyms. Otherwise only the missing synonyms were added. Then all relations leading to and from the concept were considered. Using a set of heuristics some of the relations were added to the resulting ontology.

Basic descriptions of some heuristics used are for example:

- Include all relations between added concepts, even if they were not matched.
- Use the transitive property of hierarchical relations, if an intermediate concept is missing add the child directly at the level of the missing concept.
- An associative relation which originally relates two concepts is added even if one of the concepts is missing, if and only if there is a child concept of the missing concept present in the ontology...

The resulting ontology contains 35 concepts directly beneath the root concept and in total a set of 85 concepts. Figure 6 shows a part of the resulting ontology as a screenshot from the visualisation tool in KAON. This shows for example concepts concerning products, their features, and their connection to the product requirements.

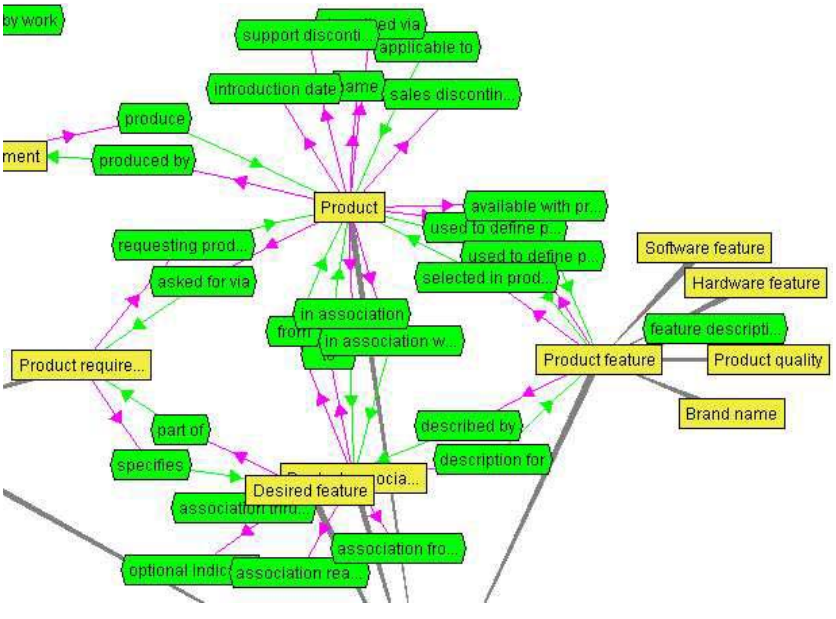


Fig. 6. A part of the resulting ontology

4.1 Remaining Issues

This first test of the method and the patterns by creating a real-world ontology has resulted in a useful and well-structured ontology. Still there remains several issues which should be subject of improvement in the future. The measures for matching patterns against extracted terms and relations need considerable improvement. The Jaccard string similarity measure and the simple scores of matched concepts and relations are too crude measures for making an accurate and reliable decision. There exist much research into the appropriateness of different methods, which will be taken more into consideration in future developments of the methodology. Today for example, small patterns are favoured by the metrics used, as well as short concept names. The effect of this were minimised in this experiment by using patterns of comparable size and reasonably short concept names, but it should be solved in other ways in the future.

Also a refinement of the ontology building phase is needed, especially when it comes to the heuristics used for inclusion or exclusion of concepts and relations. For this first experiment the heuristics were chosen without proper evaluation, simply on intuition. Also the resulting ontology needs to be more formally evaluated and validated. For the resulting ontology to be more useful there also needs to be a way of matching and including pattern axioms into the final ontology (in this first experiment all axioms applying to included concepts or relations were also included). Axioms are a very important part of an ontology so this should be a focus of future research.

A problem that could arise in the future is that some areas of enterprises that need to be present in an enterprise ontology cannot be found in any existing knowledge structure suitable for reuse as an ontology design pattern. Patterns could therefore also be created by experts and not only extracted from existing sources. The coverage of the resulting ontology needs to be considered and the possibility of finding patterns for all relevant aspects (both dependent and independent of the domain) need to be explored further. This is also true for the level of detail of the ontology. Perhaps a manual post-processing step is suitable for adding for example instances to the ontology, but this is outside the scope of our current research.

5 Conclusion and Future Work

In this paper we have shown how patterns (and other knowledge constructs) from other areas of computer science can be used as "templates" for creating ontology design patterns for automatic construction of enterprise ontologies. The resulting patterns are domain specific, in the sense that they are chosen with the intent of creating an enterprise ontology, but not domain specific when it comes to the business domain. This indicates that, much like in the database community, the ontology community could benefit greatly from reusing knowledge. The main aim is to reduce the time and effort of creating an enterprise ontology together with reducing the need for expert "ontology engineers" in small-scale application contexts.

For using the patterns, we propose an automatic construction method which matches the patterns to a set of extracted terms and relations. Patterns with enough matching concepts and relations are included in the resulting ontology, but first pruned of irrelevant or inappropriate concepts, synonyms and relations. This method is purely automatic but builds of course on the appropriate patterns being provided. The resulting ontology could be manually verified and refined as a post-processing step, if for example another level of detail (perhaps more domain dependent) is required, but this is not within the scope of our approach.

The main contribution so far by our research is the development of a method that continues where most semi-automatic ontology construction tools leave-off, namely constructing the entire ontology from the extracted concepts and relations. The resulting ontology will not be as tailored for the company in question as might a manual one, but it will be useful and mainly it will be fast and easy to create. This all fits to the requirements of a small-scale application context.

Still, there remains a great deal of refinement of the method in order for it to be used in a general case. The pattern catalogue needs to be broadened and enriched, the metrics and matching methods need to be evaluated and refined and the resulting ontologies need also to be subject of expert evaluations and validated more formally.

Acknowledgements. This work is part of the research project Semantic Structuring of Components for Model-based Software Engineering of Dependable Systems (SEMCO) based on a grant from the Swedish KK-Foundation (grant 2003/0241). Special thanks to Annika Öhgren and Kurt Sandkuhl for ideas and input during this work and also to three anonymous reviewers for valuable comments on how to improve this paper.

References

1. Guarino, N.: Formal Ontology and Information Systems. In: Proceedings of FOIS'98. (1998) 3–15
2. van Heijst, G., Schreiber, A.T., Wielinga, B.J.: Using explicit ontologies for KBS development. International Journal of Human-Computer Studies **46** (1997) 183–292
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-oriented Software Architecture - A System of Patterns. John Wiley & Sons, Chichester (1996)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
5. Hay, D.C.: Data Model Patterns - Conventions of Thought. Dorset House Publishing (1996)
6. Puppe, F.: Knowledge Formalization Patterns. In: Proceedings of PKAW 2000, Sydney, Australia, 2000. (2000)
7. Sutcliffe, A.: The Domain Theory - Patterns for Knowledge and Software Reuse. Lawrence Erlbaum Associates (2002)
8. W3C-SWBPD: Semantic Web Best Practices and Deployment Working Group. Available at: http://www.w3.org/2001/sw/BestPractices/ (2004)

9. Stuckenschmidt, H., Euzenat, J.: Ontology Language Integration: A Constructive Approach. In: Proceedings of the Workshop on Application of Description Logics at the Joint German and Austrian Conference on AI, CEUR-Workshop Proceedings. Volume 44. (2001)
10. Staab, S., Erdmann, M., Maedche, A.: Engineering Ontologies using Semantic Patterns. In O'Leary, D., Preece, A., eds.: Proceedings of the IJCAI-01 Workshop on E-business & The Intelligent Web, Seattle (2001)
11. Gangemi, A.: Some design patterns for domain ontology building and analysis. Available at: http://www.loa-cnr.it/Tutorials/OntologyDesign Patterns.zip, downloaded 2004-10-04. (2004)
12. Reich, J.R.: Ontological Design Patterns for the Integration of Molecular Biological Information. In: Proceedings of the German Conference on Bioinformatics GCB'99. (1999) 156–166
13. Blomqvist, E., Sandkuhl, K.: Patterns in Ontology Engineering: Classification of Ontology Patterns. In: Proc. of ICEIS2005 7th International Conference on Enterprise Information systems, Miami Beach, Florida (2005)
14. de Chalendar, G., Grau, B.: How to Classify Words Using their Context. In: Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management, EKAW2000, Juan-les-Pins, France, October 2000, Springer (2000) 203–216
15. Gamallo, P., Gonzalez, M., Augustinin, A., Lopes, G., de Lima, V.S.: Mapping Syntactic Dependencies onto Semantic Relations. In: 15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, France. (2002)
16. Sporleder, C.: A Galois Lattice based Approach to Lexical Inheritance Hierarchy Learning. In: 15th European Conference on Artificial Intelligence (ECAI'02): Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, France. (2002)
17. Silverston, L.: The Data Model Resource Book, Revised Edition, Volume 1. John Wiley & Sons (2001)
18. KAON: available at http://kaon.semanticweb.org/ (2005)
19. WordNet: available at http://wordnet.princeton.edu/ (2005) , downloaded 2005-04-14.
20. Fowler, M.: Analysis Patterns - Reusable Object Models. Addison-Wesley (1997)
21. Gangemi, A., Mika, P.: Understanding the Semantic Web through Descriptions and Situations. In: Proc. of the International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2003), Catania, Italy (2003)
22. Silverston, L.: The Data Model Resource Book, Revised Edition, Volume 2. John Wiley & Sons Inc. (2001)
23. Gardner, K., Rush, A., Crist, M., Konitzer, R., Teegarden, B.: Cognitive Patterns - Problem-solving Frameworks for Object Technology. Cambridge University Press (1998)
24. Maedche, A., Volz, R.: The ontology Extraction & Maintenance Framework Text-To-Onto. In: ICDM'01: The 2001 IEEE International Conference on Data Mining Workshop on Integrating Data Mining and Knowledge Management. (2001)
25. Cohen, W., Ravikumar, P., Fienberg, S.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico. (2003)
26. SecondString: available at http://secondstring.sourceforge.net/ (2005)

Automatic Ontology Extraction from Unstructured Texts

Khurshid Ahmad and Lee Gillam

Department of Computing, University of Surrey, Guildford, Surrey, GU2 7XH, UK
{k.ahmad, l.gillam}@surrey.ac.uk

Abstract. Construction of the ontology of a specific domain currently relies on the intuition of a knowledge engineer, and the typical output is a thesaurus of terms, each of which is expected to denote a concept. Ontological 'engineers' tend to hand-craft these thesauri on an ad-hoc basis and on a relatively small-scale. Workers in the specific domain create their own *special* language, and one device for this creation is the repetition of select keywords for consolidating or rejecting one or more concepts. A more scalable, systematic and automatic approach to ontology construction is possible through the automatic identification of these keywords. An approach for the study and extraction of keywords is outlined where a corpus of randomly collected unstructured, i.e. not containing any kind of mark-up, texts in a specific domain is analysed with reference to the lexical preferences of the workers in the domain. An approximation about the role of frequently used single words within multiword expressions leads us to the creation of a semantic network. The network can be asserted into a terminology database or knowledge representation formalism, and the relationship between the nodes of the network helps in the visualisation of, and automatic inference over, the frequently used words denoting important concepts in the domain. We illustrate our approach with a case study using corpora from three time periods on the emergence and consolidation of *nuclear physics*. The text-based approach appears to be less subjective and more suitable for introspection, and is perhaps useful in *ontology evolution*.

1 Introduction

Literature on intelligent systems invariably refers to a thesaurus of domain objects in the construction of knowledge bases. A 'thesaurus' suggests the existence of a range of words and phrases associated with a concept. The names of the objects form the *terminology of* the domain. The organisation of terminology is discussed under the rubric of *ontology*. Ontology is a branch of philosophy, and some philosophers believe that to understand what ***is*** in every area of reality one should look into the theories of sciences [1]. Quine, one of the proponents of modern ontology, has asked two key questions related to 'philosophy within science' [2]: (i) What are the conditions that lead to talking scientifically? (ii) How is scientific discourse possible? The answer in Quine, by no means exhaustive, is in the *ontological commitment* on the part of a scientist or group of scientists: the scientists observe physical phenomena and articulate them for others in linguistic entities (the controversial observation sentences). This sharing of common roots of reference – physical and linguistic – are, for

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1330 – 1346, 2005.

us, signs of being committed to the same set, or system, of concepts, and this is the basis of Quinian ontological commitment.

Researchers working on ontological 'engineering' tend to hand-craft thesauri on an ad-hoc basis and on a relatively small-scale. Laresgoiti et al discuss the ontology of an intelligent network control system in which the 'concepts' appear to be derived from an existing data dictionary [3]; Gómez-Pèrez, Fernández-López, and Corcho compose a *travel ontology* without reference to the source of knowledge that comprises 'concepts' such as "American Airlines Flight", "Iberia Flight", "Hotel" and "Japan Location", and a list of relations like "departurePlace" and "placed in" [4].

In this paper we argue that a more scalable, systematic and automatic approach to ontology construction is possible using methods and techniques of information extraction, corpus linguistics, and terminology science to examine archives of a specialist domain of knowledge. The methods and techniques enable the identification of the objects, processes, and concepts that describe an application area, a domain of specialist knowledge, or indeed a whole discipline. We describe a method to automatically extract key terms, and relationships between the key terms, from relatively-large corpora of unstructured, i.e. not markup up, text in particular specialisms, and how international standards (ISO) that have emerged from terminology science can facilitate construction of terminology databases and of the domain ontology. Our system generates hierarchically arranged terms from the text corpora that indicate the ontological commitment of researchers and practitioners of the domain. When represented in one kind of markup, the hierarchically arranged terms can be used as a basis for an ISO-standards conformant terminology, and when represented in an ontology interchange language they can be inspected and refined in an ontological engineering tool like *Protégé* [5]. The hierarchy can be augmented by linguistic pattern analysis to confirm, contract or expand elements of the hierarchy [6]. The method uses the frequency contrast between the general language of everyday use and the special language of the domain to identify key domain words [7], and expands the analysis to the discovery of consistently used patterns in the neighbourhood of these domain words. Our work relates to the emergence of new domains of knowledge and how scientists and philosophers construct an edifice of a new(er) branch of knowledge [8].

Our method identifies and extracts (candidate) terms and (candidate) ontologies from a set of written texts. The *candidate* nature of these results should suggest that we make no claims to treat subjective differences within ontology: on discovering *red wine*, we would, simply, present this as a class of *wine*, assuming this to be what has been discovered. We would leave it to the subject experts, and connossieurs, to debate whether *red* is the value of the attribute of *colour* of *wine*, and how *dry white wine* would now fit into this system. Such distinctions are essentially creative and mental tasks carried out with flair by human beings that is hitherto unmatched: even the most ambitious of ontology project does not attempt such a subjective qualification. Our work is no exception.

The results of our extraction – the terms and ontologies - can be validated by subject experts: the present case study is in Nuclear Physics and this subject has been studied by one of the authors (KA). Within the parameters of author-subjectivity, the results identified are in accord with current findings within the subject. Thesaurus

building systems can benefit from automated identification of terms and their inter-relationships within a specific domain of knowledge. The frequency of use of the terms, and the neighbourhood of the terms, is an indication of how knowledge in the specialism is organised by researchers and practitioners of that domain.

2 A Method for Extracting Ontology

By examining archives of a specialist domain of knowledge, we contend that one can find objects, processes, and concepts that describe an application area, a domain of specialist knowledge, or indeed a whole discipline. Approaches to the identification of domain-specific keywords – the terminology of the domain – generally rely on extensive prior linguistic knowledge and linguistic extraction techniques [9], [10], [11], [12], [13]: use of part-of-speech (POS) taggers predominates. Our treatment differs from these approaches in taking an initially statistical approach, which is suitable for subsequent augmentation using linguistic techniques. It uses the difference between the general language of everyday use and the special language of, for example, physics or philosophy or sewer engineering as its basis. This difference can be determined by comparing the relative frequency of words in texts with their relative frequency in the general language [14]. A special language is a subset, and sometimes an extension, of the natural language of a specialist. Scientists, amongst others, have to convince yet others of the value of their work: repetition is one of the rhetorical devices used in almost all enterprises for informing, exhorting or convincing others of the value of your own view. Evidence of the use of repetition can be found in repositories of specialist documents written in a natural language and adorned by images and tables of numbers. Authors of these specialist documents use a small vocabulary very productively: names of objects are used in singular and plural; a smaller number of words are used to make many of the compound terms and phrases. For example, in a research paper about modern nuclear physics one will find that the term *nucleus* is used 600 times more frequently in the subject than, say, in the 100 million-word British National Corpus – a representative sample of English language [15]. Gillam has refined and extended this contrast, and created a system that generates hierarchically arranged terms indicating ontological commitment within a domain [5].

The method is based on Quirk's notion that frequency of use of words correlates with acceptability of those words as part of the vocabulary [16]:33. The BNC is used as a reference collection of general language and we use the *weirdness index* [14] that has been adapted by smoothing [17], to seed a collocation extraction technique [18]. This analysis produces a hierarchy of terms/concepts via semantic inclusion. The hierarchy can be augmented by a linguistic pattern analysis that may confirm, contract or expand elements of the hierarchy (see [6] for details). By reference to international standards (ISO) for terminology, we can facilitate the construction of terminological resources that have a potentially wider scope of use as thesauri or ontologies: the hierarchy can be exported to an ontological engineering system like *Protégé*. Such a terminology/thesaurus/ontology is then suitable for validation by experts. The algorithm for this is shown in Fig.1.

1. **COLLATE** Text Corpora:
   a. Obtain a general language corpus $S_{General}$ comprising $N_{General}$ tokens
   b. Create a text corpus of documents in a specialist domain $S_{special}$ with $N_{Special}$ tokens
2. **COMPUTE** distribution of all 'words'
   FOR i=1 to $N_{Special}$ /* Compute frequencies of use of single words */
   w = token$_i$;
   IF w ∉ words
   THEN words:= words ∪w & frequency(w)=1
   ELSE frequency(w) = frequency(w)+1
   NEXT i
   FOR i=1 to #words /* Extract frequency f(w) of single word $\mathbf{w_j}$ from $S_{General}$ and $S_{Special}$ */
   weirdness(w_j) := ($f_{Special}$ (w_j)* $N_{General}$)/($f_{General}$ (w_i)+1) * $N_{Special}$)
   NEXT i
   avg_f:=($\sum f_{Special}(w_i)$)/ $N_{Special}$; $\sigma_{frequency}$:=($\sum(f_{Special}(w_i)$- $avg_f)^2$/ ($N_{Special}$*($N_{Special}$-1))
   avg_{weird}:= ($\sum$weirdness(w_i))/$N_{Special}$; σ_{weird}:=($\sum$(weirdness(w_i)-$avg_{weird})^2$/ ($N_{Special}$*($N_{Special}$-1))
3. **EXTRACT** 'keywords'
   FOR i=1 to #words /* Compute *z-scores*/
   $z_{frequency}(w_i)$:= ($f_{Special}$ (w_i)- avg_f)/ $\sigma_{frequency}$
   $z_{weird}(w_i)$:= (weirdness(w_i)-avg_{weird})/ σ_{weird}
   IF $z_{frequency}(w_i) > \tau_{frequency}$ & $z_{weird}(w_i) > \tau_{weird}$
   THEN keywords:= keywords∪w_i
   NEXT i
4. **EXTRACT** significant collocates of keyword
   FOR i=1 to #keywords /* Build hierarchy */
   FIND $keywords_m$ in $S_{special}$
   FOR j=-5 to +5, j ≠ 0; $f_{coll}(keyword_i, w_{i+j}) := f_{coll}(keyword_i, w_{i+j})+1$;**NEXT** j
   IF $y(f_{coll}(keyword_m, w_{m+k})) > \tau_{collocation}$
   THEN collocations:= collocations∪$(keyword_m, w_{m+k})$
   NEXT i

Fig. 1. An algorithm for extracting 'keywords' and collocates using given threshold values (τ);y is a collocation statistic due to Smadja [17]. Iterative application of step 4 using sets of collocations results are used to produce the hierarchy.

Elsewhere, we have used Zipf's Law [19] to demonstrate similarities in the patterns of frequency of words used in different specialisms: the approach may be generalisable to other specialisms as although the words differ the patterns of use are similar. In this paper, we apply the method to three sub-corpora of nuclear physics to identify changes in the ontology over time, or perhaps *ontology evolution* [20].

3 Text-Based Ontology: A Nuclear Physics Case Study

3.1 A Note on the Domain: Nuclear Physics

The evolution of nuclear physics in the 1900's provides an example of how concepts are re-defined (semantic shift) and terms re-lexicalised. Papers start to emerge early in the 20th century describing that the 'indivisible' *atom* was 'divisible' after all and contained a positive *nucleus* surrounded by negatively charged *electrons*. Ernst Rutherford conducted the first of the pioneering experiments in the emerging field of *nucleus physics* (*sic*.) and published a number of papers in this emergent field. Ruther-

ford was concerned about the *deflection* of alpha-particles when scattered on selected targets and he noted the *deflexions* using *scintillation counters.* In his later years, he worked to artificially transmutate one element into (many) others by bombarding the element with a beam of *particles* and thus found *artificial radio-activity*. Niels Bohr is regarded as one of the pioneers of modern quantum theory and he produced a model of a stable atom in which the negatively charged particles (*electrons*) precess around the positive nucleus in a *stable orbit* – that is, despite traversing in an electromagnetic field, due to the nucleus, the electrons do not radiate energy. Subsequently, Bohr was involved in nuclear *fission* and produced a model of how a Uranium nucleus, when bombarded by *neutrons*, will split into two fragments, releasing massive amounts of energy. Rutherford and Bohr's work led us to the modern conception of a nucleus comprising the positively charged *protons* and the neutral *neutron* together very compactly by exchanging elementary particles called *mesons.* A system of concepts related to the 'new' structure of matter, in many ways analogous to the planetary system, was established through the frequent use of words (terms) in physics then, especially *atom*, but with a changed meaning, and the adoption of terms from other disciplines, including *nucleus* from botany. The frequent use of these two keywords on their own and in compound terms reflects the ontological commitment of the then modern physicist.

The term *nuclear physics* was first used after the 2nd World War, and due both to its peaceful uses and destructive potential it has received substantive funding and a number of researchers are involved in this field. As time has passed, the subject has focussed on deeper and deeper studies of *nuclear matter*, and one of the current exciting developments is in the field of *exotic nuclei*: nuclear physicists can create highly unstable nuclei and extremely short-lived nuclei in laboratory conditions, and study the behaviour of such nuclei to measure nuclear forces and determine the *structure* of nuclei. Amongst the more recent discoveries are the *halo nuclei* – where *neutrons and protons* are loosely bound to a nucleus much like a halo surrounds our Moon. New structures have been discovered that have been explained by referring to a knot-like structure – the so-called *Borromean rings*. Here, physicists are introducing a new method of studying the structure, redefining the concept of *nucleus* as a stable entity, and then describing newer forms of highly unstable matter – a highly transient element comprising a *halo* around an otherwise *stable core*.

Our task is to investigate whether such key concepts, that would be articulated through frequently used keywords, are automatically extracted from a diachronic study of the texts produced in the three periods in the development of nuclear physics. To this end we have analysed three sets of texts: one written by Rutherford and his co-authors, another by Bohr, and a third that is a random sample of texts published between 1994 and 2004. Rutherford's texts are exclusively from journals; for Bohr we have also included letters he had written to his brother (another physicist) and his wife (non-physicist). The modern nuclear physics texts comprise journal papers, popular science articles and conference announcements. For our comparisons, we use the BNC as a common reference point (reference corpus). See Table 1 for details of these 4 corpora.

Table 1. Composition of the 4 text corpora

Subject	No. of texts	Time Period	No. of Tokens	Text Types
1. Nuclear Physics (*Rutherford*)	17	1908-1932	61,035	Journal Papers (JP)
2. Nuclear Physics (*Bohr*)	16	1920-1950	101,201	JP; Letters (LT)
3. Nuclear Physics (*modern*)	157	1994-2004	564,415	JP, Conference Announcements, Popular Science, Academic Course Details
4. British National Corpus	4124	1960-1993	100,106,029	Various including extracts from newspapers, specialist periodicals, journals and popular fiction in subjects including natural science, social science, commerce and leisure.

3.2 Automatic Extraction of Single-Word Terms and Diachronic Variance

The statistic we use extensively is the *weirdness index* (eqn. 1), a measure of the use of a word in special language compared to its use in a representative corpus of general language texts:

$$weirdness = \frac{N_{GL} f_{SL}}{(1 + f_{GL}) N_{SL}} \tag{1}$$

where f_{SL} is the frequency of word in the specialist corpus, f_{GL} is its frequency in BNC, and N_{SL} and N_{GL} are the token counts of the specialist corpus and the BNC respectively. The disproportionately used words are more likely to be terms of a specialist domain and terms are used to denote *concepts* [7].

Consider the distribution of 10 most frequently used words in each of the three corpora, excluding the so-called closed class or *stop* words (e.g. *the, a, an, but, if*….) as shown in Table 2. The most frequent words in Rutherford include *particle(s)*, *atoms* and *nucleus*. These words are 'disproportionately' used by Rutherford when compared with typical text in English – he uses *particle* 629 times more frequently than is used in the British National Corpus, *atom* 896 times more frequently and *nucleus* 841 times more frequently. There are clues here of the famous scattering experiments – where Rutherford measured the *range* (22 times more frequent) of *alpha* (485 times more frequent) particles emitted by a radioactive source (in *centimetres* or *cm*). The emphasis in Bohr is on the *electrons* (1652 times more frequent), and (the electron) *orbits* (1204 times more frequent); *nucleus* is used less disproportionately in Bohr (652 times) than in Rutherford (841 times). Bohr's more frequent use of *electrons* in an *orbit* should not detract from the fact that the orbit was around the *nucleus*. The word *nucleon* (a hyponym for *proton* and *neutron*) is amongst the most disproportionately used – over 36410 times more frequent in our corpus than in the BNC; *energy* (and its unit *mev –million electron volts*) is amongst the most frequently used. The frequency of *cross, section* and *scattering,* reflects the use of the term *cross-section* in nuclear physics where it is used to refer to a measure of the probability of a *nuclear* reaction; *scattering cross-section* is used in determining the determining the *structure* of *nuclei*.

The lexical choice of modern nuclear physicist has changed over time and is principally shown by the more proportionate use of the words *atom, atoms* and *atomic*

Table 2. Distribution of 10 most frequent single words (terms) in our three corpora – with number in parentheses indicating the rank of the word in a complete wordlist of the corpus

Rutherford			Bohr			Modern		
Token	***Rel. freq***	***Weirdness***	***Token***	***Rel. freq***	***Weirdness***	***Token***	***Rel. freq***	***Weirdness***
particles (10)	1.05%	629	electrons (11)	1.03%	1652	energy (21)	0.48%	39
atoms (18)	0.73%	691	atom (20)	0.60%	1084	neutron (30)	0.34%	1390
number (21)	0.61%	12	electron (25)	0.46%	488	nuclei (36)	0.30%	972
particle (22)	0.59%	847	nucleus (27)	0.44%	652	nuclear (38)	0.29%	36
nucleus (23)	0.56%	841	energy (28)	0.42%	34	cross (42)	0.28%	38
alpha (25)	0.54%	485	theory (31)	0.38%	29	mev (46)	0.27%	7193
atom (27)	0.49%	896	number (32)	0.36%	7	state (47)	0.27%	7
cm (28)	0.48%	231	orbits (33)	0.34%	1204	body (50)	0.26%	10
range (30)	0.45%	22	elements (35)	0.33%	52	nucleon (51)	0.25%	36410
hydrogen (32)	0.42%	348	atomic (39)	0.29%	264	scattering (52)	0.25%	497

– used around 30 times more in the Modern corpus compared to 800 times or more as was the case for Rutherford and Bohr.

Frequency varies considerably across a corpus of words. Some words are used very frequently and others very infrequently: *neutron* is used 1944 times in the 564115 word Modern nuclear physics corpus; *neutrons* 549 times; *dineutron* 44 times; *multineutron* 5 times; and *tetraneutron, trineutron,* and *neutronization* only once. The average frequency in the Modern corpus is 29.18 with a standard deviation of 445. Much the same can be said about the variation in weirdness across the corpus – the average weirdness is 226.5 with a standard deviation of 1307. The standard deviation of frequency in the British National Corpus is 11,000. Instead of using frequency and weirdness as a measure of disproportionate use, we calculate the *z-scores* (eqn. 2) for both frequency (*f*) and weirdness (*w*):

$$z_i(x) = \frac{(x_i - \bar{x})}{\sigma_x} \tag{2}$$

We can now specify a minimum value of z-scores for frequency and weirdness and use this to automatically select only those words that are above this value, removing the subjective treatment of importance of these words. For a threshold of 0.25, that is all words with a frequency that is above the average frequency (and weirdness) by a margin of a quarter of a standard deviation, we find: Rutherford's corpus has only 8 words that satisfy that criteria in 3446 unique words; Bohr's corpus has 17 amongst 4145 words; and the Modern corpus has 27 words amongst 19341 unique words. Table 3 shows the 'new' selection together with words that were selected on the use of statistical criteria.

Table 3 shows a trace of the ontological commitment of workers in nuclear physics over a 100 year period. This is portrayed also in Fig. 2. The commitment to study energy – one of the three concepts physicists study, the other two are force and mass – and the nucleus remains the same over the century. What changes over the period is the enthusiasm for the study of unstable systems, hence the word/term halo, by way of laboratory created nuclei – and we have the word/term projectile and a related word breakup (referring to the break up of nuclei). This conclusion is all the more gratifying as our system has no domain knowledge per se. Note that the single words (and the related concepts) denote generic concepts and it is in the specialisation of these potential terms that one can see ontologically motivated hierarchies. This we discuss next.

Table 3. Distribution of (8 or 10) most frequent words that satisfy the 0.25*standard deviation criteria across the three corpora. Words in bold are those that were identified manually in Table 2 – the combination removes from consideration those words with low weirdness. Underlining denotes words shared across these topmost – *nuclei* across all 3 with varying importance, *nucleus* and *atom* across Rutherford and Bohr.

Rutherford			**Bohr**			**Modern**		
	z (*f*)	**z (*w*)**		**z (*f*)**	**z (*w*)**		**z (*f*)**	**z (*w*)**
particle	2.45	0.29	**electrons**	4.37	3.18	**neutron**	4.30	0.89
nucleus	2.35	0.29	**atom**	2.50	2.02	**nuclei**	3.71	0.57
atom	2.04	0.32	**electron**	1.90	0.81	**mev**	3.39	5.33
scintillations	1.12	36.9	**nucleus**	1.80	1.15	**nucleon**	3.16	27.68
nuclei	0.92	0.26	**orbits**	1.37	2.27	halo	2.52	0.78
helium	0.89	0.40	**atomic**	1.17	0.36	projectile	1.47	2.27
radium	0.88	3.07	hydrogen	1.05	0.27	proton	1.46	0.27
deflexion	0.30	25.4	nuclei	0.87	1.31	6he	1.39	21.80
			quantum	0.87	0.36	coulomb	1.17	5.17
			stationary	0.80	0.86	breakup	1.12	1.42

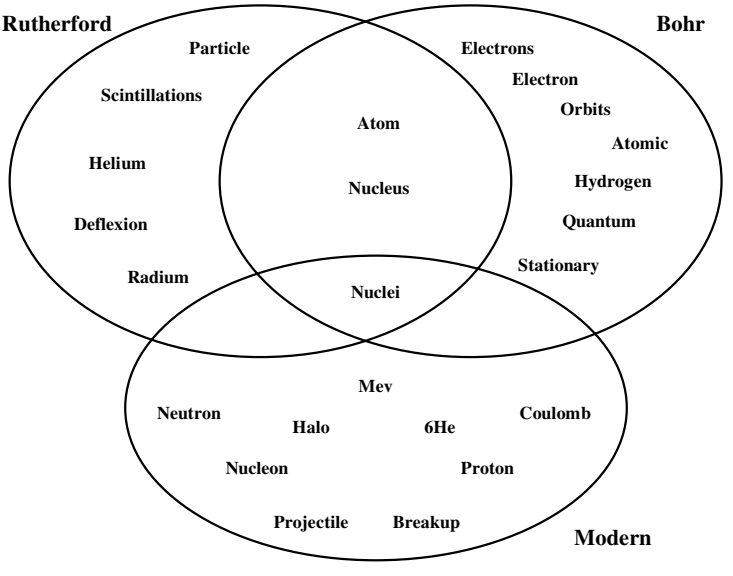


Fig. 2. Lexical sharing amongst the topmost automatically selected words from the three corpora

3.3 Automatic Extraction of Compound Words and Diachronic Variation

What is more important, perhaps, than these frequent words alone is the manner in which the frequent words produce expressions comprising multiple words. The mul-

tiword terms help to specify a, perhaps more complex, generic concept – *nuclear energy* is a form of *energy* and is different from *electrical* or *heat energy*; *nuclear reaction* is a kind of *reaction* and *direct nuclear reaction* is a *nuclear reaction* that is different from *compound nuclear reaction*.

Returning to the application of the method outlined in Section 2, for each collection we use a z-score value, manually assigned or automatically derived, to systematically determine the number of keywords for further analysis. Collocation analysis is commonly used in corpus linguistics to identify words that occur frequently within a given neighbourhood of each other, and that are used to convey specific meanings. Corpus linguists make frequent use of *mutual information* and *t-score* statistics to determine the significance of bigrams – two words within the neighbourhood. We have found these metrics to provide limited information with regard to the importance of the individual positions within the neighbourhood. Hence, we have applied the analysis due to Smadja who has argued that significant collocates of a word are within a neighbourhood of five words either side of the word (the *nucleate*) denoted as L1-L5 (left) and R1-R5 (right); Smadja has outlined metrices for quantifying the strength of the collocation including one that isolates *peakedness* (U) of the collocation in the various positions of the neighbourhood together with a z-score: significant collocates have a U-score >10 and z-score > 1. We have implemented Smadja's method and are able to automatically extract collocation patterns. Consider five of the 7 significant collocates of *nucleus* (Table 4a) and the dominant positions of collocates (Fig. 3.).

Table 4a. Selected collocates of *nucleus* in Rutherford using {U, k}= {10,1}

Collocate	L5	L4	L3	L2	L1	R1	R2	R3	R4	R5	U	k
charge	1	0	19	0	0	17	1	1	2	3	47.2	11.8
helium	1	0	0	0	**19**	0	0	0	4	0	32.0	6.1
hydrogen	2	0	1	0	**14**	0	1	3	0	1	16.4	5.6
atom	1	5	1	0	0	0	0	3	**13**	1	14.8	6.1
theory	1	0	1	0	0	**13**	0	0	1	0	14.6	3.9

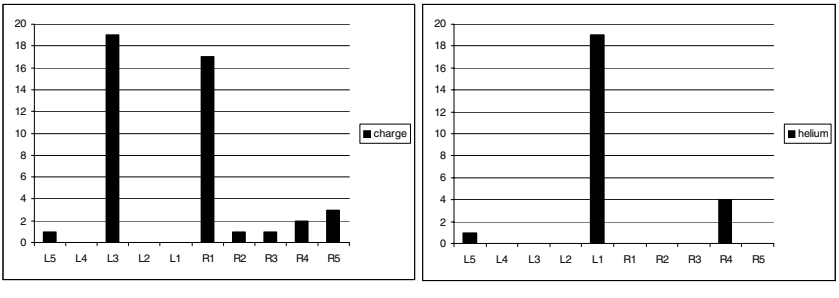


Fig. 3. Collocations in Rutherford with *nucleus* of *charge* and *helium*. Note that dominant positions for *charge* collocating with *nucleus* are L3 (*charge* [] [] *nucleus*) and R1 (*nucleus charge*), while the dominant position for *helium* collocating with nucleus is L1 (*helium nucleus*). Smadja's "z-score k" [18] would select these three patterns for further analysis.

The focus in Rutherford is on the term *nucleus charge* (44 collocates) and there is an enumeration of the nucleus of different elements (*hydrogen* and *helium nucleus)* and a reference to *nucleus theory.* The focus in Bohr is rather different (Table 4b, Fig. 4.) as shown by five (of the 7) collocates where we have *positive nucleus* and *electron(s) +X (+Y) nucleus* are amongst the more common collocates:

Table 4b. Selected collocates of *nucleus* in B**ohr** using {U, k}= {10,1}

	L5	**L4**	**L3**	**L2**	**L1**	**R1**	**R2**	**R3**	**R4**	**R5**	**U**	**k**
round	0	0	5	**38**	0	0	0	0	0	2	127.1	7.4
electron	9	19	**23**	0	0	0	0	4	0	0	68.5	9.2
rotating	0	4	**23**	0	0	0	0	0	2	0	46.5	4.6
electrons	7	**17**	15	1	0	0	5	5	5	6	30.3	10.3
positive	0	5	0	0	**17**	0	2	2	2	0	24.8	4.4

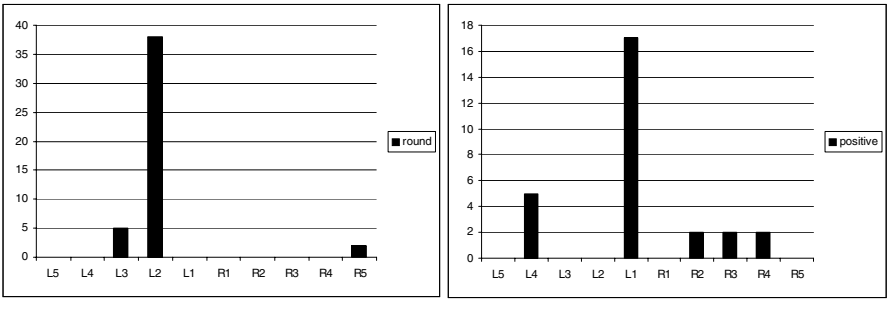


Fig. 4. Collocations in Bohr with *nucleus* of *round* and *positive*. Note that dominant position for *round* collocating with *nucleus* is L2 (*round* [] *nucleus*), while the dominant position for *positive* collocating with nucleus is L1 (*positive nucleus*). Smadja's "z-score k" would again select these two patterns for further analysis.

The collocation patterns in modern nuclear physics are somewhat different – we are in a period where the concept of a *nuclear atom* is established, and *atom* itself goes unmentioned – the collocate *atomic nucleus* only occurs 16 times in the 564,115 word modern nuclear physics corpus and *atomic nuclei* 52 times. What we find instead (Table 4c, Fig. 5.) is the phraseology of reacting nuclei (*target* and *residual*) and unstable nuclei *He*lium-*6* and *Li*thium-*11* nuclei, natural Helium has 4 nucleons and Lithium has 8.

This kind of analysis is of interest for discovering increasingly more complex *concepts* within specialist text collections. For example, the *halo* was found, as fifth most important keyword, in the modern collection; collocations (Step 4 in the Algorithm) around *halo* include: *halo nuclei*; *halo nucleus*; *neutron halo*; and *halo*

neutrons. We further discover the *weakly bound neutron halo* (*f*=2), but the most expanded tree forms under the *halo nuclei*.

Table 4c. Selected collocates of *nucleus* in Modern Nuclear Physics Corpus using {U, k}= {10,1}

Collocate	L5	L4	L3	L2	L1	R1	R2	R3	R4	R5	U	k
target	3	2	0	1	**114**	1	0	5	11	5	1117	22.4
halo	4	4	2	1	**90**	0	2	5	4	5	684	18.4
compound	0	1	0	0	**41**	0	1	0	1	2	148	7.0
residual	0	0	1	0	**25**	0	0	1	0	0	55.4	3.9
borromean	0	1	0	2	**22**	0	0	0	0	0	42.7	3.6

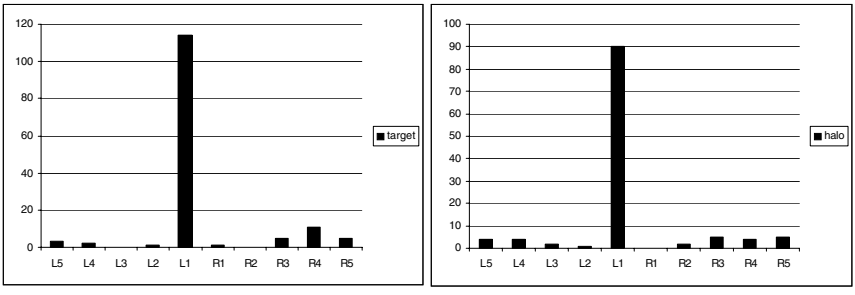


Fig. 5. Collocations in Modern corpus with *nucleus* of *target* and *halo*. Note that dominant positions both collocations is L1 (*target nucleus*, *halo nucleus*), both of which the z-score would select for further analysis.

The various patterns identified can be used to produce collocational networks [21]. Since we believe that many of these networks may provide evidence of semantic inclusion, we assert *isa* relationships between the words and their collocates and produce hierarchies from these collocational networks (see, for example, [6]). The hierarchies can be used in combination with international standards for the production of terminology interchange formats (using ISO 12620 and ISO 16642) that can be used to populate terminology databases [22]. These formats express concepts, conceptual relations, and provide for items of administrative information, including versioning, and for items of documentary information including sources and contexts of use. There is some degree of overlap with these formats and the so-called ontology exchange languages, and hence the results can also be exported to an ontological engineering tool (Protégé), using a semantic schema such as the Web Ontology Language (OWL), to facilitate knowledge base development and visual inspection and refinement: a domain expert can quickly make on-line corrections to the hierarchy in collaboration with a knowledge engineer using the ontology tool. The role of the knowledge engineer in our method relates exclusively to the construction of the

knowledge base and avoids any intuitive input on their part. The (partly-pruned) ontological commitment of modern nuclear physicists in relation to *halo nuclei*, alluded to above, can be seen in Fig. 6, as drawn using the Protégé component ("Tab") OntoViz.

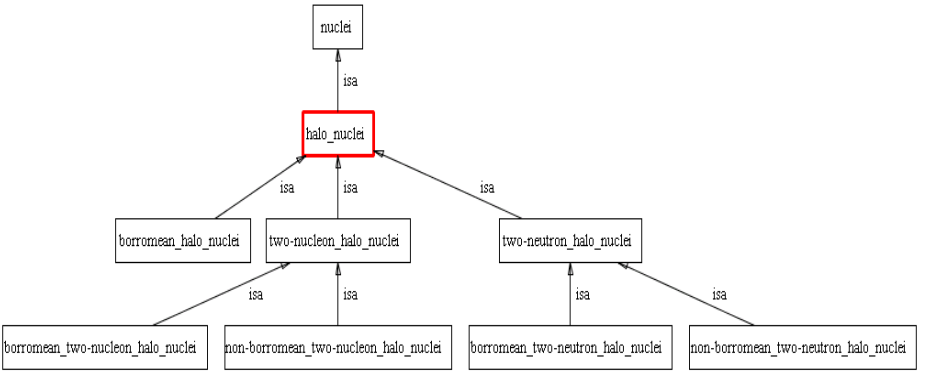


Fig. 6. A candidate hierarchy showing different types of *halo nuclei*

3.4 Ontology Evolution

Ontology Evolution has been discussed with reference to a wine ontology [20], [23]. The creation of a log of the evolution is emphasised using a differencing operation between ontologies. Without reference to the sources of the original knowledge that went into the production of the ontology, the reasons for the different ontologies may not be easy to discover. The importance of some terms at one time versus other terms at another would result in quite a large log file of changes. Whether such approaches are able to capture, for example, the *atom* changing semantically from being *indivisible* to *divisible* is not clear with from such small examples.

We use the term *ontology evolution* to refer to a change in a specialism's existing repertoire of concepts over a period of time. As the subject of nuclear physics has evolved over time, so the record of the ontological commitment of the workers has evolved. Over a short period of time (c. 25 years from 1900), the concept of a unitary atom was rejected, its constituents were identified experimentally and elaborated theoretically, and by the late 1930s a new field of physics – nuclear physics – had emerged. Researchers now seldom use the term *atom* and are careful in the use of the now generic headword *nucleus*. : nuclear physicists invariably use a qualificatory adjective or another noun when using *nucleus* and its derivatives *nuclear* and inflected form *nuclei*.

We can measure changes in ontological commitment over time, particularly with reference to the changes in importance attributed to these commitments by the authors, by calculating the weirdness index again, this time within the sub-corpora of the specialism. Words that were automatically identified as important for Bohr and that were almost irrelevant for Rutherford, are indicative of changes in the subject and can be identified, again, by high frequency and weirdness values (Table 5a).

Table 5a. Frequency and weirdness values for words of importance in the Bohr corpus, but of low importance in the Rutherford corpus

Word	f (Bohr)	f (Ruth)	Weirdness
states	231	0	139
stationary	210	0	127
ring	173	0	104
rings	104	0	63
configuration	174	1	52
configurations	79	0	48
quanta	143	2	43
fission	63	0	38
bound	155	2	31
Orbits	342	6	29

The 'ideas' that are either common-place within the subject, or have become 'suppressed' for other reasons, will occur with somewhat lower frequency and weirdness values (Table 5b): for Bohr, the atom is slightly more important (> 1) than for Rutherford. The remainder of these words are of lesser importance (< 1), with *radium*, *deflexion* and *scintillations* finding little or no interest at all. Those items that were of importance in Rutherford's work are not the subject of study for Bohr, although the field of study effectively remains the same. However, if they are so common-place that they no longer necessitate description, they may not be in the newer version of the ontology because that which is understood does not to be discussed.

Table 5b. Frequency and weirdness values for words of importance in the Rutherford corpus, but that have low importance in the Bohr corpus

Word	f (Rutherford)	f (Bohr)	Weirdness
particles	640	147	0.14
atoms	445	223	0.30
particle	358	91	0.15
nucleus	344	442	0.77
atom	302	606	1.21
scintillations	174	0	0
nuclei	146	227	0.93
helium	142	102	0.43
radium	140	3	0.01
deflexion	60	1	0.01

The same comparison can be made for the Modern Physics corpus, where we find that notions of *stationary*, *ring*, *rings*, *quanta*, *fission* and *orbits* have either become fundamental to the subject, or are suppressed for other reasons (Table 5c).

Since the most significant terms, according to our method, are changing in their importance over time, the challenge and need for managing ontology evolution, and for managing the input documents that form a part of this process, becomes significant. Terminology interchange formats, defined according to the International Standards, make provsision for the management of such reference material.

Table 5c. Frequency and weirdness values for words of importance in the Bohr corpus, but that have low importance in the Modern corpus

Word	f (Bohr)	f (Modern)	Weirdness
states	231	202	0.87
stationary	210	0	0
ring	173	0	0
rings	104	0	0
configuration	174	11	0.06
configurations	79	30	0.37
quanta	143	0	0
fission	63	0	0
bound	155	117	0.75
orbits	342	0	0

4 Discussion

We have automatically extracted hierarchical trees of terms from collections of natural language texts that demonstrate evidence of, and change in, the ontological commitment in physics over a period of time. We have demonstrated the efficacy of the automatic extraction method in a number of domains including nano-technology, forensic science, philosophy of science, financial investment, and epidemiology (for example, see: [24], [25], [26]). The principal inputs to our system are the collection of texts in an arbitrary domain and the list of general language words. Both the terminology and ontology have a reference point – the text collection: this contrasts with the rather ad-hoc work usually reported in ontological engineering.

Advocates of part-of-speech (POS) tagging might suggest that we ignore POS information at our peril. We have analysed the Rutherford corpus using the Brill tagger in its default (untrained) state. Rutherford's 8 keywords from Table 1 occur as either a kind of noun (NN, NNS or NNP), or as an unknown (UNK). At this level, there would appear to be negligible gain from POS tags *per se*.

Table 6. Part-of-speech information for the 8 keywords selected from the Rutherford corpus (Table 1)

Word	NN	NNS	NNP	UNK
atom	298			4
deflexion				60
helium	141		1	
nuclei		139	7	
nucleus	339		3	
particle	323			35
radium				140
scintillations		173	1	

If we consider the *nucleus* in Rutherford, the three patterns we find with POS information are: *hydrogen nucleus* = NN NN*; helium nucleus* = NN NN*, nucleus theory* = NN NN. While this again imparts little additional information and, indeed, one may argue about meronymy since the *nucleus* is a part of *hydrogen*, this may provide some limited evidence useful for mutual validation of our results. Where the expansion is adjectival (for example, *swift atoms* = JJ NNS) determination of whether the relationship is hierarchical, or whether this should be considered as an **attribute** or **value** remains subjective: again, is *red wine* a kind of *wine*, or is *red* a value of the *colour* of *wine*? Such judgements need to be subjectively made, and our objective method does not make provsision for such decisions.

We have explored Hearst's work [27] for augmenting our ontologies, combining phrases including *such as* with our extracted terms and with POS information to enable the bootstrapping of ontologies from unstructured texts [6]. The prior identification of terminological data may circumvent the need for training the POS taggers, which can now be used against the more grammatical elements of the texts. Consideration of the expansions of phrase patterns, for example: *properties of* [] *such as* …., or *characteristics of* [] *such as*…., where [] denotes a term, may provide for further population of the ontology. There are question marks over the scalability of approaches that use POS tagging since the taggers generally require training in new specialisations. The quality of the results is, then, a function of the training plus the coverage of the rules used for identification. Using our expanded method it may be possible to reduce the dependency on the POS tagger.

In other analysis, we have discovered phrases such as *conventional horizontal-type metalorganic chemical vapor deposition reactor*, *ridge-type ingaas quantum-wire field-effect transistors* and *trench-type narrow ingaas quantum-wire field effect transistor*. We are detecting these phrases without the need for the prior linguistic knowledge that goes into the expectation of existence of specific combinations of POS tags. It may be possible, however, to use statistical validity as a means to generate POS patterns that could be used to identify further elements of the ontology, and may be worth considering in further work. In addition, since we are in a particular specialism, we do not make consideration of different senses (concepts) being indicated by the same term. Indeed, in coining terms and retrofitting new exclusive sense to extant terms, scientists restrict the terms to a single sense. For example, although *nucleus* was, most likely, adopted from biology, it is highly unlikely that it would be used in a biological sense within these corpora. Such considerations may be of value where interdiciplinarity is evident – e.g. biochemistry, although we suspect that the discipline would soon try to remove such ambiguity to ensure good science.

We distinguish between ontology – as *the essence of being* – and ontological commitment – an extant commitment of a group of workers in a specialism as to what that *essence* is and how the essence manifests itself. The commitment shows changes over a period of time, and the change is recorded, howsoever incompletely, in the texts produced by the specialists. What we produce is candidate terminology and ontological commitment, and the statistical metrics we have used, *weirdness* and *collocation strength* metrics – candidate has to be verified and validated by domain experts. Our inputs and outputs are different from other reported systems dedicated to the identification and visualisation of ontology in a specific domain: as compared to

other workers in ontology and terminology engineering, we rely far less on our intuition and significantly more on the evidence produced by the domain community. We have presented an algorithm that encompasses the whole life cycle: from the automatic extraction of terms in free texts and onto systems' asserted knowledge representation for automatically populating knowledge bases. The above work will be boosted further by our current efforts in metadata standardisation [28].

Acknowledgements. This work was supported in part by research projects sponsored by the EU (LIRICS: eContent-22236) and by UK research councils: EPSRC (REVEAL: GR/S98450/01). We would like to thank the anonymous reviewers for their comments which we have taken into account in producing this version of the paper.

References

1. Quine, Willard, van Omran.: Theories and Things. Cambridge (Mass) & London: The Belknap Press of Harvard University Press (1981).
2. Orenstein, A.: Willard Van Orman Quine. Twayne, Boston: G. K. Hall. (1977).
3. Laresgoiti, I., Anjewierden, A., Bernaras,A., Corera, J., Schreiber, A. Th., and Wielinga, B. J.: Ontologies as Vehicles for reuse: a mini-experiment. KAW. http://ksi.cpsc.ucalgary.ca/KAW/KAW96/laresgoiti/k.html. (1996).
4. Gómez-Pèrez, Asunción, Fernández-López, Mariano., & Corcho, Oscar.: Ontological Engineering. London: Springer-Verlag. 2004.
5. Gillam, L.: Systems of concepts and their extraction from text. Unpublished PhD thesis, University of Surrey. (2004). http://portal.surrey.ac.uk/pls/portal/docs/PAGE/COMPUTING/PEOPLE/RESEARCHERS/GILLAM/PUBLICATIONS/PHD.PUBLISH.PDF
6. Gillam, L., Tariq, M. and Ahmad, K.: Terminology and the Construction of Ontology. Terminology. John Benjamins, Amsterdam. Terminology 11:1 (2005), 55–81.
7. Ahmad K.: Pragmatics of Specialist Terms and Terminology Management. In (Ed.) P. Steffens. Machine Translation and the Lexicon. Heidelberg (Germany): Springer. (1995), pp.51-76
8. Ahmad K. and Mussachio, M.T.: Enrico Fermi and the making of the language of nuclear physics. Fachsprache 25 (3-4). (2003), pp120-140.
9. Maedche, A. and Volz, R.: The Ontology Extraction and Maintenance Framework Text-To-Onto. Workshop on Integrating Data Mining and Knowledge Management. California, USA (2001)
10. Maedche, A. and Staab, S: Ontology Learning. In S. Staab & R. Studer (eds.): Handbook on Ontologies in Information Systems. Heidelberg: Springer (2003).
11. Faure, D. and Nédellec, C.: Knowledge Acquisition of Predicate Argument Structures from Technical Texts Using Machine Learning: The System ASIUM. LNCS 1621. Springer-Verlag, Heidelberg. (1999) 329-334.
12. Faure, D. and Nédellec, C.: ASIUM: Learning subcategorization frames and restrictions of selection. In Y. Kodratoff, (Ed.), 10th Conference on Machine Learning (ECML 98), Workshop on Text Mining, Chemnitz, Germany. (1998).
13. Mikheev, A. and Finch, S.: A Workbench for Acquisition of Ontological Knowledge from Natural Text. In Proc. of the 7th conference of the European Chapter for Computational Linguistics (EACL'95), Dublin, Ireland. (1995) 194-201.

14. Ahmad, K. and Davies, A.E.: Weirdness in Special-language Text: Welsh Radioactive Chemicals Texts as an Exemplar. Internationales Institut får Terminologieforschung Journal 5(2). (1994) 22-52.
15. Aston, G. and Burnard, L.: The BNC Handbook: Exploring the British National Corpus. Edinburgh University Press (1998).
16. Quirk, R.: Grammatical and Lexical Variance in English. Longman, London & New York (1995)
17. Gale, W. and Church, K. W.: What's wrong with adding one? In Oostdijk, N. and de Haan, P. (eds.): Corpus-Based Research into Languge: In honour of Jan Aarts. Rodopi, Amsterdam (1994), 189-200
18. Smadja, F.: Retrieving collocations from text: Xtract. Computational Linguistics, 19(1). Oxford University Press. (1993), 143-178
19. Zipf, G.K.: Human Behavior and the Principle of Least Effort. Hafner, New York. (1949).
20. Noy, N.F.; Musen, M.A.: Ontology versioning in an ontology management framework. Intelligent Systems 19 (4). IEEE Press (2004), 6-13
21. Magnusson, C. and Vanharanta, H.: Visualizing Sequences of Texts Using Collocational Networks. In Perner, P. and Rosenfeld, A. (Eds): MLDM 2003, LNAI 2734 Springer-Verlag, Heidelberg. (2003) 276-283
22. Gillam, L., Ahmad, L., Dalby, D. and Cox, C.: Knowledge Exchange and Terminology Interchange: The role of standards. In Proceedings of Translating and the Computer 24. ISBN 0 85142 476 7 (2002).
23. Noy, N. F., Klein, M.: Ontology evolution: Not the Same as Schema Evolution. Knowledge and Information Systems, 5 (2003).
24. Ahmad, K., Tariq, M., Vrusias, B. and Handy, C.:. Corpus-Based Thesaurus Construction for Image Retrieval in Specialist Domains. ECIR 2003, LNCS 2633. Springer Verlag, Heidelberg (2003), 502-510.
25. Gillam, L. and Ahmad, K.: Sharing the knowledge of experts. Fachsprache 24(1-2). (2002), 2-19.
26. Gillam, L. (Ed): Terminology and Knowledge Engineering: making money in the financial services industry. Proceedings of workshop at 2002 conference on Terminology and Knowledge Engineering. (2002).
27. Hearst, M.: Automatic acquisition of hyponyms from large text corpora. Proceedings of the Fourteenth International Conference on Computational Linguistics. Nantes, France. (1992), 539–545
28. Gillam, L.: Metadata descriptors: ISO standards for terminology and other language resources. Proc. of 1st International e-Social Science Conference. (2005).

Metadata Management in a Multiversion Data Warehouse*

Robert Wrembel and Bartosz Bębel

Institute of Computing Science, Poznań University of Technology, Poznań, Poland
{Robert.Wrembel, Bartosz.Bebel}@cs.put.poznan.pl

Abstract. A data warehouse (DW) is supplied with data that come from external data sources (EDSs) that are production systems. EDSs, which are usually autonomous, often change not only their contents but also their structures. The evolution of external data sources has to be reflected in a DW that uses the sources. Traditional DW systems offer a limited support for handling dynamics in their structures and contents. A promising approach to this problem is based on a multiversion data warehouse (MVDW). In such a DW, every DW version includes a schema version and data consistent with its schema version. A DW version may represent a real state at certain period of time, after the evolution of EDSs or changed user requirements or the evolution of the real world. A DW version may also represent a given business scenario that is created for simulation purposes. In order to appropriately synchronize a MVDW content and structure with EDSs as well as to analyze multiversion data, a MVDW has to manage metadata. Metadata describing a MVDW are much more complex than in traditional DWs. In our approach and prototype MVDW system, a metaschema provides data structures that support: (1) monitoring EDSs with respect to content and structural changes, (2) automatic generation of processes monitoring EDSs, (3) applying the discovered EDS changes to a selected, DW version, (4) describing the structure of every DW version, (5) querying multiple DW versions of interest at the same time, (6) presenting and comparing multiversion query results.

1 Introduction

A data warehouse (DW) is a large database (often of terabytes size) that integrates data from various external data sources (EDSs). A DW content includes historical, summarized, and current data. Data warehouses are important components of decision support systems. Data integrated in a DW are analyzed by, so called, On-Line Analytical Processing (OLAP) applications for the purpose of: discovering trends (e.g. sale of products) patterns of behavior and anomalies (e.g. credit card usage) as well as finding hidden dependencies between data (e.g. market basket analysis, suggested buying).

* This work is partially supported by the grant no. 4 T11C 019 23 from the Polish State Committee for Scientific Research (KBN), Poland.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1347 – 1364, 2005.

The process of good decision making often requires forecasting future business behavior, based on present and historical data as well as on assumptions made by decision makers. This kind of data processing is called a **what-if analysis**. In this analysis, a decision maker simulates in a DW changes in the real world, creates virtual possible business scenarios, and explores them with OLAP queries. To this end, a DW must provide means for creating and managing various DW alternatives, that often requires changes to the DW structure.

An inherent feature of external data sources is their autonomy, i.e. they may evolve in time independently of each other and independently of a DW that integrates them [40, 41]. The changes have an impact on the structure and content of a DW. The evolution of EDSs can be characterized by: **content changes**, i.e. insert/update/delete data, and **schema changes**, i.e. add/modify/drop a data structure or its property. Content changes result from user activities that perform their day-to-day work with the support of information systems. On the contrary, schema changes in EDSs are caused by: changes of the real world being represented in EDSs (e.g. changing borders of countries, changing administrative structure of organizations, changing legislations), new user requirements (e.g. storing new kinds of data), new versions of software being installed, and system tuning activities.

The consequence of content and schema changes at EDSs is that a DW built on the EDSs becomes obsolete and needs to be synchronized. Content changes are monitored and propagated to a DW often by means of materialized views [20]. and the history of data changes is supported by applying temporal extensions e.g. [12]. Whereas EDSs schema changes are often handled by applying schema evolution, e.g. [10, 26] and temporal versioning techniques [17, 18, 33]. In schema evolution approaches historical DW states are lost as there is only one DW schema that is being modified. In temporal versioning approaches only historical versions of data are maintained whereas schema modifications are difficult to handle.

In our approach, we propose a multiversion data warehouse (MVDW) as a framework for: (1) handling content and schema changes in EDSs, (2) simulating and managing alternative business scenarios, and predicting future business trends (a what-if analysis). A MVDW is composed of persistent versions, each of which describes a DW schema and content in a given time period.

In order to support the lifecycle of a DW, from its initial loading by ETL processes, periodical refreshing, to OLAP processing and query optimization, a DW has to provide metadata. Metadata are defined as data about a DW. They are used for improving a DW management and exploitation. There are two basic types of metadata, namely business and technical ones. **Business metadata** include among others: dictionaries, thesauri, business concepts and terminology, predefined queries, and report definitions. They are mainly used by end-users. **Technical metadata** include among others: a DW schema description and the definitions of its elements, physical storage information, access rights, statistics for a query optimizer, ETL process descriptions, and data transformation rules [45].

In the case of a multiversion data warehouse, metadata are much more complex than in traditional DWs and have to provide additional information. Industry standard metamodels, i.e. Open Information Model [36] and Common Warehouse Metamodel [15] as well as research contributions, e.g. [24, 37] have not yet considered the incorporation of metadata supporting either schema and data evolution or schema and data versioning.

This paper's focus and **contribution** includes the development of metaschemas for the support of: (1) detecting structural and content changes in EDSs and propagating them into a MVDW, (2) automatic generation of software monitoring EDSs, based on metadata, (3) managing versions of schemas and data in a MVDW, (4) executing queries that address several DW versions, (5) presenting and comparing multiversion query results. Based on the developed metamodel, a prototype MVDW system was implemented in Java and Oracle PL/SQL language, whereas data and metadata are stored in an Oracle10g database. To the best of our knowledge, it is the first approach and implemented system managing multiple, persistent, and separate DW versions as well as supporting the analysis of multiversion data.

The rest of this paper is organized as follows. Section 2 presents basic definitions in the field of DW technology. Section 3 discusses existing approaches to handling changes in the structure and content of a DW as well as approaches to metadata management. Section 4 overviews our concept of a multiversion DW and presents its metaschema. Section 5 presents the mechanism of detecting changes in EDSs and its underlying metamodel. Finally, Section 6 summarizes the paper.

2 Basic Definitions

A DW takes advantage of a multidimensional data model [21, 24, 29] with **facts** representing elementary information being the subject of analysis. A fact contains numerical features, called **measures**, that quantify the fact and allow to compare different facts. Values of measures depend on a context set up by **dimensions**. Examples of measures include: quantity, income, turnover, etc., whereas typical examples of dimensions include *Time*, *Location*, *Product*, etc. (cf. Fig. 1). In a relational implementation, a fact is implemented as a table, called a **fact table**, e.g. *Sales* in Fig. 1.

Dimensions usually form **hierarchies**. Examples of hierarchical dimensions are: (1) *Location*, with *Cities* at the top and *Shops* at the bottom, (2) *Product*, with *Categories* and *Items* (cf. Fig. 1). A schema object in a dimension hierarchy is called a **level**, e.g. *Shops*, *Cities*, *Categories*, *Items*, and *Time*. In a relational implementation, a level is implemented as a table, called a **dimension level table**.

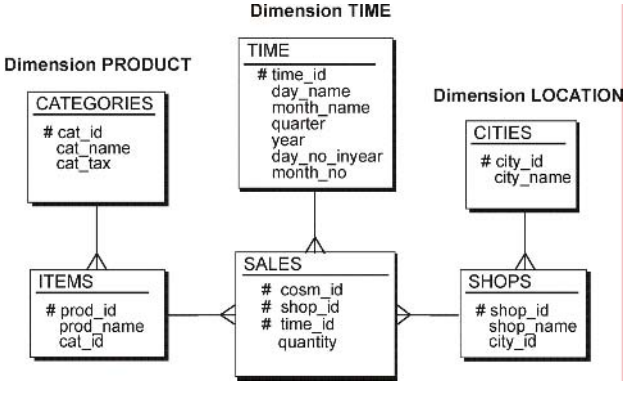


Fig. 1. An example DW schema on sale of products

A dimension hierarchy specifies the way measures are aggregated. A lower level of a dimension rolls-up to an upper level, yielding more aggregated data. Values in every level are called **level instances**. Example instances of level *Items* may include: *'t-shirt'* and *'shampoo'*, whereas instances of level *Categories* may include: *'clothes'* and *'cosmetics'*. The **dimension instance** of dimension D_i is composed of hierarchically assigned instances of levels in D_i, where the hierarchy of level instances is set up by the hierarchy of levels. Example instances of dimension *Product* include: {*'t-shirt'* → *'clothes'*, *'shampoo'* → *'cosmetics'*}, where → is the hierarchical assignment of a lower level instance to an upper level instance.

3 Related Work

The problem of schema changes appeared in mediated and federated database systems that were used for interconnecting heterogeneous data sources, e.g. [8, 9, 13, 30, 43]. A lot of research have been carried out in order to handle schema changes and propagate them into a global schema, e.g. [1, 6, 7, 31, 32]. Handling schema changes in EDSs and propagating them into a DW is partially based on that solutions. However, DW systems have different characteristics requiring new approaches to this problem. First of all, a final DW schema is usually totally different form EDSs schemas. Second of all, a DW stores persistent elementary data as well as data aggregated at many levels that have to be transformed after a DW schema updates.

The approaches to the management of changes in a DW can be classified as: (1) schema and data evolution: [10, 22, 23, 26, 27, 44], (2) temporal and versioning extensions [3, 11, 12, 14, 17, 18, 25, 28, 29, 33, 41].

The approaches in the first category support only one DW schema and its instance. In a consequence, many structural modifications require data conversions that results in the loss of historical DW states, e.g. dropping an attribute, changing an attribute data type, length or domain.

In the approaches from the second category, in [12, 17, 18, 33] changes to a DW schema are time-stamped in order to create temporal versions. The approaches are suitable for representing historical versions of data, but not schemas.

In [14, 41] data versions are used to avoid duplication anomaly during DW refreshing process. The work also sketches the concept of handling changes in an EDS structure. However, a clear solution was not presented on how to apply the changes to DW fact and dimension tables. Moreover, changes to the structure of dimensions as well as dimension instances were not taken into consideration.

In [25, 28, 38] implicit system created versions of data are used for avoiding conflicts and mutual locking between OLAP queries and transactions refreshing a DW.

On the contrary, [11] supports explicit, time-stamped versions of data. The proposed mechanism, however, uses one central fact table for storing all versions of data. In a consequence, only changes to dimension and dimension instance structures are supported. In [19] a DW schema versioning mechanism is presented. A new persistent schema version is created for handling schema changes. The approach supports only four basic schema modification operators, namely adding/deleting an attribute as well as adding/deleting a functional dependency. A persistent schema version requires a population with data. However, this issue is only mentioned in the paper. [42] addresses the problem of handling changes only in the structure of a

dimension instances. To this end, a time-stamped history of changes to dimension instances is stored in an additional data structure. The paper by [29] addresses the same problem and proposes consistency criteria that every dimension has to fulfill. It gives an overview how the criteria can be applied to a temporal DW only.

In [3] a virtual versioning mechanism was presented. A virtual DW structure is constructed for hypothetical queries simulating business scenarios. As this technique computes new values of data for every hypothetical query based on virtual structures, performance problems will appear for large DWs.

In order to handle schema and data evolution as well as versioning and in order to allow querying such evolving DW systems, the set of well defined metadata is required. From the approaches discussed above, only [18] presents a metamodel for a temporal DW. Additionally, [37] discusses and presents high level metamodel for handling and assuring data quality in a DW. The author only mentions the need for a metamodel supporting DW evolution, without giving any solutions.

The need for metadata describing multiple areas of a DW system design, development, deployment, and usage as well as the need for data exchange between different heterogeneous systems resulted in two industrial metadata standards, namely the *Open Information Model* (OIM) [24, 36, 45] and the *Common Warehouse Metadata* (CWM) [15, 24, 45], developed by multiple industry vendors and software providers. **OIM** was developed by the **Meta Data Coalition** (MDC) for the support of all phases of an information system development. OIM is based on UML, XML, and SQL92. It includes the following models: (1) object-oriented analysis and design, (2) object and component development life-cycles, (3) business engineering, (4) knowledge management tool, and (5) database and data warehousing model, including: database and multidimensional schema elements, data transformations, non-relational source elements, report definitions. OIM is supported among others by Microsoft, Brio Technologies, Informatica, and SAS Institute.

On the contrary, **CWM** was developed by the **Object Management Group** (OMG) for the support of integrating DW systems and business intelligence tools. The standard is based on XML, CORBA IDL, MOF, and SQL99. It includes the following models: (1) foundation of concepts and structures, (2) warehouse deployment, (3) relational interface to data, (4) record-oriented structures, (5) multidimensional database representation, (6) XML types and associations, (7) type transformations, (8) OLAP constructs, (9) warehouse process flows, (10) warehouse day-to-day operations. CWM is supported among others by IBM, Oracle, and Hyperion.

In 2000, the standard developed by MDC was integrated into the standard developed by OMG. Since then, the integrated standard is developed under OMG [46, 47].

None of the discussed standards, however, includes models supporting detection and propagation of changes from an EDS to a DW, or models supporting schema and data evolution in a DW. Consequently, they do not provide support for temporal or cross-version queries. Whereas our approach and implemented prototype system supports handling the evolution of DW schema and data by applying versioning mechanism. Moreover, a user can query multiple DW version, analyze, and compare the query results. In order to support these functionalities the system has to manage various metadata that are described by the metaschema that we have developed.

4 Multiversion Data Warehouse

This section informally overviews our concept of a multiversion DW, presents its metaschema, and outlines the approach to querying multiple DW versions. Formal description of a MVDW was presented in [34].

4.1 Basic Concepts

A **multiversion data warehouse** (MVDW) is composed of the set of its versions. A DW version is in turn composed of a schema version and an instance version. A **schema version** describes the structure of a DW within a given time period, whereas an **instance version** represents the set of data described by its schema version.

We distinguish two types of DW versions, namely real and alternative ones. **Real versions** are created in order to keep up with changes in a real business environment, like for example: changing organizational structure of a company, changing geographical borders of regions, changing prices/taxes of products. Real versions are linearly ordered by the time they are valid within. **Alternative versions** are created for simulation purposes, as part of the what-if analysis. Such versions represent virtual business scenarios. All DW versions are connected by **version derivation relationships**, forming a **version derivation graph**. The root of this tree is the first real version. Fig. 2 schematically shows real and alternative versions. *R1* is an initial real version of a DW. Based on *R1*, new real version *R2* was created. Similarly, *R3* was derived from *R2*, etc. *A2.1* and *A2.2* are alternative versions derived from *R2*, and *A4.1* is an alternative version derived from *R4*.

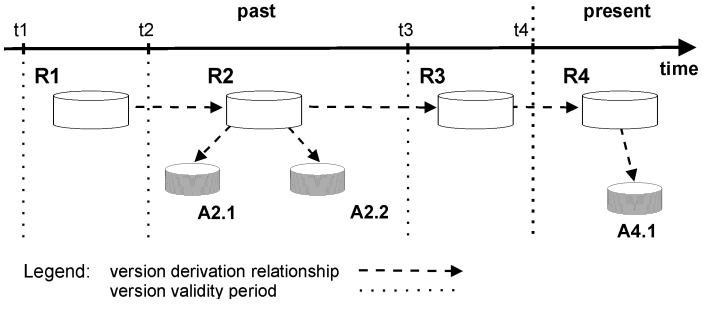


Fig. 2. An example real and alternative versions derivation graph and version validity times

Every DW version is valid within certain period of time represented by two timestamps, i.e. **begin validity time** (BVT) and **end validity time** (EVT) [5]. For example, real version *R1* (from Fig. 2) is valid within time *t1* (BVT) and *t2* (EVT), *R2* is valid within *t2* and *t3*, whereas *R4* is valid from *t4* until present. Alternative versions *A2.1*, *A2.2*, and *A4.1* are valid within the same time period as the real versions they were derived from.

A schema version, after being derived, is modified by means of operations that have an impact on a DW schema - further called schema change operations, as well as by operations that have an impact on the structure of a dimension instance - further

called dimension instance structure change operations. **Schema change operations** include among others: adding a new attribute to a level, dropping an attribute from a level, creating a new fact table, associating a given fact table with a given dimension, renaming a table, creating a new level table with a given structure, including a super-level table into its sub-level table, and creating a super-level table based on its sub-level table. The last three operations are applicable to snowflake schemas.

Dimension instance structure change operations include among others: inserting a new level instance into a given level, deleting an instance of a level, changing the association of a sub-level instance to another super-level instance, merging several instances of a given level into one instance of the same level, and splitting a given level instance into multiple instances of the same level. The full list of schema and dimension instance structure change operations with their formal semantics, their application to a MVDW, and their outcomes can be found in [4]. Their presentation is out of scope of this paper.

4.2 MVDW Metaschema

The model of a MVDW is composed of multiversion dimensions and multiversion facts. A multiversion dimension is composed of dimension versions. A dimension version is in turn composed of level versions that form hierarchies. A multiversion fact is composed of fact versions. A fact version is associated with several dimension versions. This association represents a cube version. A fact version and a dimension version can be shared by several DW versions.

The overall metaschema of our prototype MVDW is shown in Fig. 3. It is designed in the Oracle notation [2] where: a dashed line means a not mandatory foreign key, a solid line means a mandatory foreign key, a line end split into three means a relationship of cardinality many, whereas a simple line end means a relationship of cardinality one.

The *Versions* dictionary table stores the information about all existing DW versions, i.e. version identifier, name, begin and end validity times, status (whether a version is committed or under development), type (a real or an alternative one), parent-child (derivation) dependencies between versions. The meta information about fact versions is stored in the *Fact_Versions* dictionary table, i.e. fact identifier, name, an identifier of a multiversion fact a given fact belongs to, fact implementation name, DW version identifier a given fact belongs to, the identifier of a transaction that created a given fact. The meta information about dimension versions is stored in *Dim_Versions*, i.e. dimension version identifier, name, an identifier of a multiversion dimension a given dimension belongs to, DW version identifier a given dimension belongs to, the identifier of a transaction that created a given dimension.

The description of versions of hierarchies and their assignments to a given dimension version are stored in *Hier_Versions* and *Dim_Hier_Versions*, respectively. Versions of hierarchies are composed of level versions, whose descriptions are stored in *Lev_Versions*, i.e. level identifier, name, an identifier of a multiversion level a given level belongs to, implementation name, DW version identifier a given level belongs to, the identifier of a transaction that created a given level. Level versions are components of versions of level hierarchies. These associations are stored in *Hier_Elements*.

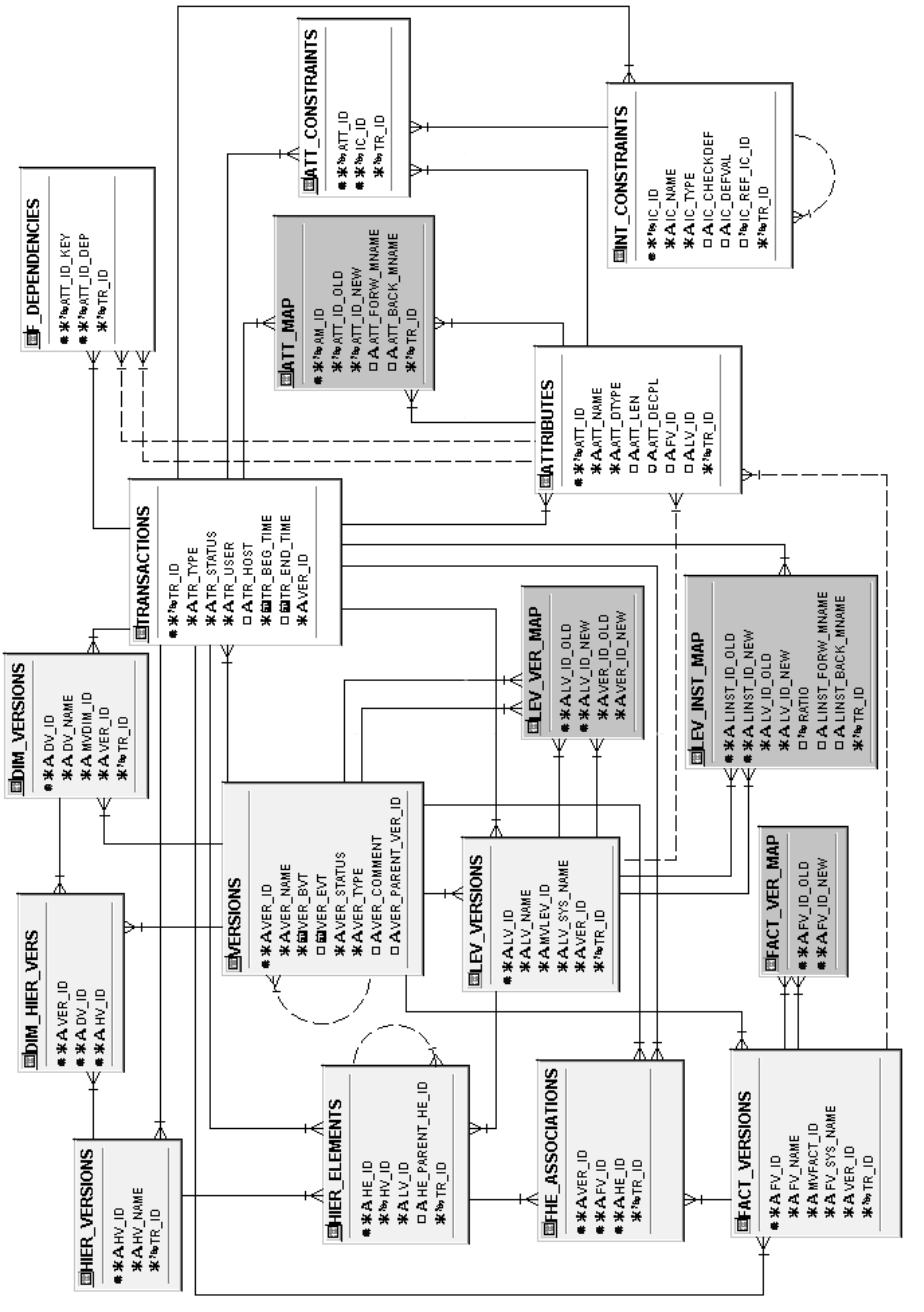


Fig. 3. The metaschema of our prototype MVDW

Fact versions are associated with dimension versions via level versions. The associations are stored in *FHE_Associations*. Every record in this metatable contains an identifier of a fact version, and identifier of the version of a hierarchy element (an association with the lowest level in a level hierarchy), an identifier of a DW version this association is valid in, and an identifier of a transaction that created this association.

Every fact version and level version includes the set of its attributes, that are stored in the *Attributes* dictionary table. Notice that attributes are not versioned in order to simplify the model. In a consequence, a single attribute can't be shared by multiple DW versions. Integrity constraints defined for attributes as well as for fact and level tables are stored in *Att_Constraints* and *Int_Constraints*. Functional dependencies between attributes in level versions are stored in *F_Dependencies*.

Table *Att_Map* is used for storing mappings between an attribute existing in DW version V_o and a corresponding attribute in a child version V_p. This kind of mappings are necessary in order to track attribute definition changes between versions, i.e. changing attribute name, data type, length, and integrity constraints. Some changes in attribute domain between two consecutive DW versions, say V_o and V_p (e.g. changing university grading scale from the Austrian one to the Polish one) will require data transformations, if the data stored in V_o and V_p are to be comparable. To this end, forward and backward conversion methods have to be provided. Their names are registered in *Att_Map* as the values of *Att_Forw_Mname* and *Att_Back_Mname*, respectively. In our prototype system, conversion methods are implemented as Oracle PL/SQL functions. The input argument of such a function is the name of an attribute whose value is being converted and the output is the converted value. Conversion methods are implemented by a DW administrator and they are registered in the metaschema by a dedicated application.

The *Fact_Ver_Map* dictionary table is used for storing mappings between a given fact table in DW version V_o and a corresponding fact table in version V_p, directly derived from V_o. This kind of mappings are necessary in order to track fact table definition changes between versions, i.e. changing table name or splitting a table.

The purpose of *Lev_Ver_Map* is to track changes of level tables between versions, i.e. changing table name, including a super-level table into its sub-level table, creating a super-level table based on its sub-level table, cf. [4].

As outlined in Section 4.1, the instances of level versions can be modified by changing associations to super-level instances as well as by merging and splitting them. Operations of this type result in new dimension instance structures. In order to allow querying multiple DW versions under such modifications, the system has to map level instances in version V_o into their corresponding instances that were modified in version V_p. To this end, the *Lev_Inst_Map* data dictionary table is used.

Example 1. In order to illustrate the idea and usage of mapping tables, let us consider a DW schema from Fig. 1 and let us assume that initially, in a real version from February (R^{FEB}) to March (R^{MAR}) there existed 3 shops, namely *ShopA*, *ShopB*, and *ShopC* that were represented by appropriate instances of the *Location* dimension. In April, a new DW version was created, namely R^{APR} in order to represent a new reality where *ShopA* and *ShopB* were merged into one shop - *ShopAB*. This change was reflected in the *Location* dimension instances. To this end, two following records were inserted to the *Lev_Inst_Map* dictionary table:

```
<id_ShopA,id_ShopAB,id_ShopsR^MAR,id_ShopsR^APR,
                                    100,null,null,id_tr>
<id_ShopB,id_ShopAB,id_ShopsR^MAR,id_ShopsR^APR,
                                    100,null,null,id_tr>
```

The first and the second value in the above records represents an identifier of *ShopA* and *ShopAB*, respectively. The third and fourth value represents the *Shops* level table identifier in version R^{MAR} and R^{APR}, respectively. The fifth value stores the merging/splitting ratio. In our example, the ratio equals to 100, meaning that the whole *ShopA* and *ShopB* constitute *ShopAB*.

For more advanced splitting or merging operations it will be necessary to provide a backward and forward transformation methods for converting facts from an old to a new DW version. If such methods are explicitly implemented and provided by a DW administrator, then their names are registered as the values of the sixth and seventh attribute. The last attribute stores transaction identifier of a transaction that carried out the modifications. □

The prototype MVDW is managed in a transactional manner and the *Transactions* dictionary table stores the information about transactions used for creating DW versions and modifying them.

4.3 Metadata Visualization - MVDW User Interface

A MVDW administrator manages the structure and content of a MVDW via a graphical application, implemented in Java. Its main management window is shown in Fig. 4. It is composed of the *version navigator*, located at the left hand side and the *schema viewer*, located at the right hand side. Both visualize the content of the MVDW metaschema.

The main functionality of the application includes:

- the derivation of a new (real or alternative) version of a data warehouse schema and its instance;
- the modification of a schema version and dimension instance structures, by means of operations outlined in Section 4.1;
- loading data from EDSs into a selected DW version (any ODBC data sources, sources accessible via a gateway, or text files can be used);
- visualizing the schema of a selected DW version;
- visualizing a schema version derivation graph;
- querying multiple DW versions and presenting query results.

4.4 Metadata in Multi-version Queries

The content of a MVDW can be queried either by a query that addresses a single version - further called a **single-version query** (SVQ) or by addressing multiple versions - further called a **multi-version query** (MVQ).

In a MVDW, data of user interest are usually distributed among several versions and a user may not be aware of the location of particular data. Moreover, DW versions being addressed in multi-version queries may differ with respect to their schemas. For these reasons, querying a MVDW is challenging and requires intensive usage of metadata.

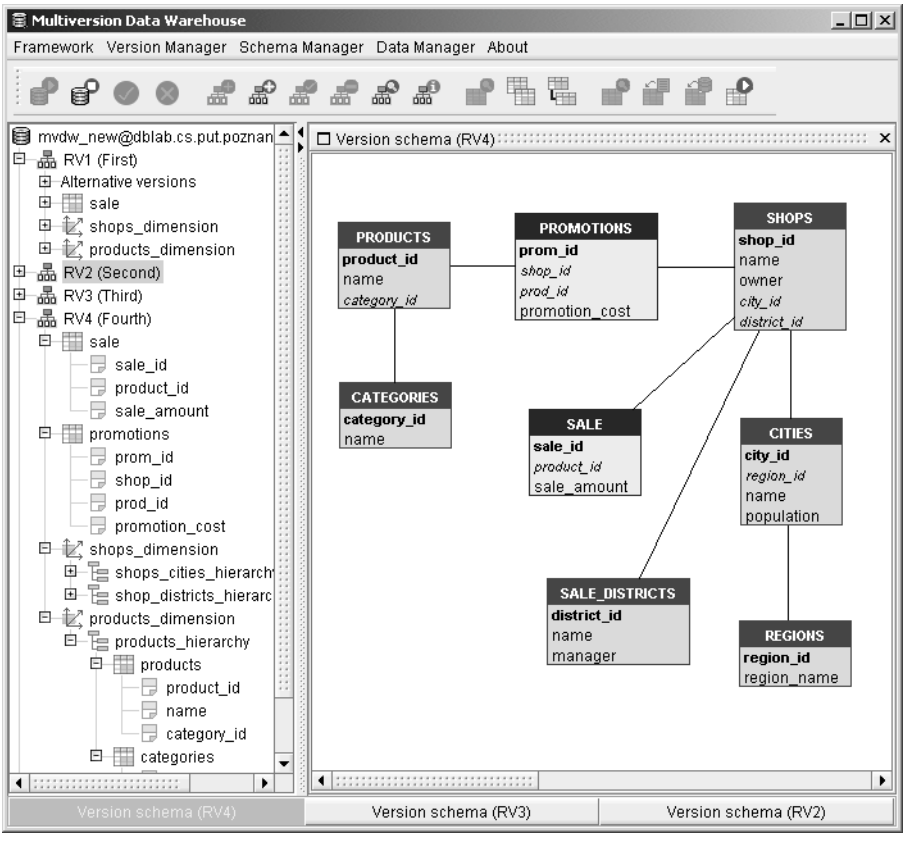


Fig. 4. The application for managing a MVDW

For the purpose of querying a MVDW, a traditional SQL `select` command has to be extended. To this end, we proposed clauses that allow querying: (1) a single DW version, that can be either a real or an alternative one, (2) the set of real DW versions, (3) the set of alternative DW versions. By including clauses (2) and (3) in one query, a user can query real and alternative versions at once. The detail description of the clauses as well as a user interface for specifying multi-version queries and visualizing their results is discussed in [35].

A user's multi-version query is processed by the MVQ parser and executor in the following steps.

1. **Constructing the set of DW versions**
   The set S^V of versions that is to be addressed in a MVQ is constructed by the parser by using version *begin validity time* and *end validity time* (cf. Section 4.1), which are stored in the *Versions* dictionary table.
2. **Decomposing MVQ**
   Next, for every DW version in S^V, the parser constructs an appropriate single-version query. In this process, the differences in version schemas are taken into consideration. If some tables and attributes changed names between versions, then

appropriate names are found in data dictionary tables and are used in SVQs. If an attribute is missing in DW versions V_i, V_j, V_k, then the attribute is excluded from single-version queries addressing V_i, V_j, V_k. The data dictionary tables used in this step include among others: *Fact_Versions*, *Dim_Versions*, *Hier_Verisions*, *Dim_Hier_Versions*, *Hier_Elements*, *FHE_Associations*, *Lev_Versions*, *Fact_Ver_Map*, *Lev_Ver_Map*, *Attributes*, *Att_Map*.

3. **Executing SVQs**
   Every single version query constructed in step 2) is next executed in its own DW version. Then, the result set of every SVQs is returned to a user and presented separately. Additionally, every result set is annotated with:
   - an information about a DW version the result was obtained from,
   - a meta information about schema and dimension instance changes between adjacent DW versions being addressed by a MVQ. The metadata information attached to SVQ result allows to analyze and interpret the obtained data appropriately.
4. **Integrating SVQ results**
   Result sets of single-version queries, obtained in step 3), may be in some cases integrated into one common data set. This set is represented in a DW version specified by a user (the current real version by default). This integration will be possible if a MVQ addresses attributes that are present in all versions of interest and if there exist transformation methods between adjacent DW versions (if needed). For example, it will not be possible to integrate the results of a MVQ addressing DW version V_o and V_p, computing the sum of products sold (`select sum(amount) ...`), if in version V_o attribute *amount* exists and in version V_p the attribute was dropped.
   While integrating result sets the following dictionary tables are used among others: *Fact_Versions*, *Fact_Ver_Map*, *Lev_Versions*, *Lev_Ver_Map*, *Attributes*, *Att_Map*, *Lev_Inst_Map*.

Example 2. In order to illustrate annotating result sets of SVQs with meta information let us consider a DW schema from Fig. 1 and let us assume that initially, in a real version from February R^{FEB}, there existed 3 shops, namely *ShopA*, *ShopB*, and *ShopC*. These shops were selling *Ytong bricks* with *7%* of VAT. Let us assume that in March, *Ytong bricks* were reclassified to *22%* VAT category (which is a real case of Poland after joining the European Union). This reclassification was reflected in a new real DW version R^{MAR}.

Now we may consider a user's MVQ that addresses DW versions from February till March and computes gross total sale of products. The query is decomposed into two partial queries: one for R^{FEB} and one for R^{MAR}. After executing the corresponding SVQs in their proper versions, the result set of SVQ addressing version R^{MAR} is augmented and returned to a user with meta information describing changes in the structure of the *Product* dimension instance between R^{FEB} and R^{MAR}, as follows:

```
Dimension PRODUCT: Level PRODUCTS:
          change association:
          Ytong bricks(vat 7% → vat 22%)
```

This way a sale analyst will know that a gross sale increase form February to March was at least partially caused by VAT increase. □

5 Detecting Changes in EDSs

For each external data source supplying a MVDW we define the set of events being monitored and the set of actions associated with every event.

5.1 Events and Actions

We distinguish two types of events, namely: structure events and data events. A **structure event** signalizes changes in an EDS's structure, that include: adding an attribute, modifying the name or domain of an attribute, dropping an attribute, adding a new data structure (table, class), dropping a data structure, changing the name of a data structure. A **data event** signalizes changes in an EDS's content, that include: adding, deleting, or modifying a data item. The set of events being monitored at EDSs is explicitly defined by a DW administrator and stored in so called **mapping metaschema**, cf. Section 5.2.

For every event in the set, a DW administrator explicitly defines one or more ordered **actions** to be performed in a particular DW version. We distinguish two kinds of actions, namely messages and operations. **Messages** represent actions that can not be automatically applied to a DW version, e.g. adding an attribute to an existing data structure at an EDS, creating a new data structure. These events may not necessarily require DW version modification if a new object is not going to store any information of user's interest. Messages are used for notifying a DW administrator about certain source events. Being notified by a message, an administrator can manually define and apply appropriate actions into a selected DW version. **Operations** are generated for events whose outcomes can be automatically applied to a DW version, e.g. the insertion, update, and deletion of a record, the modification of an attribute domain or name, the change of a data structure name. The associations between events and actions is stored in the mapping metaschema.

From the implementation point of view, operations are represented by SQL DML and DDL statements or stored procedures addressing an indicated DW version. The executable codes of operations and bodies of messages are automatically generated by monitors, cf. Section 5.3.

5.2 Mapping Metaschema

The structure of the mapping metaschema is shown in Fig. 5 (represented in the Oracle notation). The *SRC_SOURCES* dictionary table stores descriptions of external data sources. Information about EDSs data structures whose changes are to be monitored, are registered in two dictionary tables: *SRC_OBJECTS* and *SRC_ ATTRIBUTES*. All monitored events at EDSs are stored in *SRC_EVENTS*.

DW_AC_SRC_EV_MAPPINGS stores mappings between events detected at EDSs and their associated actions that are to be executed in a given DW version. Action definitions, i.e. an action type and a data warehouse object an action is to be performed on, are stored in *DW_ACTIONS*. Data warehouse object descriptions (i.e. fact and dimension level tables, dimensions and hierarchies) are stored in the *DW_OBJECTS* and *DW_ATTRIBUTES* dictionary tables. Values taken from EDSs may need transformations before being loaded into a DW version (e.g. conversion of

GBP into Euro). Expressions that transform/compute values of attributes are stored in the *DW_ATTR_EXPRESSIONS*.

A DW administrator defines the content of the mapping metaschema (i.e. mappings between events and actions) by means of a graphical application, called the *metaschema manager* written in Java. The mapping metaschema in stored in an Oracle10g database.

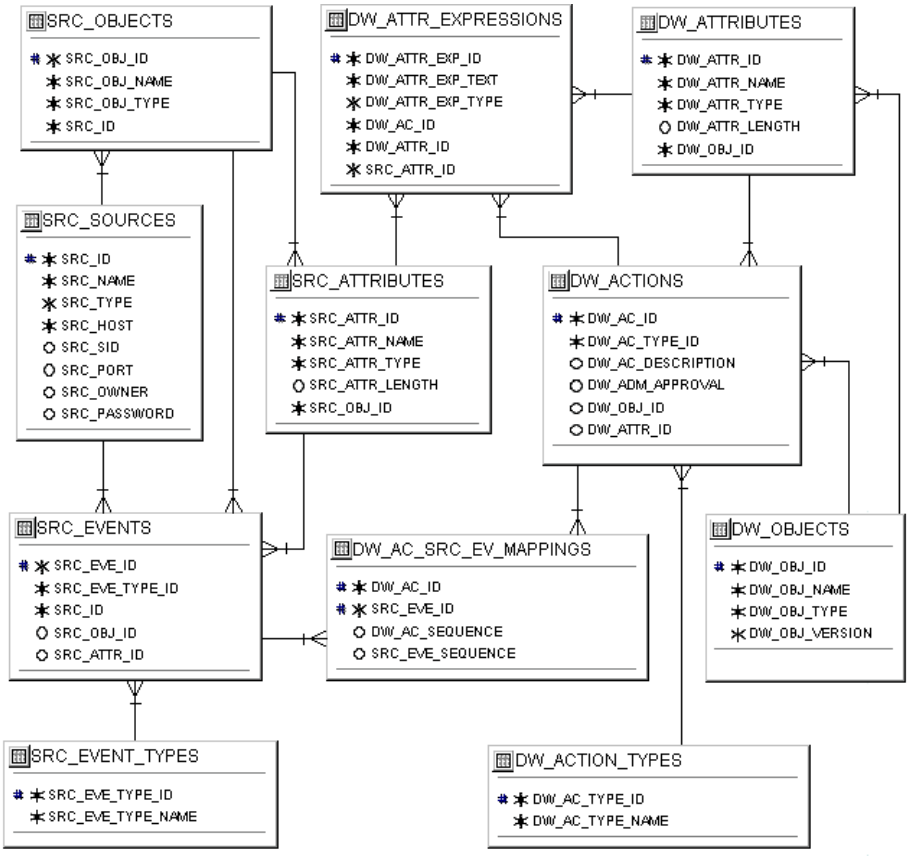


Fig. 5. The structure of the mapping metaschema

5.3 Automatic Generation of Monitors

External data sources are connected to a MVDW in a standard way via software called monitors. Each EDS has its own dedicated **monitor** that is responsible for detecting the occurrences of predefined events at that source. For every EDS, the code of its monitor is automatically generated by a software module called the **monitor generator**, based on the content of the mapping metaschema. In the current prototype system, monitors are implemented in the Oracle PL/SQL language as stored packages and triggers detecting defined events.

After installing monitors at EDSs, they generate executable codes of operations and bodies of messages in response to EDS events. Generated actions are stored in a special data structure called the **DW update register**. Every action is described by its type (message or DML statement), its content (e.g. SQL statement or stored procedure) addressing particular objects in a particular DW version, and its sequence that represents the order of action executions. When an administrator decides to refresh a DW version, he/she selects actions for execution and runs a dedicated process, called the **warehouse refresher**, that reads actions stored in the DW update register and applies them to a specified DW version.

6 Summary and Conclusions

Handling changes in external data sources and applying them appropriately into a DW became one of the important research and technological issues [16, 39]. Structural changes applied inappropriately to a DW schema may result in wrong analytical results. Research prototypes and solutions to this problem are mainly based on temporal extensions that limit their use. Commercial DW systems existing on the market (e.g. Oracle10g, Oracle Express Server, IBM DB2, SybaseIQ, Ingres DecisionBase OLAP Server, NCR Teradata, Hyperion Essbase OLAP Server, SAP Business Warehouse, MS SQL Server2000) do not offer mechanisms for managing multiple DW states.

Our approach to this problem is based on a multiversion data warehouse, where a DW version represents the structure and content of a DW within a certain time period. Managing multiple persistent versions of a DW allows to:

- store history of real world changes without loss of any information,
- create alternative business scenarios for simulation purposes,
- query multiple DW states and compare query results.

A fully functional DW system needs managing metadata in order to support the full lifecycle of a system. In the case of a multiversion data warehouse, metadata are much more complex than in traditional DWs and have to provide additional information, among others on: the structure and content of every DW version, a trace of schema and dimension instance changes applied to every DW version. Industry standard metamodels, i.e. Open Information Model and Common Warehouse Metamodel as well as research contributions have not yet considered the incorporation of metadata supporting either schema and data evolution or schema and data versioning.

In this paper we contributed by:

1. The development of a MVDW metamodel that is capable of: (1) managing versions of schemas and data in a MVDW, (2) executing queries that address several DW versions, and (3) presenting, comparing, and interpreting multiversion query results.
2. The framework for detecting changes in external data sources and propagating them into a MVDW, with the functionality of: (1) automatic generation of software monitoring EDSs, based on metadata, (2) automatic generation of actions in response to EDSs events.

Based on the developed metamodels, a prototype MVDW system was implemented in Java and Oracle PL/SQL language, whereas data and metadata are stored in an Oracle10g database.

In the current implementation, monitors are automatically generated for data sources implemented on Oracle databases. In future we plan to extend automatic generation of monitors for other database systems including: IBM DB2, Sybase Adaptive Server Enterprise, and MS SQL Server as well as for non-database sources including: text and XML files. Future work will also focus on extending our metamodels in order to handle data quality issues in a MVDW. Another interesting task is to extend the accepted industry standard CWM so that it is suitable for describing a multiversion DW.

References

1. Andany J., Leonard M., Palisser C.: Management of schema evolution in databases. Prof. of VLDB, 1991
2. Barker R.: Case*Method: Entity Relationship Modelling Addison-Wesley, 1990, ISBN 0201416964
3. Balmin, A., Papadimitriou, T., Papakonstanitnou, Y.: Hypothetical Queries in an OLAP Environment. Proc. of VLDB, Egypt, 2000
4. Bębel B.: Transactional Refreshing of Data Warehouses. PhD thesis, Poznań University of Technology, Institute of Computing Science, 2005
5. Bębel B., Eder J., Konicilia C., Morzy T., Wrembel R.: Creation and Management of Versions in Multiversion Data Warehouse. Proc. of ACM SAC, Cyprus, 2004
6. Bellahsene Z.: View Mechanism for Schema Evolution in Object-Oriented DBMS. Proc. of BNCOD, 1996
7. Benatallah B.: A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases. Proc. of ER, 1999
8. Bouguettaya A., Benatallah B., Elmargamid A.: Interconnecting Heterogeneous Information Systems, Kluwer Academic Publishers, 1998
9. Elmagarmid A., Rusinkiewicz M., Sheth A.: Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann Publishers, 1999
10. Blaschka, M. Sapia, C., Hofling, G.: On Schema Evolution in Multidimensional Databases. Proc. of DaWak99, Italy, 1999
11. Body, M., Miquel, M., Bédard, Y., Tchounikine A.: A Multidimensional and Multiversion Structure for OLAP Applications. Proc. of DOLAP, USA, 2002
12. Chamoni, P., Stock, S.: Temporal Structures in Data Warehousing. Proc. of DaWaK99, Italy, 1999
13. Chawathe S.S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J.D., Widom J.: The TSIMMIS project: Integration of heterogeneous information sources. Proc. of IPS, Japan, 1994
14. Chen J., Chen S., Rundensteiner E.: A Transactional Model for Data Warehouse Maintenance, Proc of ER, Finland, 2002
15. Object Management Group. Common Warehouse Metamodel Specification, v1.1. http://www.omg.org/cgi-bin/doc?formal/03-03-02
16. Panel discussion on "Future trends in Data Warehousing and OLAP" at DOLAP2004, ACM DOLAP, USA, 2004

17. Eder, J., Koncilia, C.: Changes of Dimension Data in Temporal Data Warehouses. Proc. of DaWaK, Germany, 2001
18. Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for Temporal Data Warehouses. Proc. of CAISE, Canada, 2002
19. Golfarelli M., Lechtenbörger J., Rizzi S., Vossen G.: Schema Versioning in Data Warehouses. ER Workshops 2004, LNCS 3289
20. Gupta A., Mumick I.S. (eds.): Materialized Views: Techniques, Implementations, and Applications. The MIT Press, 1999, ISBN 0-262-57122-6
21. Gyssens M., Lakshmanan L.V.S.: A Foundation for Multi-Dimensional Databases. Proc. of the 23rd VLDB Conference, Grece, 1997
22. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. Proc. of ICDE, Australia, 1999
23. Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Updating OLAP Dimensions. Proc. of DOLAP, 1999
24. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer-Verlag, 2003, ISBN 3-540-42089-4
25. Kang, H.G., Chung, C.W.: Exploiting Versions for On–line Data Warehouse Maintenance in MOLAP Servers. Proc. of VLDB, China, 2002
26. Kaas Ch.K., Pedersen T.B., Rasmussen B.D.: Schema Evolution for Stars and Snowflakes. Proc. of ICEIS, Portugal, 2004
27. Koeller, A., Rundensteiner, E.A., Hachem, N.: Integrating the Rewriting and Ranking Phases of View Synchronization. Proc. of DOLAP, USA, 1998
28. Kulkarni, S., Mohania, M.: Concurrent Maintenance of Views Using Multiple Versions. Proc. of IDEAS, 1999
29. Letz C., Henn E.T., Vossen G.: Consistency in Data Warehouse Dimensions. Proc. of IDEAS, 2002
30. Levy A., Rajaraman A., Ordille J.: Querying heterogeneous information sources using source descriptions. Proc. of VLDB, 1996
31. McBrien P., Poulovassilis A.: Automatic Migration and Wrapping of Database Applications - a Schema Transformation Approach. Proc. of ER, 1999
32. McBrien P., Poulovassilis A.: Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. Proc. of CAiSE, 2002
33. Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. Proc. of VLDB, Egypt, 2000
34. Morzy, T., Wrembel, R.: Modeling a Multiversion Data Warehouse: A Formal Approach. Proc. of ICEIS, France, 2003
35. Morzy T., Wrembel R.: On Querying Versions of Multiversion Data Warehouse. Proc. ACM DOLAP, USA, 2004
36. Meta Data Coalition. Open Information Model. http://www.MDCinfo.com
37. Quix C.: Repository Support for Data Warehouse Evolution. Proc. of DMDW'99
38. Quass, D., Widom, J.: On–Line Warehouse View Maintenance. Proc. of SIGMOD, 1997
39. Rizzi S.: Open Problems in Data Warehousing: 8 Years Later. Keynote speech at DOLAP2003, ACM DOLAP, USA, 2003
40. Roddick J.: A Survey of Schema Versioning Issues for Database Systems. In Information and Software Technology, volume 37(7):383-393, 1996
41. Rundensteiner E., Koeller A., and Zhang X.: Maintaining Data Warehouses over Changing Information Sources. Communications of the ACM, vol. 43, No. 6, 2000
42. Schlesinger L., Bauer A., Lehner W., Ediberidze G., Gutzman M.: Efficienlty Synchronizing Multidimensional Schema Data. Proc. of DOLAP, Atlanta, USA, 2001

43. Templeton M., Henley H., Maros E., van Buer D.J.: InterVisio: Dealing with the complexity of federated database access. The VLDB Journal, 4(2), 1995
44. Vaisman A.A., Mendelzon A.O., Ruaro W., Cymerman S.G.: Supporting Dimension Updates in an OLAP Server. Proc. of CAISE02 Conference, Canada, 2002
45. Vetterli T., Vaduva A., Staudt M.: Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata. SIGMOD Record, vol. 29, No. 3, Sept. 2000
46. http://xml.coverpages.org/OMG-MDC-20000925.html
47. http://www.omg.org/news/releases/pr2000/2000-09-25a.htm

Metadata Management for Ad-Hoc InfoWare - A Rescue and Emergency Use Case for Mobile Ad-Hoc Scenarios

Norun Sanderson, Vera Goebel, and Ellen Munthe-Kaas

Department of Informatics, University of Oslo
{noruns, goebel, ellenmk}@ifi.uio.no

Abstract. In this paper we present a multi-tier data dictionary approach to metadata management in mobile ad-hoc networks (MANETs). Our work is part of the Ad-Hoc InfoWare project, aiming to find middleware solutions to information sharing in MANETs used in rescue and emergency scenarios. Such scenarios are characterised by highly dynamic and time critical issues, and pose new challenges to solutions for information sharing and integration. We present the rescue and emergency scenario, the results of the requirements analysis, and some use cases resulting from this analysis. We describe the design of a knowledge manager component and how it may provide solutions, the main focus being on metadata management and the use of global and local data dictionaries. We present the status of our work, including detailed design and ongoing implementation of a data dictionary manager.

1 Introduction and Motivation

The aim of the Ad-Hoc InfoWare project[1] [1] is to develop middleware services to facilitate information sharing in a mobile ad-hoc network (MANET) for rescue scenarios. MANETs are typically composed of heterogeneous devices of various resources and capabilities. The devices can range from cell phones and PDAs to laptops equipped with wireless network interfaces. Devices can join and leave the network range at all times, which make MANETs very dynamic. Such an infrastructure could be very helpful for information sharing in rescue and emergency operations where the personnel are carrying wireless devices. Sharing information between these wireless devices aids gathering and distribution of information, and may in addition ease coordination and cooperation of rescue personnel, which in rescue and emergency operations typically belong to different organisations. This means that the problems faced by supporting middleware include exchange of information both within and across organisations. To facilitate this, it is necessary that applications running on devices from different organisations are supported in understanding the descriptions of the information available for sharing, i.e., the

[1] This work has been funded by the Norwegian Research Council under the IKT-2010 program, Project no. 152929/431.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1365 – 1380, 2005.

symbolic languages and data models used. Thus, metadata management is essential in enabling effective and efficient sharing of information and resources in a MANET. Our proposed solution for how this may be achieved is the focus of this paper.

Issues of semantic heterogeneity and interoperability relevant for information sharing have been addressed in the distributed database domain, and solutions to some of the challenges may be found here, but our solutions have to take into consideration the added challenges of MANETs, e.g., that information (periodically) may become unavailable, that the devices used are often small and resource-weak, and that the network topology is dynamic.

In this paper we introduce a multi-tier data dictionary approach for metadata management targeted at the challenges to information sharing posed by MANETs and rescue and emergency scenarios. Conceptually, our approach is a three layer approach of increasing levels of knowledge abstraction, realised through Local Data Dictionaries and Semantic Linked Distributed Data Dictionaries. Two-way links connect context and information items, and may also function as a *semantic overlay network* of semantically related nodes. A requirements analysis based on interviews and accident reports of rescue organisations and monitoring of real-life emergency exercises, has been conducted. Results from this analysis include a six phase scenario for MANETs in rescue and emergency operations, as well as issues and requirements that have guided the design of a Knowledge Manager component. Our multi-tier data dictionary solution is realised as a Data Dictionary Manager, which is the main metadata handling component in the Knowledge Manager.

We believe that solutions from the information and knowledge management domain can contribute to solutions for information sharing and integration in MANETs for rescue and emergency scenarios. More specifically, we claim that metadata management is essential in such a solution, and that our multi-tiered data dictionary approach using semantic linked distributed data dictionaries can be a contribution in solving many of the challenges of information sharing and integration in using MANETs as an infrastructure for rescue and emergency use cases.

The rest of the paper is organised as follows. In Section 2 we present the results from our requirements analysis and introduce our six phase scenario for a rescue and emergency operation. Issues and challenges related to knowledge management for MANETs are presented in Section 3, together with the Knowledge Manager design. In Section 4 we describe our multi-tiered data dictionary approach for metadata management, including design of our Data Dictionary Manager, example and use cases. Finally, in Section 5 we give some conclusions and describe future work.

2 Requirements Analysis and Scenario Description

Rescue and emergency scenarios are characterised by involving cooperation of personnel from different organisations; policemen, firemen, physicians, and paramedics. The dimension of heterogeneity imposed on the network by such inter-organisational cooperation makes it essential to find ways to present information so that it can be understood by all participating organisations. This implies supporting functionality akin to high-level distributed database system functionality, keeping track of what information is available in the network, and supporting querying of available information. Ontologies can be used to enable sharing and reuse of knowledge within and across communities. An agreed-upon vocabulary can be shared

and used for querying and assertions, independently of any representations used internally in the organisations.

Each organisation has its own set of rescue operation procedures and guidelines. In addition there are cross-organisational procedures involving governmental and other authorities, and these can be expected to include a command structure. This means that some level of (inter- and intra-) organisational structure should be supported by the middleware, as well as some contextual support for reflecting the type of rescue operation in question. Other valuable uses of context include support for user role and device profiling, personalisation, and providing temporal and spatial information.

In our scenario, we have identified six different phases:

- *Phase 1 – A priori:* This phase is actually before the accident, when the relevant organisations - in cooperation with the authorities - will exchange information on data formats and any shared vocabularies, and make agreements on procedures and working methods.
- *Phase 2 – Briefing:* This phase starts once an accident has been reported. The briefing involves gathering of information about the disaster, e.g., weather, location, number of people involved, and facilities in the area. Some preliminary decisions about rescue procedures and working methods are made, according to the nature of the accident.
- *Phase 3 – Bootstrapping the network:* This phase takes place at the rescue site, and involves devices joining and registering as nodes in the network on arrival. The appointing of rescue leaders also takes place in this phase.
- *Phase 4 – Running of the network:* This is the main phase during the rescue operation. Events that may affect the middleware services, include nodes joining and leaving the network and network partitions and merges. Information is collected, exchanged and distributed. There may be changes in the roles different personnel have in the rescue operation, e.g., change of rescue site leader. New organisations and personnel may arrive and leave the rescue site, and ad-hoc, task-oriented groups involving people from different organisations may form.
- *Phase 5 – Closing of the network:* At the end of the rescue operation all services must be terminated.
- *Phase 6 – Post processing:* After the rescue operation it could be useful to analyze resource use, user movements, and how and what type of information was shared, to gain knowledge for future situations.

Although fixed networks cannot be relied on during the rescue operation itself, the opening phases (phases 1-2) have no such restrictions, and thus give possibilities for preparations that can compensate somewhat for a lack of resources during the rescue operation.

In addition to the possible lack of a fixed network, MANETs pose other challenges that have to be reflected in non-functional requirements, e.g., node movement; different types of devices; scarcity of resources; dynamic network topology etc. It is desirable to keep communication (number of messages) and storage space consumption as low as possible. The dynamic environment and lack of routing table for query routing, implies that query results may depend on the specific location and time of placing the query even though the query itself does not address spatio-temporal data; it is all a question of which nodes are within reach, and which information they contain.

The added challenges of a dynamic and partly unpredictable environment are not met by traditional (distributed) database management systems (DBMS) or systems for information sharing. For instance, DBMSs rely on a stable network topology; a node/server being unavailable is seen as a state of node failure, while in a MANET such a state has to be handled as a normal state, which implies that new solutions for e.g., metadata management, query execution, and transaction management are needed.

The overall requirements resulting from the above analysis, include support for intra- and inter-organisational information flow and knowledge exchange, as well as means to announce and discover information sources. Contextual support enables applications to adapt better to particular scenarios and allow them to fine-tune according to spatial and temporal data. Profiling and personalisation can assist in filtering and presenting information in accordance with the user's and device's needs, as well as display their capabilities.

Other important issues for information sharing in rescue and emergency scenarios include access control and security, management of scarce resources, and asynchronous communication. In the Ad-Hoc InfoWare project, these issues have been addressed by designing an architecture consisting of five components: *distributed event notification system* (DENS), decoupling subscribers and publishers through mediator nodes; *resource management* keeping track of nodes and their resources; *watchdogs* monitoring changes and issuing notifications on change; *security and privacy management*, assuring secure communication and access control, and finally *knowledge management* for handling and integrating ontologies, metadata and information. In this paper, we have limited ourselves to include only issues relevant to knowledge management and metadata management. We refer to [3] for a more general discussion of the scenario and requirements analysis.

Examples of possible applications in our scenario include applications for registering casualties, with identification (if known), photo, degree of urgency (e.g., green, yellow, red), treatment, and data from patient journals if available; and applications for supervising and tracking of personnel. For instance, firemen in a burning building could be given route instructions by their supervisor or team leader, their body temperature measured etc. Another example could be on-site identification of passengers and gathering of information relevant to the investigation of the accident.

3 Knowledge Management Issues and Challenges

From the requirements analysis we have a set of main issues that needs considering in relation to knowledge management. Due to the dynamic environment in both MANETs and in rescue and emergency scenarios, information may become unavailable in an unpredictable fashion. Thus we have to keep track of the **availability** of shared information. Another issue is that the sharing of information is both within and across the participating organisations, so it is essential to aid **understanding** in relation to the data models, standards, languages and vocabularies used in the different organisations. Time and resources are critical factors in rescue operations; this means that issues of **information overload** have to be handled through filtering, personalisation and context awareness. As the purpose is sharing information, the means to query for relevant information, search in available topics, and **retrieval** of relevant information is a major issue. Information **exchange** has to be conducted in agreed-upon standardised formats.

Data management, which is mainly targeted towards the management of storage and distribution of data, does not provide adequate solutions. We have to look towards metadata management and knowledge management, as we here can find solutions for many of the issues regarding sharing knowledge in heterogeneous environments, for instance by the use of ontologies in sharing vocabularies and semantic contexts, and (metadata) models to describe the information items/resources that are to be shared.

3.1 Knowledge Manager Design

The main objective of the Knowledge Manager (KM) is to manage knowledge sharing and integration in the MANET. It adds a layer of knowledge to the information shared in the network. The KM should provide services that allow relating metadata descriptions of information items to a semantic context. The services should be offered both internally to the system as well as to the application level. It should only give the tools, the means for information sharing, not decide usage and content. It is a goal to keep the KM as flexible as possible.

The requirements for the KM include support of the use and sharing of existing domain ontologies (from organisations); enable querying and retrieval of relevant information items, and storage and management of metadata (data structures, content descriptions) and ontologies (vocabularies). The latter includes the classification of, e.g., events, resource types, access levels etc., for use internally in the system, and domain ontologies and different metadata standards for external use. Another highly relevant requirement is to keep track of the availability of information items in the current MANET. Personalisation and filtering of information is also required; by using profiles, e.g., user and device profiles, and contexts, like location time, and situation. As devices of differing size and capability may be used, a scalable configuration is necessary.

We have found that in our scenario there are three kinds of metadata that needs handling, and these metadata categories are reflected in the KM design. The first category is *information structure and content description metadata*; this kind addresses information item contents, structure and localisation. The component focusing on this class of metadata is the Data Dictionary Manager. The second kind, semantic metadata, covers concepts and relations in a domain, providing a semantic context. The Semantic Metadata and Ontology Framework component deals with this kind of metadata. The third class of metadata is profile and context metadata, handled by the Profile and Context Management component. This class of metadata includes profiles for users and devices, and context-related information, typically including location, time, and situation (e.g., the current rescue operation). A user can have different roles in different contexts, and use different devices. A device is of a particular type, and has a set of resources, capabilities, an owner (e.g., organisation) etc. The use for this kind of metadata is filtering and ranking to avoid information overload.

The KM is composed of five components working together to fulfil the KM requirements. Each component is addressing one of the main issues, as illustrated in Figure 1. In the following, each of the KM components will be briefly described.

The *Data Dictionary Manager* (DDM) is the main component for metadata management, designed to manage metadata storage through multi-tiered data dictionaries, and to support tracking of availability. The design of the DDM is more

thoroughly presented in Section 4. Each DDM has a *Local Data Dictionary* (LDD), containing metadata descriptions of information items to be shared in the MANET, and a *Semantic Linked Distributed Data Dictionary* (SDDD), keeping a global view of sharable information.

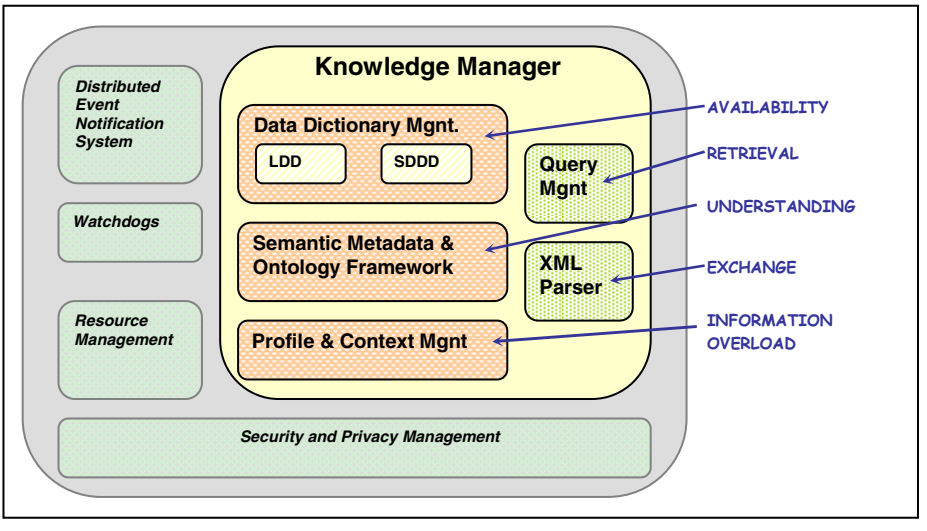


Fig. 1. The Knowledge Manager

The purpose of the *Semantic Metadata and Ontology Framework* is to aid use and sharing of ontologies and semantic metadata. The requirements include supporting information integration through the use of ontologies for solving problems of semantic heterogeneity, and to offer functionality related to concept matching, traversal, consistency and validity checking, and simple reasoning. It should also handle problems of partially available metadata and ontologies – which is particularly relevant in a MANET given the dynamic network topology – as well as dealing with run-time changes in semantic metadata. A set of language and model standards for semantic metadata and ontologies may also have to be supported. In addition there may be need for some automatic or semi-automatic dynamic gathering of metadata. Internally in the system, ontologies are useful for describing categories of devices; events; resources; access levels etc. Externally, i.e., to the application level, the focus is on use and sharing of existing domain ontologies from the participating organisations. We assume that ontology creation is handled *a priori* in the organisations.

The main objective for *Profile and Context Management* is to support personalisation, ranking and filtering of information through profiling and context. A profile gives the "what" and "who" of an entity, and contains fairly static information. Examples are device type and resources, user identification, roles and preferences. Contexts on the other hand, contain information concerning the "where", "when" and "why" of an entity, for instance the location, time, and situation. This is a more dynamic kind of information, changing along with shifts in situations and locations (e.g., node movement).

In addition to the three metadata handling components, the KM has two tool components, providing functionality related to searching/querying and parsing. The *Query Manager* should support different approaches to querying, and the dispatching of queries. It should also allow creation of simple queries, and support filtering and ranking of results. Given that XML is becoming a *de facto* standard for information exchange, e.g. in health and medical domains [20] [21], we will need an *XML Parser*, offering services related to parsing of XML documents. In addition, if RDF and RDF-based ontology languages are used, an RDF parser will also be needed.

3.2 Related Work

Approaches for solving syntactic and structural heterogeneity have been addressed in standardisation work [20] [21] [23] and in the distributed database domain.

Approaches for information and ontology integration, and the use of ontologies for solving semantic heterogeneity, can be found in [9] [12]. In information integration, ontologies can be used as a solution to solve semantic heterogeneity, both to explicitly describe the semantics of information sources, and as a language for translation. There are several approaches to information integration, Wache et al [9] describe three general approaches: *Single ontology approach* – having one global ontology with shared semantics, that all users have to conform to; *Multiple ontology approach* – where mapping between (each pair of) ontologies is required; and *Hybrid ontology approach* – where multiple ontologies are built on top of or linked to a shared vocabulary of basic terms which may function like a bridge or translation between ontologies. We believe a variety of the hybrid approach will be the most advantageous in relation to our Knowledge Manager.

Ontology-based solutions for information sharing and Semantic Web are described in [10] [11]. Different XML parsing paradigms are presented in [16] [17], and the use of XML as standard for information exchange in rescue organisations is addressed by [18] [20] [22] [23] [24].

Shark [8] [13] is an approach where topic-based knowledge ports are used to handle knowledge management tasks. This is a three-layer approach similar to ours. The knowledge ports are defined as topic types and declare topics for knowledge exchange. The (mobile) users form groups, and both intra- and inter-group knowledge sharing and exchange is possible. The architecture relies on stationary server nodes for knowledge and synchronisation management, which is a drawback in MANET scenarios. Topic Maps is introduced in [19].

The approach in MoGATU [5] is relevant to knowledge and context management in their use of profiles and ontologies for filtering and prioritisation of data. In this approach, information managers (InforMa) functioning as local metadata repositories are used for metadata handling, enabling semantic-based caching. DAML is used as a common language for metadata representation. There is no global view of knowledge available in the network.

A service oriented and data-centric approach is DBGlobe [7][14], where devices form data sharing communities that together make up an ad-hoc database of the collection of data on devices that exist around a specific context (e.g., location or user). Profile and metadata describing each mobile device, including context and which resources are offered, is stored on fixed network servers. The servers are also used to keep track of the movement of mobile units. Service location and query routing is realised through the use of distributed indexes based on Bloom filters.

Relying on a fixed network is a drawback both in relation to emergency and rescue scenarios and MANETs.

By providing a global database abstraction over a MANET, AmbientDB [6] [15] adds high-level data management functionalities to a distributed middleware layer. This is a non-centralised, ad-hoc/dynamic approach using structured queries. Indexing is realised through Distributed Hash Tables (DHT). This approach constitutes a full-fledged distributed database system for MANETs, but does not support use of ontologies or methods from knowledge management.

4 Design and Implementation of the Data Dictionary Manager

In this part, we first present our three-layered approach, and then go on to describe the Data Dictionary Manager, which is the main component for metadata management in the KM. Every node has a DDM, each containing a *Local Data Dictionary* (LDD) and a (global) *Semantic Linked Distributed Data Dictionary* (SDDD). The DDM uses services from the *Semantic Metadata and Ontology Framework* for discovering concept structure and relationships. Alternative approaches that we have considered for storage and distribution of a global data dictionary, are described in [2].

4.1 Three-Layered Approach

Conceptually, our approach consists of three layers of increasing abstraction levels: information layer, semantic context layer, and knowledge layer. At the lowest abstraction level, the information layer concerns the information to be shared on the nodes, represented by metadata descriptions of information item structure and content. The semantic context provides meaning through relating the metadata descriptions to vocabularies/ontologies defining the domain in question. These ontologies constitute the highest level of abstraction – the knowledge layer. Topic Maps or RDF can be used to realise links from semantic context layer to information layer; our solution differs in that we have (indirect) links from the information layer to the semantic context layer, i.e., we support two-way links.

The main reason why we believe two-way links are useful in a MANET is that it will contribute to keeping communication needed for query and search related to information retrieval as low as possible, which is highly desirable in mobile wireless networks. The semantic context layer can function like a (semantic) index structure – indicating which nodes to send queries to (assuming semantically related nodes have a higher probability of having relevant information). Through functioning similarly to a *semantic overlay network*, having two-way links will enable traversal among (semantically) related nodes, allowing query/search to be limited to these, e.g., using partly local query/searches, or query on a cluster of related nodes. By the merging of the semantic contexts of information resources on mobile nodes when forming a network, each node's global view of what knowledge is available for sharing in the MANET is gradually expanded and updated. Thus nodes (information items on nodes) can be said to be related semantically both "vertically" to semantic context and domain ontologies, and "horizontally" to other related nodes. Neither Topic Maps nor RDF have links from (metadata descriptions of) resources to concept/topic (although this may possibly be achieved by extension).

At the time of network bootstrap, and before a node has discovered any other nodes in network range, the "world" and the network available as known to the node consists only of the node itself and what is stored locally (metadata registered in the LDD for sharing). So initially, the global view of the knowledge in the MANET is only what exists locally on the single node. When the node comes into network range with other nodes, the contents of its global view will be exchanged and merged with the global views of its neighbouring nodes, i.e., each node exchanges the contents of its SDDD with the SDDD of the neighbouring node. Keeping an SDDD on all nodes at all times, solves many of the problems encountered at bootstrap time (e.g., where to find vocabulary terms/concepts for describing and linking resources, and where the *Distributed Event Notification System* (DENS) can look for initial subscriptions).

In Figure 2 a conceptual view of this layered solution can be seen. The figure also indicates that our solution will be realised through metadata management, using local and semantically linked distributed data dictionaries (LDDs and SDDDs) on the mobile nodes.

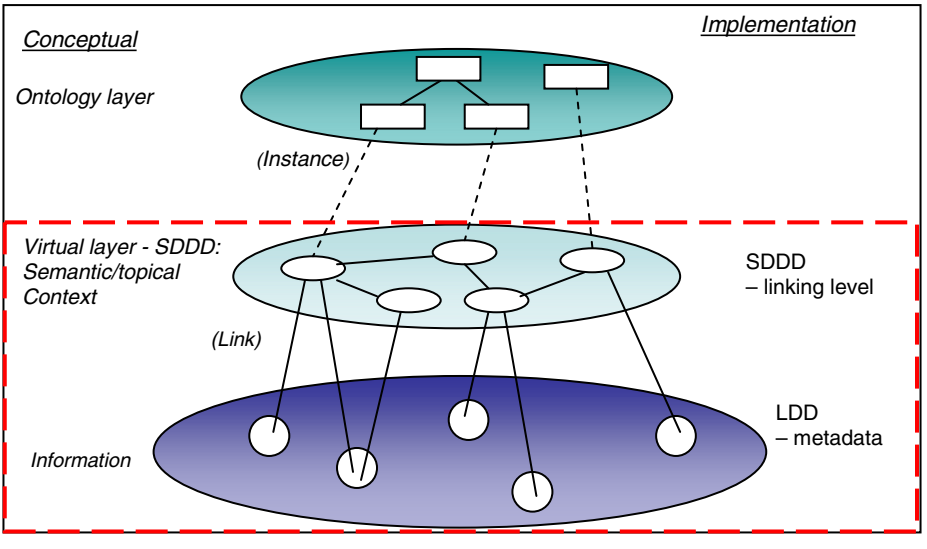


Fig. 2. Layered solution

4.2 Data Dictionary Manager

The DDM is responsible for storage and management of metadata for items registered to be shared, as well as responsibility for structure and content description metadata. The main requirements of the DDM are listed below:

- Manage organisation and update of data dictionaries:
  - Add/update/delete content of LDD and SDDD
  - Manage merging of SDDD
- Keep track of data availability
- Support request for information from data dictionaries (search and traversal)
- Link validity checking.

The DDM offers APIs to the application layer; e.g., adding and updating items to LDD, and requesting (querying) information globally (through SDDDs). It should also have APIs for services offered internally in the system to the other (middleware) components as well as other KM components. It has protocols for communicating with DDMs on other nodes, e.g., for traversal of semantic net (in querying), and exchanging SDDD contents at merge. In addition, an interface towards the data layer is needed for lookups (e.g., a file storage system, or a DBMS) locally on the node.

4.3 Local Data Dictionary

The metadata contained in the LDD includes metadata for ontologies/vocabularies, metadata models, and system related data, all kinds of data that also are stored on a node, described by metadata and shared in the network. Metadata is added at the application level, using appropriate concepts and terms from vocabularies (/ontologies) and metadata standards agreed upon in the *a priori* phase. These vocabularies are also described, and their metadata registered, in the LDD. Vocabulary mapping of shared and local vocabularies is achieved through employing services offered by the *Semantic Metadata and Ontology Framework*. Some kind of consistency check and cleanup of the LDD to remove outdated information should be conducted periodically. How the registering of items is done by the DDM is described in more detail in the section on LDD and SDDD life cycle.

For the purpose of illustration, we here introduce a small and very simple vocabulary consisting of these few concepts and relations:

concept(CasualtyReg), concept(CasualtyID), concept(CasualtyCode),
concept(CodeRed), concept(CodeYellow), concept(CodeGreen),
relation(isPartOf), relation(isKindOf),
isPartOf(CasualtyID, CasualtyReg), isPartOf(CasualtyCode, CasualtyReg),
isKindOf(CodeGreen, CasualtyCode), isKindOf(CodeRed, CasualtyCode),
isKindOf(CodeYellow, CasualtyCode)

By using concepts from a vocabulary in the metadata descriptions, the information items are indirectly linked to the semantic context level – realised by the SDDD – and to related information (on other nodes). The concepts are used as (property) values in the metadata descriptions. For instance, say we have a data item for registration of a casualty – an instance of 'CasualtyReg'. The metadata description of the item would contain some properties characterising the item, e.g., it having CasualtyCode 'CodeRed'. By simple reasoning, we can follow a link from entry CodeRed in the metadata description, to its corresponding ontology individual 'CodeRed', and further find the kind of concept it is ('CasualtyCode'), and that this is in turn part of a 'CasualtyReg'. Thus we have an indirect link via property values used in metadata descriptions to the semantic context level. The semantic network of concepts and relations can be further traversed, and nodes containing information items belonging to a concept, can be reached. Thus we can find related information through finding semantically related nodes in a MANET.

The LDD can be realised in many ways, for instance as XML structures or as tables in a database. Below is the terminology we use to describe LDD items and structure.

Dictionary Item. This is an ”envelope” or container containing the metadata about some information item. The purpose of having a Dictionary Item is to have a homogeneous “outer shell” common for all entries in the LDD, independent of what kind of information item or metadata it contains. The Dictionary Item provides information regarding identification (a dictionary item id); which type of item it contains metadata about (Info-Item-Type); and the metadata model used.

Info-Item-Type. There are many different kinds of information objects that need describing, e.g., an image, a device profile, component specific information etc. Different types of information items may need different structure of the metadata descriptions. Therefore, we need to have defined Info-Item-Types for all the different kinds.

Info-Item. All Info-Items are of a given Info-Item-Type, and all Info-Items have some common metadata content: item structure (e.g., format), item content (intellectual), semantic context information, and administrative metadata (e.g., access, distribution, number of copies, localisation etc). It will also most likely have something specific to its Info-Item-Type.

Data-Item. This denotes what the metadata is describing – e.g., a document file, sensor data, or a video clip.

4.4 Semantic Linked Distributed Data Dictionary

The main task of the SDDD is to provide linking tables for linking LDD items to a semantic context through concepts used in metadata descriptions. The concepts and relations between them make a semantic network, and the SDDD should allow traversal of this semantic net. The SDDD keeps an overview of sharable items that are known by the node to be available in the MANET, and keeps tables linking concepts and nodes. The SDDD should also do some “self administration”, i.e., distribution and replication of the SDDD itself (the linking tables). In our approach of having an SDDD on every node, the contents of the SDDD is distributed and replicated at merge time when SDDD content of two nodes is exchanged and merged with existing SDDD content on each node. This means full replication and distribution of SDDD content (depending on whether enough resources exist on all nodes).

The SDDD can be said to be a ‘virtual’ data dictionary in the sense that it does not store any copy, abstract, or summary of LDD contents, but is realised as a set of *linking tables* linking concepts to where the resource is found (the node). It also includes tables for relations between concepts. An important issue in this context is the *level of granularity* for linking. We have chosen to use two levels of granularity, depending on whether the link points to items in a local LDD, or to items found at a different node. Pointing to the node will be sufficient in the latter case, the rest of the link being resolved locally on the node where the item is, while when linking to the LDD on the same node, the link points to the Dictionary Item (in the LDD). How the merging and linking is done by the DDM is described in more detail in the next section.

4.5 LDD and SDDD Life Cycle

At bootstrap time, the DDM first creates the LDD, or restores it from permanent storage. There will always be some initial content to be registered in LDD, e.g.,

metadata of vocabularies and standards to be used, and other information from the *a priori* phase. The application should be able to decide whether the LDD should keep the previous content. If the previous content belonged to the same rescue context, e.g., due to a temporary shut down, we assume that the DDM persisted the LDD and thus can restore the LDD content. At re-startup, pointers to resources in the LDD should always be checked and confirmed so that the content is as accurate as possible.

Next, the SDDD is created. The LDD content is scanned for concepts, and these are added to the SDDD tables together with the Dictionary Item ID of resources related to the concept. As long as the resource pointed to by a link is registered in the LDD on the node, the link points to the Dictionary Item (DI) in the LDD. After a merge, links can point to items registered for sharing on a different node. In this case, the link points to the other node as such, and not to a DI in the LDD of that node. The link is then resolved locally on the node. This means that a higher level of granularity is used when links are pointing locally on a node, while a lower granularity is used when pointing to a different node.

Items are registered in the LDD the following way: The application requests the DDM to add an item to the LDD, providing the metadata to be stored. A new DI entry is created, and the LDD tables are updated with the new information. The DDM then initiates update of the SDDD.

When two nodes come into network range, they will exchange SDDD content, and depending on available resources, each node will merge the received content with its own SDDD. This merge can be viewed as an expansion of each node's worldview. The merge is completed in the following steps: 1) concepts are matched with existing concepts in the SDDD. 2) If there is a concept match, the SDDD linking table will be updated with node-IDs (node links). In the case of new concepts, the DDM adds the concepts and node-IDs to the SDDD linking table providing there are enough resources. Otherwise it adds no concepts, only a link to the node that expands its global view. 3) The DDM updates the availability table in the SDDD.

When there are changes in the LDD-related information resources, the LDD has to be updated to reflect these changes. The possible ways this is realised, will depend on the LDD implementation as well as the underlying system for storage used on the node. If the underlying storage system is a database system, triggers can be used. The availability information in the SDDD is updated in a lazy fashion, i.e., updating node availability when a node is discovered to be out of reach instead of polling for changes. In addition, both LDD and SDDD will have some periodical clean-up of links.

On shut-down, the DDM should store the contents of both LDD and SDDD in permanent storage. For the LDD, pointers to resources will be checked and updated on next start-up. In the case of SDDD, if there is limited storage space on a node, the SDDD may be discharged and then rebuilt on start-up. Some clean-up of links may be conducted at start-up, but through the combination of lazy update and periodical clean-up, the SDDD will gradually become updated. Prioritizing metadata of different types and importance should also be considered.

4.6 Example and Use Cases

In our mini-scenario, two doctors at a rescue site carry devices running an application for casualty registration and middleware for information sharing in MANETs. Each of the doctors registers one or more casualties before coming into network range and forming a network.

Below is a brief walk-through of the mini-scenario.

- At starting point (after bootstrap): Both the LDD and SDDD have been created on all nodes. Some *a priori* info (e.g., metadata describing the used vocabulary) has been added to the LDD. The SDDD contains concepts and links to DIs in its local LDD.
- The two (doctor-) nodes, Node1 and Node2, are not in range.
- On each node, metadata of casualty registration information is added to the LDD, and the SDDD is updated with appropriate concepts and links (and availability information).
- The nodes come into range and form a network, exchange SDDD contents, and update their SDDDs.

The above mini-scenario is split into two use cases presented below.

Use case 1: Registering metadata of information to be shared.

1. A doctor uses the casualty registration application to register casualty data about an injured passenger
2. The application creates a metadata description using concepts from a vocabulary
3. The application requests the DDM to add the metadata description to the LDD for sharing
4. DDM adds received metadata to LDD
5. DDM initiates that the SDDD is updated with the new information

The result of this use case is that the casualty registration data is registered with its describing metadata in LDD, and is available for sharing in the network.

Use case 2: Merging global view of sharable information.

1. Node1 comes into network range with Node2; the nodes perform necessary procedures of security and recognition (not elaborated here)
2. The DDM on each node requests exchange of metadata to expand the global view (the SDDD)
3. The DDMs exchange metadata
4. The DDM initiates merging of received content to its global view (the SDDD)

The result of this use case is that the nodes share a common view of what information is available for sharing globally in the network.

4.7 Implementation and Testing

Some of the factors we considered when choosing a DBMS to build on include it having a data dictionary/directory, code availability, that it be lightweight and can run on several platforms, and that it is standards based. Ideally, we would have liked to use a distributed DBMS for MANETs, but have not found any suitable existing systems that are available. Therefore, to test our ideas, we have chosen to build on Derby [25], which is an open-source, light footprint DBMS. It is standards based (SQL-99), and can run on different platforms and in different environments, in embedded mode or in client/server mode. In our case, it will be part of a DDM module for testing. Building on (parts of) an existing DBMS will save time and allow us to focus on our metadata management solution.

A small example to show how the LDD and SDDD can be realised using database tables, and to demonstrate merging of SDDDs, is given below. The basis for this example is the use cases presented in the previous section. We assume that a common vocabulary (introduced in 4.3) and metadata model is used. To give an indication of a possible LDD and SDDD structure, we show examples of LDD and SDDD schemas related to our working example below. Only some example attributes are shown in the schema.

LDD_Dict_Item (**DI-id**, InfoItem-Type, Metadata_model_DI, InfoItem-ID)
InfoItem_DataStructure (**InfoItem-id**, Metadata_model_InfoItem, Format)
InfoItem_Content(**InfoItem-id**, Description, Owner)
InfoItem_Context(**InfoItem-id**, Vocabulary_Used, TypeOfRescueOperation, NameOfRescueSite)
InfoItem_Admin(**InfoItem-id**, Location, Resource_Owner, Access_level)
InfoItem_TypeSpecific(**InfoItem-id**, InfoItem-Type)

The SDDD schema is for linking tables, availability tracking, and concept names. The attribute MetaResID (metadata resource id) in the tables 'SDDD_Linking' and 'SDDD_Availability' denotes node ID or Dictionary Item ID depending on whether the link points to metadata descriptions in the local LDD or on another node (using different levels of granularity).

SDDD_Linking(**ConceptID**, **MetaResID**)
SDDD_Availability(**MetaResID**, Available)
SDDD_Concept(**ConceptID**, ConceptName)
SDDD_Relation(**RelationID**, RelationName)
SDDD_RelatedConcept(**ConceptID1, ConceptID2, RelationID**)

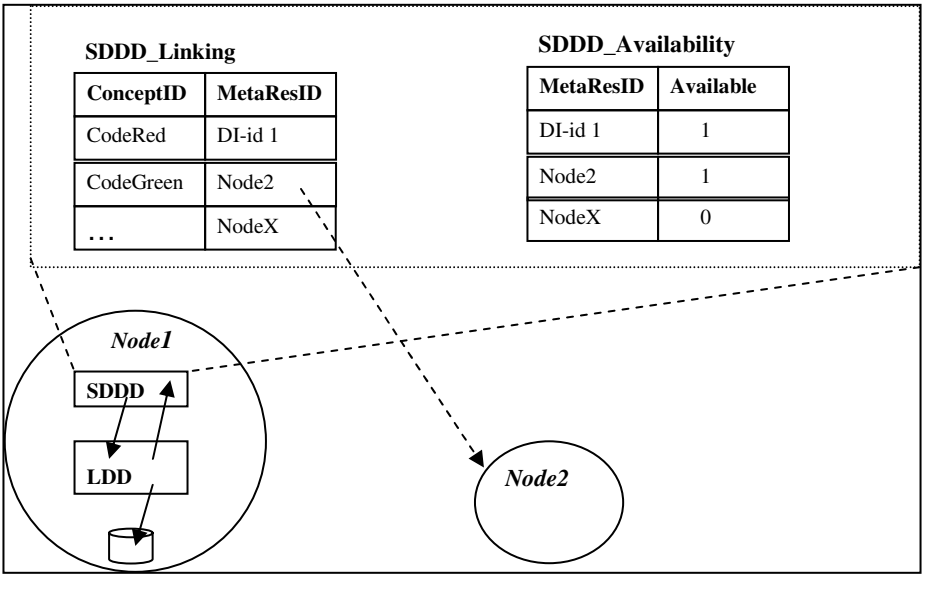


Fig. 3. Illustrating SDDD contents after merge

Figure 3 illustrates what the contents of SDDD linking and availability tables will be after a merge. The illustration shows a case where both nodes have enough resources to add concepts to their SDDD tables. The tables show the contents of (two of) the SDDD tables for linking concept to resource (node) and for tracking availability on Node1 after a merge between Node1 and Node2. The ConceptID 'CodeRed' points to a Dictionary Item in the local LDD on Node1, and the ConceptID 'CodeGreen' points to Node2. When Node1 wants to find the linked-to metadata for the resource on Node2, the DDM on Node1 will query Node 2 DDM for the item. Node2 DDM will query its SDDD for the item's DI-id, then ask its local LDD for the information and return this to Node1.

5 Conclusions and Future Work

In this paper we have shown that knowledge management and particularly metadata management is essential to solving many of the challenges related to information sharing and integration in MANETs for rescue and emergency scenarios. We believe that the presented multi-layer data dictionary approach is a step in the right direction, offering possible solutions towards global views of available knowledge and keeping communication for query and search at a low level.

We are currently continuing our work on the detailed design and implementation of the DDM module to test our ideas. In time, we would like to incorporate the DDM in the network emulator testbed, NEMAN [4], which has been developed in the Ad-Hoc InfoWare project. Some of the issues we think may be interesting to explore include different eager/lazy propagation policies for metadata of different types, how to handle priority of metadata in different situations, measurements of how fast the dissemination of metadata is achieved in different situations, and possibly also measurements of how much SDDD deviates from a genuine global data dictionary over time. In the future, we may also consider expanding our solution to use Distributed Hash Tables for optimisation.

References

1. Plagemann, T., et al: Middleware Services for Information Sharing in Mobile Ad-hoc Networks - Challenges and Approach. Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004
2. Sanderson, N., Goebel, V., Munthe-Kaas, E.: Knowledge Management in Mobile Ad-hoc Networks for Rescue Scenarios. Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications, The 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 2004
3. Munthe-Kaas, E., Drugan, O., Goebel, V., Plagemann, T., Pužar, M., Sanderson, N., Skjelsvik, K. S.: Mobile Middleware for Rescue and Emergency Scenarios, Mobile Middleware, Paolo Bellavista and Antonio Corradi, ed., CRC Press, 2006 (to appear)
4. Puzar, M., Plagemann, T.: NEMAN: A Network Emulator for Mobile Ad-Hoc Networks, 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

5. Perich, F., Avancha, S., Chakraborty, D., Joshi, A., Yesha, Y.: Profile Driven Data Management for Pervasive Environments, Proceedings 13th International Conference on Database and Expert Systems Applications (DEXA 2002), Aix-en-Provence, France, September 2002
6. Fontijn, W., Boncz, P.: AmbientDB: - P2P Data Management Middleware for Ambient Intelligence, Middleware for Pervasive Computing Environment I (PERWARE04) Workshop at the 2nd Conference on Pervasive Computing (PERCOM 2004), Orlando, Florida, USA, March, 2004.
7. Pfoser, D., Pitoura, E., and Tryfona, N.: Metadata Modeling in a Global Computing Environment, Proc. of the 10th ACM International Symposium on Advances in Geographic Information Systems, pp 68-73, McLean, VA November 8-9, 2002.
8. Schwotzer, T., Geihs, K.: Shark - a System for Management, Synchronization and Exchange of Knowledge in Mobile User Groups, Proceedings 2nd International Conference on Knowledge Management (I-KNOW '02), Graz, Austria, July 2002, pp. 149-156
9. Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hubner, S.: Ontology-based integration of information - a survey of existing approaches. In Stuckenschmidt, H., ed., IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, Washington, USA, August 2001, pp.108--117.
10. Davies, J., Fensel, D., Van Harmelen, F. (eds.): Towards the Semantic Web: ontology-driven knowledge management, John Wiley & Sons Ltd 2003. ISBN 0-470-84867-7
11. Fensel, D. et.al: Ontology-Based Knowledge Management, IEEE Computer, November 2002, pp.56-59.
12. Stuckenschmidt, H. and van Harmelen, F.: Information Sharing on the Semantic Web, ISBN 3-54-20594-2, Springer 2005
13. Schwotzer, T., Geihs, K.: Mobiles verteiltes Wissen: Modellierung, Speicherung und Austausch, Datenbank-Spektrum 5/2003, pp.30-39.
14. Pitoura, D. et.al: DBGlobe: A Service-Oriented P2P System for Global Computing, ACM SIGMOD Record, pp 77-82, vol.32/3, Sept. 2003.
15. Boncz, P. A. , Treijtel, C.: AmbientDB: relational query processing in a P2P network. Technical Report INS-R0306, CWI, Amsterdam, The Netherlands, June 2003., http://www.cwi.nl/ftp/CWIreports/INS/INS-R0306.pdf
16. A Survey of APIs and Tecniques for Processing XML, http://www.xml.com/pub/a/2003/07/09/xmlapis.html
17. XML and Perl: Now Lets's Start Digging, http://www.informit.com/articles/article.asp?p=30010
18. KITH rapport nr R25/02, http://www.kith.no/arkiv/rapporter/rammeverk-v09-testing-2.pdf, ISBN 82-7846-151-1.
19. Pepper, S.: The TAO of Topic Maps, http://www.ontopia.net/topicmaps/materials/tao.html
20. http://www.kith.no/
21. http://www.centc251.org/
22. http://www.ebxml.org/
23. http://www.oasis-open.org/news/oasis_news_03_29_04.php
24. http://www.oasis-open.org/
25. http://incubator.apache.org/derby/index.html

Managing Petri Nets in MOF Repositories

Hélio L. dos Santos, Paulo R.M. Maciel, Nelson S. Rosa, and Roberto S.M. Barros

Universidade Federal de Pernambuco - UFPE
{hls, prmm, nsr, roberto}@cin.ufpe.br

The necessity of interoperability among Petri net tools has led to the development of the PNML (Petri Net Markup Language) standard. By adopting PNML, tools mainly concentrated on modeling activities should generate PNML files to be analyzed by analysis-specific Petri net tools. In this context, we propose an extension to the PNML based on MOF (Meta Object Facility). The implementation of the MOF metamodel enables us to manager Petri net specifications within MOF repositories. In order to illustrate our approach, we present a case study.

1 Introduction

Over the last three decades a myriads of Petri nets tools have been developed for supporting modeling, analysis and verification of such models. Since mid 1990's demands for interoperability among these tools turned to be an important concern in Petri net community. The interoperability makes it easier for the users to get the benefit of using several tools. However, some factors such as different types of Petri nets and absence of a standard make this process hand to cope with [JKW2000].

Nevertheless, PNML (*Petri Net Markup Language*) proposed in the *Petri Net Kernel* Project [KIWE1999] is emerging as a standard for the interchange and storage of Petri nets. PNML is based on XML (*Extensible Markup Language*) [ABK2000] and introduces the major characteristics of Petri nets [BILL2003]. Specifics aspects of each type of Petri nets are to be defined as extensions using PNTD (*Petri Net Type Definition*), expressed as a DTD or XML Schema [AHAY2001].

Because it is based on XML, PNML presents some advantages as a standard for interchange and storage of data [AHAY2001]: XML is an open standard, established by an international organization (W3C - *Word Wide Web Consortiun*), platform and vendor independent; it supports the ISO Unicode Standard to represent characters of any dialects; it has a textual syntax; it is supported by a large number of tools and; it is flexible enough to store all the types of Petri nets.

This paper proposes an extension to PNML for MOF repositories (*Meta Object Facility*). A metamodel is a set of inter-related metadata used to define models [MAIN2000, TANN2002]. The metamodel formally describes the elements of the model, its syntax and semantics. Concepts defined in the PNML metamodel include "Place, "Transition" and "arc".

MOF [MOF1999] is a standard created by the OMG (*Object Management Group*) to support the construction of metamodels. One of its aims is to permit interoperability among tools in order to produce and share metadata (instance of metamodels).

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1381 – 1397, 2005.

This approach makes it possible for tools which implement the MOF specification to manage (create, query, change and delete) Petri nets in MOF repositories and use a unique set of interfaces. At this point, MOF specification defines a mapping for IDL (*Interface Definition Language*) CORBA [MOF1999] and JMI (*Java Metadata Interface*) [JMI2002]. Some tools that support MOF are MDR (*Metadata Repository*) [MDR2002], dMOF [DMOF2001] and UREP (*Universal Repostiory*). Some of them map objects to CORBA (dMOF) and others to JMI (MDR); some store objects in DBMS.

Another advantage is the usage of XMI, as the standard for metadata representation. XMI is supported by several tools aimed at different areas, such as software development (*Netbeans*); system modeling specification (*Rational Rose and Argo UML*); Data warehousing (*DB2 Warehouse Manager and Oracle Warehouse Builder*); metadata repositories (*UREP, MDR and dMOF*).

The following sections present related work, MOF standard and its architecture and the standard for mapping it to Java and CORBA interfaces. Then, we present the metamodel for Petri net systems, the interfaces generated from this metamodel, the mapping from PNML metamodel to PNML and *vice-versa*, and a case study that depicts the interface generated for PNML metamodel. Finally we present our conclusions and proposed future works.

2 Related Work

MOF metamodel proposed in this paper is based on PNML, which was initially proposed as the standard for storage in the PNK (*Petri Net Kernel)* project. This project provides infrastructure to support development of Petri nets tools. It is currently implemented in Java and Python [KIWE1999].

Breton [BRBE2001] proposed a MOF metamodel to Petri nets. However, his work is conceptual and does not present an approach to implement and use Petri nets. Moreover, it does not use PNML, a standard used by several Petri net tools.

Other MOF metamodels include UML [UML2001], which is a language for software specification, visualization, implementation and documentation, and CWM (*Common Warehouse Metamodel*) [PCTM2001], which is the OMG standard for Data Warehousing tools integration. This integration is based on metadata sharing. CWM defines a set of metamodels to the several phases of Data Warehousing.

Santos *et al* present two MOF metamodels to manage XML and DTD metadata [SBF2003a] as well as the integration of the W3C metadata standards RDF (*Resource Description Framework*) and RDF Schema and MOF.

Another important work by Sun Microsystems is a metamodel to Java [DEMA2002], which proposes a mean of managing Java Code in MOF repositories.

3 Petri Net Systems

Petri net is collective term used to describe a family of graphical formalism based on specific assumptions about the world of information processing. The most basic notions are the concepts of local states (places), local actions (transitions) and their mutual relationships [ROZ1998a].

One of main attractions of Petri nets is the way in which the basic aspects of concurrent systems are dealt with both conceptually and mathematically. The Petri net family of formalisms can be divided into three layers: the most fundamental layer suits for basic research on concurrent system foundations. The intermediate level includes models that provide a more compact representation of systems by folding repetitive structures of basic models. The third level includes the so called high-level nets, more adequate for real applications, use algebraic and logical mechanism for allowing compact representations [ROZ1998b].

With respect to the concurrency interpretation, at least four possibilities have been described: transition semantics, single step semantics, step semantics and Process nets [PET1981, REIS1985, ROZ1998b]. Transition semantics describes concurrency by the interleaving of individual transition firings (action execution). Step semantics, on the other hand, are true concurrent interpretations. Single step semantics considers the simultaneous firing of concurrent transitions (step), but it does not allow the simultaneous firing of same transition as the step semantics does. (Single) Step semantics may be viewed as a formalization of simultaneity, but in general actions do not occur simultaneously, they indeed may arbitrarily overlap in time. Therefore, one system's component may execute a given number of actions and another independent component may execute a distinct number of actions. Process nets aims to formalize the notion of concurrent runs [ROZ1998b].

Several extensions have been proposed to the autonomous Petri net models. Among them, nets with inhibitor arcs, net with priorities and several interpretations of time have been considered. Time has been associated to actions and local states under deterministic and stochastic views [RAMCHA, MERL1976, MOL1981, MAR1985, ZUBE1991].

The multitude of distinct Petri net types, tools and file formats motivated the creation of a XML-based interchange format for Petri Nets. The starting point of this standardization effort was the International Conference on Application and Theroy of Petri Nets 2000, when several proposals for XML-based interchanged formats were presented. Amongst them, Petri Net Markup Language (PNML) [JKW2000] is the most prominent.

The following principles governed the design of PNML:

- Readability: the format should be readable and editable with conventional text editor;
- Universality: the format should be able to represent any version of Petri nets with any extensions.
- Mutuality: the user should be able to extract as much information as possible from a Petri net - even if the Petri net type is unknown.

Readability is guaranteed by using XML. Universality is achieved by attaching information required by a particular Petri net type to the objects of the net.

PNML uses the Petri Net Type Definition (PNTD) to express the additional information attached to the net objects. Finally, mutuality is guaranteed by defining a document called conventions. It contains a set of standardized labels, along with their semantics description and typical use.

The graphical net representation is very useful since it allows for process visualization and its communication. A Petri net graph is composed of two kinds of

vertices: places represented by circles, and transitions represented by boxes, rectangles or lines. Directed arcs interconnect these vertices. According to the multiset theory, Place-Transition nets may be defined as a 6-tuple: set of places (local states and resources), set of transitions (actions), input multi-set (pre-conditions), output multi-set (post-conditions) , a mapping called place capacities and a mapping called marking (global state.

The Figure 1 presents an example of Petri nets which models the Producer Consumer Problem. In this example, two processes share a *buffer* of fix length. The producer process includes information in the *buffer*, while the consumer process removes one. If the *buffer* is full, the producer process is blocked, i.e. it cannot produce. Likewise, if the *buffer* is empty, the process consumer is blocked and it cannot consume information.

In this example, only transition *t1* is enabled, since place *p1* is marked. If *t1* is fired, then one token is removed from place *p1* and one token stored in place *p2*. The number of removed and stored tokens in such places is one[1] due to the weight of arcs interconnecting these places to fired transition. When *t1* is fired, the system is ready to deliver information by the transition (action) *t2*.

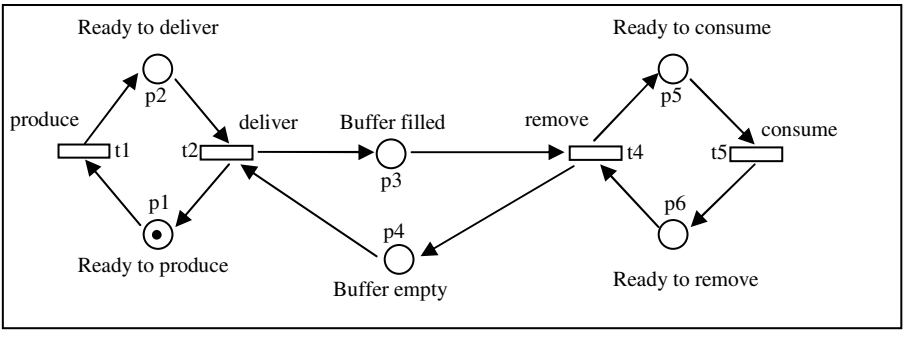


Fig. 1. ProducerCosumer Petri Net

4 MOF – Meta Object Facility

MOF is an abstract language and a framework to specify, build and manage platform independent metamodels. Examples include the UML, CWM and MOF metamodels. MOF also supports a set of rules to implement repositories, which deal with metadata described by the metamodels. Theses rules define a standard mapping between the MOF metamodels and a set of APIs to manage metadata, instances of the metamodels. For instance, the *MOF -> IDL CORBA* mapping is applied to MOF metamodels (UML, CWM) to generate CORBA interfaces that manage the corresponding metadata, instance of metamodel. The *MOF -> Java* mapping is similar and generates Java interfaces using the JMI standard. This work uses JMI.

[1] When the value of the arc is greater one, this is subscribed around the arc.

The MOF specification is composed of:

- A formal definition of the MOF meta-metamodel, an abstract language to define metamodels;
- Rules to map MOF metamodels to an implementation technology such as CORBA or Java; and
- The XMI standard to interchanging metadata and metamodels among tools using XML. XMI defines rules to map MOF metamodels and metadata to XML documents.

4.1 OMG Metadata Architecture

MOF is an extensible framework, i.e., new metadata standards may be added to it. It is based on a four-layer architecture, called the OMG Metadata Architecture [MOF1999], which is presented in Table 1.

The instance of one layer is modeled by an instance of the next layer. So, M0 layer, where there are data, is modeled using UML models (such as class diagrams), which are stored in layer M1. Accordingly, M1 is modeled by the UML metamodel, layer 2, and uses constructors such as classes and relationships. This metamodel is an instance of the MOF model (also called meta-metamodel). Another example, a CWM model in layer M1 is an instance of the CWM metamodel of M2. Extensibility is achieved through the possibility of adding classes that are instances of classes of the immediately superior layer.

Table 1. OMG Metadata Architecture

MOF Level	Used terms	Examples
M3	Meta-Metamodel	MOF Model
M2	Metamodel, Meta-Metadata	UML and CWM Metamodel
M1	Models, Metadata	UML Models – class diagrams, Relation schemas, instance of CWM Metamodel of the M2 layer
M0	Data, Objects	Data warehouse data

4.2 The MOF Model

MOF model, which is in the M3 layer, is presented in this subsection. MOF model is its own metamodel, i.e., MOF is described in terms of itself. MOF specification describes itself using natural language and using UML (*Unified Modeling Language*) [UML2001], tables and OCL (*Object Constraint Language*) expressions. UML is used because of its graphical representation of the models, which makes it easier to read. It does not define the semantics of the MOF model, which is completely defined in the MOF specification and does not depend on any other model. MOF does not specify which language is to be used to define constraints of the metamodels, even though it uses OCL as standard.

MOF is an object oriented model and includes a set of modeling elements that are used to build the metamodels and rules to use them. Examples of these elements are classes, associations, packages, data types, etc.

4.3 Meta-Objects

Interfaces created by the mapping MOF-> *Specific platform* share a common set of four kinds of meta-objects: *instance*, *class proxy, association* and *package*.

Package – An instance of the *Package* class. A meta-object package represents a container of other kinds of meta-objects.

Class proxy – Each class in layer M2 has a corresponding class proxy. There is a proxy object to each class of layer M2. This kind of meta-object produces meta-objects of type instance.

Instance – Instances of classes in layer M2, i.e., of the metamodels, are represented by meta-objects of type instance. A meta-object instance manipulates the states of the attributes and references of the classes in layer M2.

Association – Theses objects manipulate the collection of links corresponding to the associations of layer M2. They are static objects and its containers are meta-objects package. The interfaces of the association objects are operations to retrieve a link in the set of links, to add, change, and remove links in the set of links, and to retrieve the set of links.

4.3.1 The MOF -> CORBA IDL/Java Mapping

The MOF -> CORBA IDL mapping is part of the MOF specification. The MOF->Java mapping was defined by the Java Community and was called JMI [JMI2002]. The general rules of the mapping are basically the same for any platform.

The output of these mappings is a set of interfaces to support creating, changing and querying metadata, instances of the MOF metamodels. For examples, should the interfaces be created using the MOF->CORBA IDL mapping, the user may use CORBA clients to access the interface; should it be JMI, the user may use Java clients.

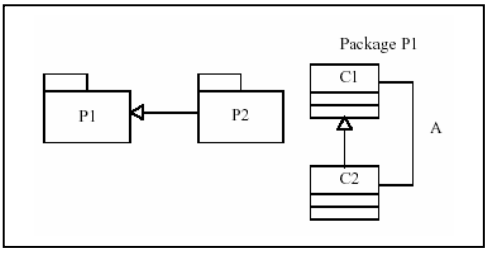


Fig. 2. Example of a MOF metamodel

This subsection describes the inheritance standard of the interfaces mapped from the MOF metamodels. The Figure 2 shows an example of MOF metamodel written in UML, which has two packages P1 and P2. P1 has the C1 and C2 classes, C2 is subclass of C1, and A is an association between C1 and C2. P2 is a sub-package of P1.

The Figure 3 presents the UML diagram that shows the inheritance graph generated from the MOF->Java mapping. The root of the graph is a set of interfaces, which are part of the MOF reflexive package. The interfaces generated from the metamodels inherit, directly or indirectly, from the reflexive interfaces.

The mapping rules say that, for each package and each association of the metamodel, a *package* interface and an *association* interface are created, respectively. For each class of the metamodel, a *proxy* interface and an *instance* interface are created. The inheritance standard is based on the following rules:

- A meta-object *instance* without a super type inherits from *RefObject*; all other meta-object *instances* extend their super types;
- A meta-object *package* without a super type inherits from *RefPackage*; all other meta-object *packages* extend their super types;
- The meta-object *class proxies* extend *RefClass;* and
- The meta-object *associations* extend *RefAssociation.*

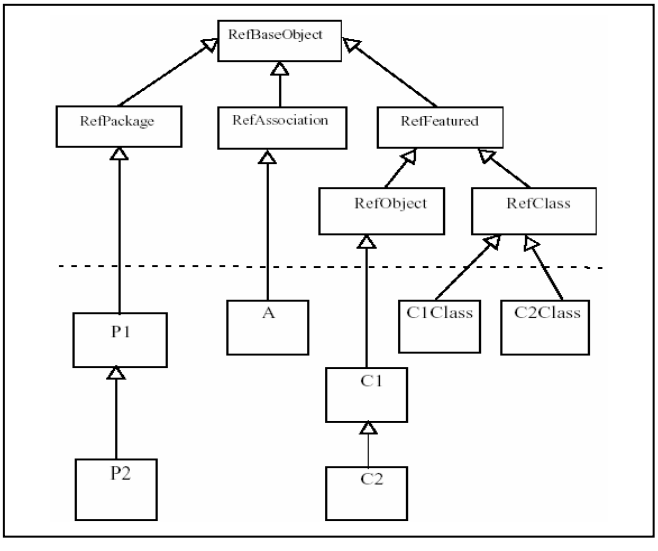


Fig. 3. Example of a MOF Metamodel mapped to Java interfaces

In the examples of Figure 3, two *package* interfaces were created from P1 and P2. The P1 package, which does not have a super type, inherited from *RefPackage*, whereas P2, which does, inherited from P1. The interface for the association between the C1 and C2 classes of the metamodel was created as well. For each class of the metamodel the mapping created two classes: one for the *instance* and another for the *proxy class*. The C1Class and C2Class interfaces represent the *proxy* interfaces created from the classes C1 and C2 of the metamodel, respectively: they inherit from *RefClass* directly. The C1 and C2 interfaces are the interfaces to the meta-objects *instance*. Only the C1 interface inherited from *RefObject*, because it does not have a super type.

5 The Petri Net Metamodel

Modeling and implementation of this metamodel used a set of tools and a sequence of steps, which are described below and presented in Figure 4:

- Definition of the metamodel using UML -> because there are no tools to support MOF modeling, we use UML as the standard to represent the metamodel.
- Generation of XMI document from the metamodel. We used a *Plug-in*[2] of the *Rational Rose* tool, which exports a model of classes to XMI MOF.
- Changing manually the XMI document. This is necessary because we cannot represent in UML everything that is expressed in MOF. These changes allow for a more adequate XMI representation of the metamodel.
- The XMI document is imported to the MOF repository. This was implemented using the MDR[3] (*Metadata Repository*) tool.
- The Java interfaces referring to the metamodel are then created and compiled, following the JMI mapping of MOF metamodels to Java interfaces.
- Implementation of classes to transform Petri net described in the PNML to MOF metamodel and vice-versa.

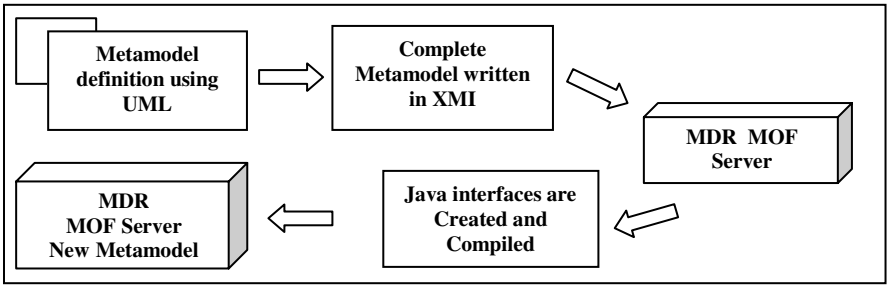


Fig. 4. Steps to model and implement MOF metamodels using MDR

Any tool that implements the MOF specification may be used in the implementation of the metamodel. We chose the MDR tool because it is open-source and generates JMI interfaces. Moreover it has a browser that shows all objects of the MOF repository using a graphical interface. Besides, Java is a popular programming language in academic institutions.

Figure 5 presents the UML diagram of the PNML metamodel. Every component of a MOF metamodel (classes, attributes and associations) must be placed within a package. At this particular case, the PNMLMetamodel package has been created. The shadow classes are abstract ones, i.e., they do not directly instantiate objects.

The PNDocument and PetriNet classes represent a PNML document and a Petri net, respectively. A document may contain many Petri nets through the DocumentContains. A Petri net is made up of objects that are instances of the PNObject class. The Petri net is made up of arcs, nodes and pages, which are modeled by the PNArc, PNNode and PNPage classes, respectively. A page is a container of pages, arcs and nodes objects. The nodes may be either places (PNPlace) or transitions (PNTransition). In addition to places and transitions, a node may also be a reference to either node or transitions. This point is modeled through the PNRefPlace

[2] http://www.rational.com/download/
[3] http://mdr.netbeans.org

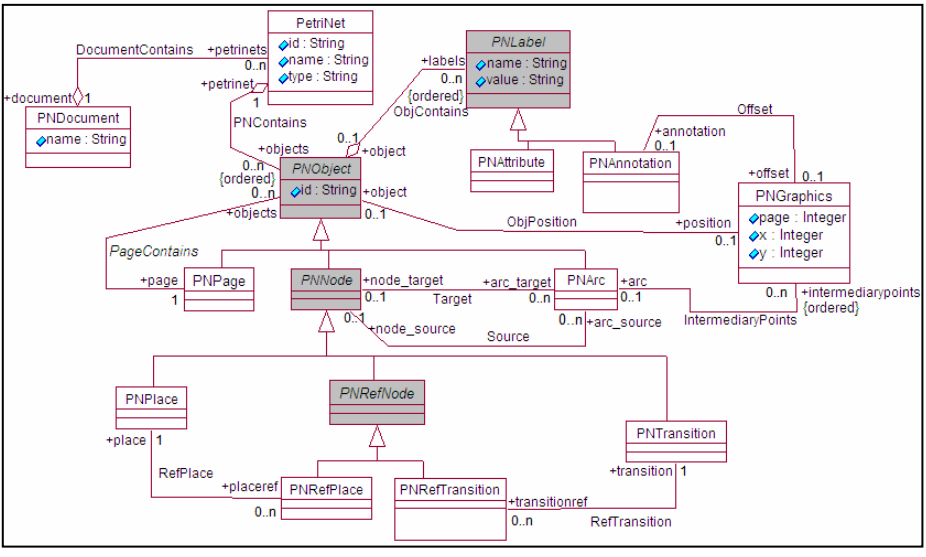


Fig. 5. UML diagram of the PNMLMetamodel

and PNRefTransition classes. Both of them are used to define places and transitions in other pages. A PNML page is a container of PNObject objects.

Each object (instance of the PNObject class) has a set of labels that are modeled by the PNLabel class. These labels define some characteristics of PNObject objects, .e.g, the weight assigned to the arcs, the initial mark of the initial state and so on. They may be attributes or notation and are modeled by the PNAttribute and PNAnnotation classes, respectively. An attribute differentiates from a notation as the last one includes a set of coordinates that enables its exhibition on the screen.

5.1 The Interfaces Created from Petri Net Metamodel

As mentioned before, the MOF has a set of rules to map a MOF metamodel into a set of interfaces that enables the management of medatada. In this case, the metadata represent Petri nets and are stored in the repository.

The developer may store the Petri nets in the repository in two different ways. In the first way, he/she imports the metadata described through XMI. In the second one, the developer may generate interfaces from the metamodels. These interfaces enable the developer to create, update and access the instances of metamodels through Java.

By following the steps mentioned in the precious section, the XMI document is generated and then stored in the MOF repository. Next, the set of interfaces for managing metadata interfaces is also generated.

A basic rule of MOF defines that each class of the metamodel generates two interfaces: the first interface represents instances of the class (the instance metaobjects) and the second one represents a proxy class (the class proxy metaobjects).

Report 1 presents the PetriNet interface generated from the PNML metamodel. This interface contains the methods for accessing and manipulating both the state of attributes and the class references of the metamodel.

Report 1 – The PetriNet Interface of the PNML Metamodel

```
public interface PetriNet extends javax.jmi.reflect.RefObject {
    public java.lang.String getId();
    public void setId(java.lang.String newValue);
    public java.lang.String getName();
    public void setName(java.lang.String newValue);
    public java.lang.String getType();
    public void setType(java.lang.String newValue);
    public Pndocument getDocument();
    public void setDocument(Pndocument newValue);
    public java.util.List getObjects();}
```

Report 2 presents two proxy interfaces, namely PetriNet and PNObject. The proxy interfaces contains operations that create instances of their classes if the class is not an abstract one. The PNObject class of the PNML metamodel is abstract. Hence, the PNOjbectClass interface has no operations for creating PNObject objects.

Report 2 – PetriNetClass and PNObjectClass Interfaces of the PNML Metamodel

```
public interface PetriNetClass extends
                                    javax.jmi.reflect.RefClass {
         public PetriNet createPetriNet();
         public PetriNet createPetriNet(java.lang.String id,
                  java.lang.String name, java.lang.String type);
}
public interface PnobjectClass extends
                  javax.jmi.reflect.RefClass { }
```

An interface and a set of methods are generated for each association of the metamodel, which allows to access, update and insert the instances of associations. Each instance represents a binding between two objects that are instances of the metamodel classes. The PNML metamodel has a PNContains association that defines a Petri net as a set of objects (arcs, places and pages objects). Figure XX presents the PNContains interface.

Report 3 – PNContains Interface of the PNML Metamodel

```
public interface Pncontains extends
                                javax.jmi.reflect.RefAssociation {
  public boolean exists(Pnobject objects, PetriNet petrinet);
  public java.util.List getObjects(PetriNet petrinet);
  public pnmlmetamodel.PetriNet getPetrinet(Pnobject objects);
  public boolean add(Pnobject objects, PetriNet petrinet);
  public boolean remove(Pnobject objects, PetriNet petrinet);
  }
```

An interface is generated for each package of the metamodel. The interface includes methods for accessing the proxys objects that refers to classes and association of the metamodel. Report 4 presents the interface for the *PNMLMetamodelPackage* package.

Report 4 – PNMLMetamodelPackage Interface of the PNML Metamodel

```
public interface PnmlmetamodelPackage extends
                                javax.jmi.reflect.RefPackage {
    public PndocumentClass getPndocument();
    public PetriNetClass getPetriNet();
    public PnobjectClass getPnobject();
    public PnlabelClass getPnlabel();
    public PnattributeClass getPnattribute();
    public DocumentContains getDocumentContains();
    public Pncontains getPncontains();
    public Target getTarget();
    public Source getSource();
    …
    }
```

As the interfaces have been generated, they must be compiled. Next section illustrates how to use the metamodels through the interfaces that have been generated.

6 PNML to MOF and *Vice-Versa*

In addition to importing metadata as XMI documents and using the interfaces generated from the metamodels, users might need to import or export metadata as PNML documents. This is important because users might uses tools which produce metadata written in PNML.

To import theses documents to the MOF repository, it was necessary to implement a piece of software to map PNML metadata to the corresponding MOF metamodel. This constitutes another way interchanging PNML metadata.

Figure 6 presents the architecture of the modules implemented to import/export PNML documents to/from the MOF repositories. It is important to notice that importing and exporting documents is already supported by MOF tools. However, to import and export PNML documents it was necessary to implement a set of methods to transform and object in the MOF repository in PNML documents and *vice-versa*. The users might also use Java programs to access the repository and manage Petri nets.

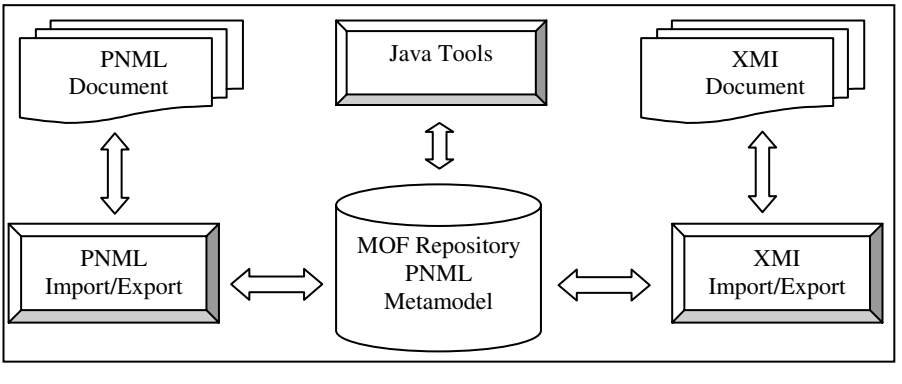


Fig. 6. Architecture of PNML and XMI documents Import/Export Modules

6.1 PNML Import

Importing a PNML document, mean the submitting a document to an XML parser which processes the document and writes it to the repository, as an instance of the PNML metamodel. The import model executes the following tasks:

1. Process the PNML document, checking whether it is a valid PNML document.
2. Start the MOF repository and search PNML repository, represented by the PNML metamodel. This means finding an instance of the *MOFPackage* class of the MOF metamodel named *PNMLMetamodel*.
3. Create an instance of *PNDocument* which will be the new document imported to the repository.
4. For each Petri net in the PNML document, create an instance of class *PetriNet*.
5. For each place, transition and arc of the Petri net, create an instance of the corresponding class in the metamodel. For example, for each place of the Petri net, create an instance of class *PNPlace*.
6. Create labels for each object created in the metamodel. These labels may be attributes or annotations. Labes presented in the screen of the tools are created as annotations; the others are created as attributes.
7. Create the arcs among objects using the associations of the PNML metamodel.

Report 5 – Subset of the Java Code used to generate instances of the PNML metamodel from PNML Documents

```
public static Pndocument
 pnml2mof(PnmlmetamodelPackage pkg,  Node node, String docname){
        Pndocument d=null;
        DocumentContains dc;
        Node childNode;
        if (node!=null){
            d = pkg.getPndocument().createPndocument(docname);
            dc = pkg.getDocumentContains();
            NodeList l = node.getChildNodes();
            for(int i=0; i<l.getLength();i++){
             childNode = l.item(i);
             if (childNode.getNodeName().equals("net")){
              PetriNet p =
                        pnml2mof(pkg, childNode);
             dc.add(p,d);
             }
            } }
         return d;}
```

Report 5 presents a subset of the Java Code used to implement the mapping. The parser used to process the documents was Xerces, using DOM (*Document Object Model*). This code shows the mapping of a PNML document to the MOF repository.

6.2 PNML Export

Exporting a PNML document means to generate the PNML document from the information stored on the MOF repository. To do that, it is necessary to start the MOF repository and search the PNML subrepository, i.e. the *PNMLMetamodel* package.

Then, access the instance of class *PNDocument* (inside the package) which represents the PNML document to be exported. This instance is a parameter to method mof2pnml of class *PNDocumentFactory*. This method will create the PNML document based on the object received.

Report 6 – Subset of Java Code to Search a PNML Document

```
1 - if (pkg !=null){
2 -          javax.jmi.reflect.RefClass refclass =
                                        pkg.refClass("PNDocument");
3 -          java.util.Collection c = refclass.refAllOfClass();
4 -          java.util.Iterator iter = c.iterator();
5 -          while(iter.hasNext()) {
6 -              Pndocument pndoc = (Pndocument)iter.next();
7 -             if (pndoc.getName().equals(docname)){
8 -                 Node node = mof2pnml(xmldoc,pndoc,pkg);
9 -                xmldoc.appendChild(node);
10-               savePNML(xmldoc); }
          }
 }
```

Report 6 presents a subset of the Java code used to search an object (instance of class *PNDocument*) in the MOF repository. Method *refClass* (line 2) of reflexive interface *RefPackage* returns the proxy interface of a specific class of the metamodel (passed as a parameter). In this case, the class is *PNDocument,* and the interface returned is *PNDocumentClass*. The method *refAllOfClass* returns a list of all instances (objects) of a given class of the metamodel, *PNDocument* in the example. The next task is to search (using the *name* attribute) the object in the list of objects and to pass it to method *mof2pnml* (line 8), which returns an XML *node* containing the PNML document.

6.3 Access by Java Tools

In addition to using the structure provided to import and export PNML and XMI documents, users can write client programs to directly connect to the MOF repository, obtain a package, which refers to a metamodel, and use it to build new metadata as instance of this metamodel.

Report 7 presents a subset of the Java code written to access the PNML repository, represented as package *PNMLMetamodel*. In line 1 of the Report 7, *MDRManager.getDefault().getDefaultRepository()* returns the standard repository. It is always de MOF metamodel. Then, it is necessary to search the repository to obtain the proxy package referring to the PNML metamodel. Method *repository.getExtent("pnmlmetamodel")* (line 2) looks for a specific proxy package inside the repository. It receives the name of the package as parameter and returns an object of type *RefPackage,* which is part of the reflexive package of JMI. If the search is successful, the package will be used to manage PNML metadata.

To create an object in the metamodel, it is necessary to obtain a reference to the proxy interface of the object to be created. For example, to create a new PNML document, it is necessary a reference to *PndocumentClass*, which is obtained by method *getPndocument()* of the *PnmlmetamodelPackage*. The *PndocumentClass* has

a method to create a new PNML document, instance of *Pndocument.* Likewise, to create an object as instance of any class of the metamodel, it is necessary to obtain the reference to its proxy interface.

Report 7 – Java Code to Create Petri Nets in the MOF Repository

```
1 - MDRepository repository =
                MDRManager.getDefault().getDefaultRepository();
2 - PnmlmetamodelPackage extent =
    (PnmlmetamodelPackage)repository.getExtent("pnmlmetamodel");
3 - if (extent !=null)   {
4 - repository.beginTrans(true);
5 - Pndocument d = pkg.getPndocument().createPndocument("Doc1");
6 - PetriNet p = pkg.getPetriNet().createPetriNet("n1",
                              "Example","timedNet");
7 - Pnplace p1 = pkg.getPnplace().createPnplace("p1");
8 - Pnplace p2 = pkg.getPnplace().createPnplace("p2");
9 - Pntransition t1 =
                pkg.getPntransition().createPntransition("t1");
10- Pnarc a1 = pkg.getPnarc().createPnarc("a1");
11- Pnarc a2 = pkg.getPnarc().createPnarc("a2");
12- pkg.getDocumentContains().add(p,d);//add PN to doc.
13- pkg.getPncontains().add(p1, p);//add p1 to PN
14- Pnannotation p1_a1 =
        pkg.getPnannotation().createPnannotation("marking","1");
15- Pnannotation t1_a1 =
          pkg.getPnannotation().createPnannotation("delay","5");
16- pkg.getSource().add(a1, p1);//add source of a1
17- pkg.getSource().add(a2, t1);//add source of a2
18- pkg.getTarget().add(a1, t1);//add target of a1
19- pkg.getTarget().add(a2, p2);//add source of a2
20- repository.endTrans();
}
```

Executing the program presented in Report 7 creates a PNML document with a timed Petri net [ZUBE1991]. This net has two places, one transition and two arcs. The code in Report 7 makes the associations among the several objects created. For instance, *PNContains* makes associations among every place, arc and transitions to an object, instance of class *PetriNet*. In addition, labels are created for every place, transition and arcs objects. Line 14 shows a piece of code to create a label to place *p1*.

Figure 8 presents the Petri net created by the execution of the code presented in Report 7.

This Petri net can be exported from repository as PNML or XMI document. Figure 9 presents the corresponding PNML document.

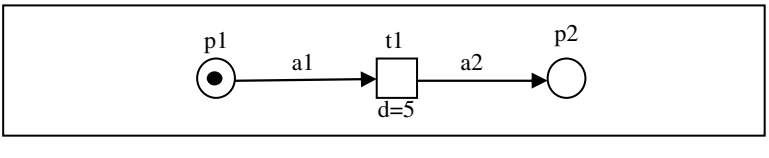


Fig. 8. Timed Petri Net Created

```
<?xml version="1.0" ?>
- <pnml>
  - <net id="n1" type="timedNet">
      <name>Example</name>
    + <place id="p1">
    + <place id="p2">
    + <transition id="t1">
    + <arc id="a1" source="p1" target="t1">
    + <arc id="a2" source="t1" target="p2">
    </net>
  </pnml>
```

Fig. 9. PNML Document Created from the Petri Net of Figure 8

7 Case Study

We implemented the interchange of Petri nets amongs the MDR and dMOF MOF Repositories as well as the Petri Net Kernel tool.

The net presented in Figure 10 was modelled in the Petri Net Kernel tool and saved as a PNML document. This document was then imported to the MDR repository. After that, this document was exported from MDR as an XMI document, which was then imported to the dMOF repository.

From the dMOF repository, this document could then be exported as a PNML document which was then read and processed by the Petri Net Kernel tool without any problems. Figure 10 shows the steps adopted in the case study.

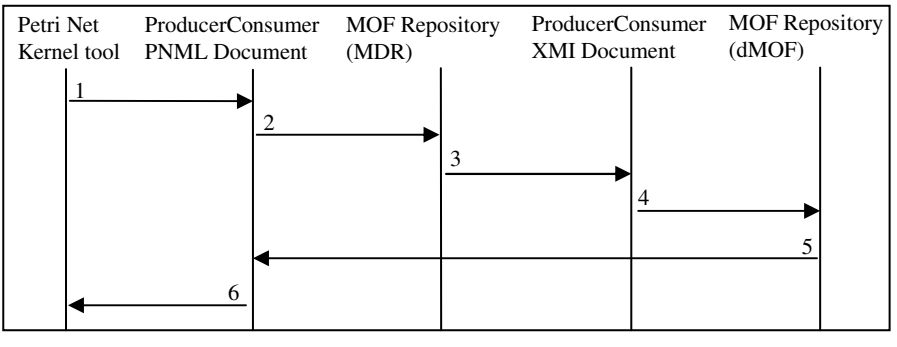


Fig. 10. Case Study Steps

8 Conclusion and Future Work

This paper described the design and implementation of a MOF metamodel based on PNML, which is a standard for storing and interchanging Petri nets. This extension makes it possible to use MOF tools to manage Petri nets.

As part of this work, an XMI document was created from proposed metamodel. Any MOF repository can import this document and, so, they can support management of Petri net metadata.

Besides the metamodel, Petri net metadata, instances of the metamodel, can also be imported and exported by any MOF repository as PNML documents. The implementation of these conversion facilities was also carried out as part of this work.

We have also presented a case study which shows how interfaces of the metamodel can be used to interchange metadata among different MOF repository, as well as Petri net tools.

A work that could follow from here is to add support to modules [JKW2000]. A module is a Petri net that has two kinds of objects: a) *internal objects*, which are part of the implementation and cannot be accessed by other nets and; b) *interface objects*, which can be accessed by other nets. Modules allow for a better reuse of Petri net components.

Another possible work regards the integration of this metamodel with the DTD Metamodel [SBF2003a]. This would make it possible to specify the Petri nets Schemas using a DTD.

References

[ABK2000] ANDERSON, R.; BIRBECK, M.; KAY, M.; et all. "Professional XML". Wrox Press Ltda. 2000.

[AHAY2001] AHMED, Kal; AYERS, Danny; et al. "Professional XML Metadata". UK. Wrox Press Ltda. 2001.

[BILL2003] BILLINGTON, Jonathan; et all. "The Petri Net Markup Language: Concepts, Technology, and Tools". Proc. 24th Int. Conf. Application and Theory of Petri Nets (ICATPN'2003), Eindhoven, The Netherlands, June 2003. Lecture Notes in Computer Science. Springer, 2003.

[BRBE2001] BRETON, Erwan; BÉZIVIN, Jean. "Towards an understanding of a model executability" In FOIS'01 – Formal Ontology in Information Systems, Ogunquit, Maine, USA October 2001

[DEMA2002] DEDIC, Svata; MATULA, Martin. "Metamodel for the Java language". In: http://java.netbeans.org/models/java/java-model.html. 2002.

[DMOF2001] "DMof – An OMG Meta Object Facility Implementation". In http://www.dstc.edu.au/Products/CORBA/MOF/. June, 2001.

[JKW2000] JUNGEL, M.; KINDLER, E.; WEBER, M. "The Petri Net Markup Language". In S. Philippi, editor, Proceedings of AWPN 2000 - 7thWorkshop Algorithmen und Werkzeuge für Petrinetze, pages 47–52. Research Report 7/2000, Institute for Computer Science, University of Koblenz, Germany, 2000.

[JMI2002] Java Metadata Interface, JSR-40 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_040_jolap.html. March, 2002.

[KIWE1999] KINDLER, E.; WEBER, M. "The Petri Net Kernel: An infrastructure for building Petri Net tools" In 20th International Conference on Application and Theory of Petri Nets. Petri Net Tool Presentations, Williamsburgs, USA, June, 1999.

[MAIN2000] MARCO, David; INMON, W. H. "Building and Managing the Metadata Repository". New York. John Wiley & Sons, Inc. 2000.

[MAR1985] MARSAN, A. M; GALBO, G; BOBBIO, A; CHIOLA, G; CONTE G; CUMANI, A. "On the Petri Nets with Stochastic Timing". Internation Workshop on Timed Petri Nets. IEEE press. Torino, Italy, 1985.

[MDR2002] Sun Microsystems. "Metadata Repository Home" http://mdr.netbeans.org/. 2002.

[MERL1976] MERLIN, P. M; FABER, D. J. "Recoverability of Communication Protocol Implications of Theoretical Study". IEEE Transaction Communication, vol COM-24, September, 1976.

[MOF1999] OMG Meta Object Facility Specification, Version 1.3. http://www.dstc.edu.au/Research/Projects/MOF/rtf/. http://www.omg.org/. September, 1999.

[MOL1981] MOLOY, M. K. "On the Integration of Delay and Throughput Measures in Distributed Processing Models". PhD thesis. UCLA, USA, 1981.

[PCTM2001] POOLE,John; CHANG,Dan; TOLBERT, Douglas; MELLOR, David. "Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration". New York. John Willey & Sons, Inc. 2001.

[PET1981] PETERSON, James L. "Petri Net Theory and the Modeling of Systems". Prentice-Hall, Englewood Cli s, NJ, USA. 1981.

[REIS1985] REISIG, W. "Petri Nets. An Introduction". Volume 4 of Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[RAMCHA] RAMCHANDANI. "Analysis of Asynchronous Concurrent Systems by Timed Petri Net". Techinical Report n 120, Laboratory of Computer Science, MIT, Cambridge, MA, USA.

[ROZ1998a] ROZENBERG, G; REISIG, W. "Informal Introduction to Petri Nets". Lecture Notes on Petri Nets I: Basic Models. Springer Verlag, 1998.

[ROZ1998b] ROZENBERG, G; ENGELFRIET, J. "Elementary Net Systems". Lecture Notes on Petri Nets I: Basic Models. Springer Verlag, 1998.

[SAN2003] SANTOS, Hélio. "A metadata solution based on MOF and XML". M.Sc. Dissertation - Centro de Informática/UFPE. March, 2003. In Portuguese.

[SBF2003a] SANTOS, Hélio; BARROS, Roberto; FONSECA, Décio. "A Proposal for Management of XML and DTD Metadata in MOF". Proc. 18th Brazilian Symposium on Database, Manaus, Brazil, October, 2003. In Portuguese.

[SBF2003b] SANTOS, Hélio; BARROS, Roberto; FONSECA, Decio. "A Proposal for Management of RDF and RDF Schema Metadata in MOF". Proc. Int. Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE'2003), Sicily, Italy, November, 2003. Lecture Notes in Computer Science. Springer, 2003.

[TANN2002] TANNENBAUM, Adrienne. "Metadata Solutions: Using Metamodels, Repositories, XML and Enterprise Portals to Generate Information on Demand". New York. Addison Wesley. 2002.

[UML2001] OMG Unified Modeling Language Specification, Version 1.4. http://cgi.omg.org/docs/formal/01-09-67.pdf. September, 2001.

[XMI2000] Object Management Group, XML Metadata Interchange Specification, Version 1.1, http://www.omg.org/. June, 2000.

[ZUBE1991] ZUBEREK, W. M. "Timed Petri Nets: Definitions, Properties and Applications". Microeletronic and Reliability, vol 31, n 4, 1991.

A Meta-ontological Architecture for Foundational Ontologies

Heinrich Herre and Frank Loebe

Research Group Ontologies in Medicine (Onto-Med),
Institute of Medical Informatics, Statistics, and Epidemiology (IMISE) and
Institute of Informatics (IfI), Leipzig University, Germany
{herre,loebe}@informatik.uni-leipzig.de
http://www.onto-med.de

Abstract. In this paper we present and discuss a meta-ontological architecture for ontologies which centers on *abstract core ontologies* (ACOs). An ACO is the most abstract part of a foundational ontology. It is useful for an ontologically founded description of ontologies themselves, therefore ACOs are lifted to the meta-level. We propose a three-layered meta-ontological architecture which distinguishes an object level comprising foundational, generic or domain-specific ontologies, a meta-level with abstract core ontologies, and a meta-meta-level employing *abstract top ontologies* for the formalization of the underlying levels. Moreover, two axiomatic fragments for ACOs are provided, one of which is applied to *formal concept lattices* [1]. This demonstrates the use of ACOs for the ontological foundation of representation formalisms and illustrates advantages in comparison to the usual direct formal reduction to set theory. Finally, related work with respect to the architecture is briefly discussed.

1 Introduction

There is a rapidly growing body of work on ontologies in information systems over the last 10 to 15 years, which has been boosted by the vision of the Semantic Web. Likewise, research in formal tools and techniques related to ontology development (or *ontological engineering*) is very active. By Grubers definition of ontologies as sharable conceptual specifications [2], their development is an issue closely related to the field of conceptual modeling.

From the very beginning, ontologies were distinguished according to their intended range of applicability. In particular, *foundational ontologies*[1] were considered to provide the most general kinds of entities as a basis for more specific ontologies. However, work on foundational ontologies has resulted in rather large and complex systems. This is problematic if foundational ontologies are to be applied to identify and express ontological commitments of representation formalisms.

[1] Also referred to as *top-level ontologies* in the literature; examples are: DOLCE [3], GFO [4], SUMO [5], and Seibt's [6], Sowa's [7] and West's [8] ontologies.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1398–1415, 2005.

Furthermore, like conceptual models ontologies face the problem of meta-modeling, i.e., the question of which basic vocabulary should be used to define and explain ontologies, especially their entities as well as constraints and logical interdependences among them. Of course, there are already efforts like [9], many of which are inspired by meta-modeling experience in conceptual modeling. Nevertheless, we intend to contribute to a clear and comprehensible meta-architecture for ontologies. This is largely due to the fact that most ontology languages are directly reduced to an underlying set-theoretical model in order to provide a formal semantics, which especially holds for logic-based formalisms like the Web Ontology Language (OWL, [10]).

The aim of this paper is to provide a clear analysis of meta-language aspects of ontologies and ontology languages in order to develop a comprehensible, yet comprehensive meta-ontological architecture. In particular, a clearly arranged core of entity types is to be extracted from current foundational ontologies, which will be called an *abstract core ontology* (ACO). On the one hand, ACOs are to be used as a meta-level for ontologies. On the other hand, they can be employed for the ontological foundation of representation formalisms in an easier way compared to rich foundational ontologies.

According to these aims, the organization of the paper is as follows. In section 2 we analyze the use of meta-languages in general as well as the role of set theory for modeling and modeling formalisms. On this basis our meta-ontological architecture is presented. Section 3 introduces the central notion of ACOs in detail. First, a number of entities and relations for ACOs are identified and informally described. Following an approach which admits variants of ACOs, two particular ACOs are presented as formal fragments. The first of these is applied to Formal Concept Analysis (FCA) [1] in section 4. The final section 5 concludes the paper with a summary, a brief comparison with related work, and future directions.

2 Meta-ontological Analysis and Architecture

For every formalism there is the need to explain the relationship between its syntax and semantics. In particular, this task comprises the determination of the ontological commitments of the formalism itself or in conjunction with its application to a certain task. Thus, we start with considerations of meta-languages.

2.1 Meta-languages

Let W be a world of objects. A formal language $\mathcal{L}$ whose expressions refer to the objects in W is called an *object-level language* for W. In order to specify and communicate the meaning of these expressions, a *meta-language* $\mathcal{M}$ for the pair $(\mathcal{L}, W)$ is required. That means, $\mathcal{M}$ is a language whose expressions refer to the items included in $\mathcal{L}$ or in both, $\mathcal{L}$ and W, but which also refer to relations between $\mathcal{L}$ and W. A formal language $\mathcal{L}$ has a semantics if there is a class Sem of objects and a relation $\mathsf{den}(x, y)$ relating expressions of $\mathcal{L}$ to the objects of Sem.

The denotation relation $\mathsf{den}(x, y)$ stipulates a connection between a symbol x and a semantic object y.

The first point to note is that the notions of symbol and denotation are at the heart of the transition from informal to formal languages, but equally relevant for the explanation of natural language semantics. These notions have been puzzling philosophers of logic and language ever since (cf. [11]–[14]), but herein we restrict to a simplified view.

According to this, one may assume a basic relation which associates the symbols of a language to the objects of the real world. For example, we may say that the phrase "the moon" denotes a certain real object in the sky. We take denotation as an interface between the informal and the formal treatment of an ontology. Therefore, the notions around denotation are used in informal explanations of meta-language aspects, but in the formal treatment of meta-levels these entities are not taken into account. Further, it is assumed that only natural language can be used as a meta-language for any kind of language, including itself. However, an infinite regress arises if every expression is to be defined within language. For instance, if one claims that the word "moon" denotes a certain real thing in the sky, then the phrase "a certain real thing in the sky" is another expression which requires a certain meaning. This regress can only be avoided if we assume that there is an original anchoring relation relating symbols to objects, which has to be assumed as a basic intuition without further specification in language, neither formal nor informal.

2.2 The Role of Set Theory

Set theory is a convenient mathematical tool to describe and model things and structures. One may speak about sets (collections) of things, about graphs, algebras, operations etc. Sets are abstract atemporal entities; for example, considering a set $\{a\}$ of an apple a does not change anything on that apple. Mathematical objects may be founded on set theory, and hence, in describing parts of the world we may construct a set-theoretical structure which is associated to these parts of the world and which *models* them. For describing the language of set theory we need an appropriate meta-language; to simplify the matter we assume that in any case natural language is available as a (non-formal) meta-language. As is well-known, the meta-account of set theory includes the notions of *sets*, *urelements*, and the *membership* relation.

Moreover, set theory is intimately tied to logical languages due to the fact that the commonly accepted approach of Tarski-style model-theoretic semantics (cf. [15]) is based on set-theoretical constructions. The relationship between such languages and their meta-theoretical treatment is well established. At present many ontology languages are logical languages with model-theoretic semantics (cf. [10, 16, 17, 18]). Therefore, a sufficiently rich fragment of set theory should provide an appropriate basis for a formal account of a meta-language, i.e., set theory then serves as a meta-meta-language for object-level ontologies.

On the other hand, we do not intend to restrict ourselves to set theory alone, of which distinct variants exist. Instead, a more generic route is taken in the next

section, generalizing the role of set theory as a meta-meta-language for ontologies to the notion of an abstract top ontology. This allows for other formalisms to be used as the meta-meta-account. For example, a minor deviation from set theory would be to consider variants of it, like hypersets [19]. But more radical choices are conceivable, for instance in favor of *mathematical category theory* [20]. In every case, any of these have to be chosen with great care as they lay the formal foundation for analyses of ontologies.

2.3 Meta-ontological Architecture

In our work on the General Formal Ontology (GFO, cf. [4]), which is a foundational ontology, the need for a meta-ontological level arose, which is not already a set-theoretical reduction. In striving towards such an approach we emphasize generality and comprehensibility as requirements. Note that herein we focus on a basic vocabulary for entities within ontologies, whereas ontologies as a whole or complex components of ontologies are not discussed.

In figure 1 a three-layered meta-ontological architecture is proposed. As introduced in section 2.1, natural languages form a universally applicable, but informal approach to every level. The three layers on the right-hand side of figure 1 provide a well-founded formal approach.

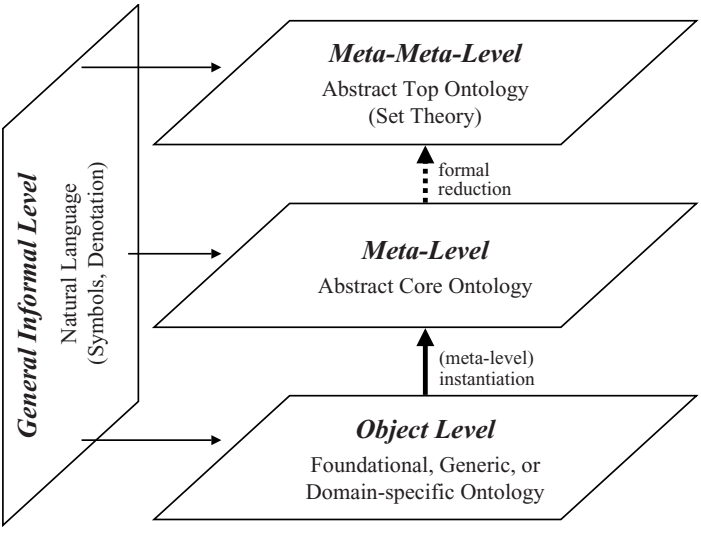


Fig. 1. Meta-ontological Architecture

The lowest level refers to the objects of the meta-architecture, viz. to ontologies with varying degrees of their range of applicability. The meta-level provides the notions to work with object-level ontologies. In section 3.1 the most general basic entities are specified which can be included in the meta-level. Based

on these notions several abstract core ontologies may be constituted, which are determined by a selection of basic notions and by axioms stated about them. Hence we admit different variants of ACOs, two of which are presented in sections 3.2 and 3.3. In this respect our work is inspired by conceptual modeling, cognitive linguistics, philosophy, and by our work on the foundational ontology GFO, expounded in [4].

In order to be able to treat ACOs in a formal manner, there is a need for a formal meta-account of them. This leads to the introduction of the notion of an *abstract top ontology* (ATO), which generalizes the role of set theory as described in section 2.2.

Often set theory itself is viewed to take the role of an ACO instead of an ATO. However, our experience shows that in many cases representation languages are ontologically richer than set theory (cf. [21]). Hence the introduction of an intermediate layer appears reasonable. Looking at ACOs from the opposite side, namely from the object level, formerly the notion of foundational ontologies was considered to fill this role already. However, the growing body of work on foundational ontologies shows that these are too rich already to integrate all their features into formalisms directly. Thus, we introduce the distinction between ACOs and foundational ontologies.

Some remarks on the *relationships* between object, meta- and meta-meta-level need to be made. Frequently, the relations between these levels are considered to be of the same character, often a vague notion of instantiation (cf. section 5.2). This is not the case for the levels above. From our point of view, the ontological notion of instantiation is preserved only between the object and the meta-level, since they provide ontological content. However, the transition from the ACO level to the ATO level is different. For example, in the case of set theory as an ATO one may speak of a set-theoretical reduction of the entities on the meta-level. Again, the mere intention behind the ATO level is to provide a rigorous mathematical framework in order to study formal properties of ontologies on the lower levels.

These issues may become clearer by example. In section 3.1 several entities and relations will be introduced, among them the notion of *category* and (object-level) *instantiation*. These belong to the meta-level, i.e., they are ontological entities whose interconnections to other entity types need to be clarified. Later then, when it comes to formalizing these interconnections, these notions are expressed in the vocabulary of the meta-meta-level. More precisely, category will be modeled as a set Cat whose elements are set-theoretical urelements, whereas instantiation is modeled as a set denoted by $::$ with pairs of urelements as its elements. In contrast to this reduction on the meta-meta-level, the relation between meta- and object level is different. Categories and instantiation are not merely "implementation devices" for ontologies. In fact, it is debatable whether some or even all of the meta-entities should be reflected on the object level. However, in contrast to a fully self-reflective ontology and for the sake of clarity and simplicity, the separation into meta- and object level is preferred herein (cf. also [22]).

3 Abstract Core Ontologies (ACOs)

This section elaborates the meta-level of the architecture which is formed by abstract core ontologies. Let us first introduce and discuss the pragmatic and functional aspects of ACOs. An abstract core ontology captures such entities which are built – implicitly or explicitly – into the semantics of a wide number of general representation formalisms and which commonly occur in foundational ontologies. Examples for foundational ontologies are given in section 1, hence it remains to clarify the term "general representation formalism". It refers to such formalisms which are not adapted to particular domains of discourse, but which can be applied to a broad scope of modeling and representation problems. Examples range from first-order logic over Semantic Web languages like the Web Ontology Language (OWL, [10]), frame-based approaches as in [23], to modeling languages like UML [24].

Apart from pragmatic aspects, ACOs need to be determined firstly by their main entity types and relations among them, for which a certain vocabulary is proposed below. Secondly, logical interdependences of those entities and relations need to be specified. After an informal description of these connections formal accounts are elaborated in sections 3.2 and 3.3. The latter exemplify the formalization of several types of interdependences by axioms of first-order logic. Note that this approach relies on the meta-meta level, where in our case set theory is employed as the abstract top ontology.

3.1 Entities for ACOs

In identifying entities for ACOs we pursue a pragmatic approach which is further based on philosophical considerations, in particular on the work of J. Gracia [25, 26]. We will not insist on a single ACO, but we specify a set of entities (cf. table 1), of which certain subsets may suffice for a particular ACO.

Before we go into details of this vocabulary, let us once more mention the strict separation between the levels introduced in section 2.3. We stipulate that none of the meta-level relations are available as objects on the object level. This requirement implies that infinite regresses like Bradley's infinite regress of instantiation (cf. [27]) can be avoided. The same requirement is assumed for most of the meta-level entities, although reflections of them on the object level may be less complicated. For example, one could include the meta-level entity "Individual" also as an entity of the object level.

Our starting point is that the entities of the world may be divided into *categories* and *individuals*, i.e., everything in an ontology is either a category or an individual. Categories are entities which may be predicated of or *instantiated by* other entities; in the opposite, individuals essentially cannot be instantiated or predicated of other entities [26]. Note, however, that there are also categories which cannot have instances, e.g. due to their logical structure, like a "round square". Nevertheless, such categories are distinct from individuals. The most natural individuals are related to time and space, whereas categories are atemporal, abstract entities.

Table 1. Basic Entity Types and Relations

Meta-Level Entity Types (Sets of urelements)			
Name	*Symbol*	*Name*	*Symbol*
Category	Cat	Individual	Ind
Object Category	OCat	Object	Obj
Property	P	Attribute	Att
Role Category	RCat	Role	Rol
Relation	R	Relator	Rel

Meta-Level Relations (Sets of pairs of urelements)		
Name	*Symbol*	*Argument Restrictions*
identity	$x = y$	–
instantiation	$x :: y$	$\mathsf{Cat}(y)$
inherence	$\mathsf{inh}(x, y)$	$\mathsf{Att}(x)$ or $\mathsf{Rol}(x)$
role-of	$\mathsf{role}(x, y)$	$\mathsf{Rol}(x)$, $\mathsf{Rel}(y)$
categorial part-of	$\mathsf{catp}(x, y)$	$\mathsf{Cat}(x)$, $\mathsf{Cat}(y)$

Moreover, among individuals we distinguish *objects*, *attributes*, *roles* and *relators*. Objects are "complex" entities which have attributes and which play certain roles in respect to other entities. Objects are to be understood as similarly general as the notion of "object" in object-oriented analysis. In particular, objects comprise animate and inanimate things like humans, trees or cars as well as processes like this morning's sunrise[2]. Examples of attributes are particular weights, forms, colors, etc. A sentence like "This rose is red." refers to a particular object which is a rose, and to a particular attribute which is a red. Another basic relation is needed to connect objects and attributes. The phrases "having attributes" and "playing a role" above are captured by the basic relation of *inherence*, which is such that an attribute or a role inheres in some object. This relation expresses the dependence of attributes and roles on entities in which they can inhere.[3]

The difference between attributes and roles consists in the fact that roles are interdependent with other roles [28]. Examples of roles are available through terms like parent, child or neighbor. Here, parent and child would be considered as a pair of interdependent roles. Apparently, these examples easily remind one of relations like "is-child-of". Indeed, a composition of interdependent roles is a *relator*, i.e., an entity which connects several other entities. The formation of relators from roles further involves the basic relation of *role-of*.

Thus far we have explained the right-hand side of the upper part of table 1, i.e., the subdivision of individuals and the interrelations among them. However, in explaining roles by reference to terms like parent and child, there was already

[2] More fine-grained distinctions among objects do not belong to the ACO level, but appear as object-level instances of ACO notions, e.g. within some foundational ontology.

[3] This commonality allows one to generalize attributes and roles to *qualities*. Note that this notion of quality differs from the term as used in [4].

a transition to the realm of categories. Actually, parent and child refer to *role categories*. As is obvious from the table, for all specializations of individuals introduced above there are categorial counterparts.

Two basic relations have not been discussed thus far: *identity* and *categorial part-of*. We will not dwell on the intricacies of identity, only to mention that we refer to it as a notion in the interface between natural and formal languages. We expect a full account of identity to involve denotation, possibly along the lines of the Peircean notion of co-reference (cf. [7]). Categorial part-of is more in the focus of the present work. Its arguments are categories; it directly reflects dependencies among categories and uncovers how one category may be constructed out of others. For example, the property of being round may be a categorial part of the category of round glasses. Another example is the category apple, of which one may consider categorial parts like a certain form or color and the like. There are two intricate issues regarding categorial part-of. Firstly, a categorial part must not be misinterpreted to apply to a category directly. For example, a category with the property red as categorial part is not red itself. This problem is easily avoided in the formalization. Secondly, it should be noted that inherence and categorial part-of are irreducible with respect to each other. Section 4 will provide further applications of categorial part-of.

3.2 Categories, Properties, and Objects

In this section we consider an abstract core ontology, denoted by CPO, which is based on categories and properties only, together with their individual counterparts and the corresponding meta-level relations. Object-level relations and role categories as well as relators and roles are not included. Formally this corresponds to the following signature:

$$\Sigma^{CPO} = (\mathsf{Cat}, \mathsf{OCat}, \mathsf{P}, \mathsf{Ind}, \mathsf{Obj}, \mathsf{Att}, =, ::, \mathsf{inh}, \mathsf{catp}) \tag{1}$$

Several languages can be used to formalize such systems. In the following, a type-free first-order language is assumed, although others may be appropriate as well (e.g. typed languages). We present axiomatic fragments pertaining to the signature introduced above.

First of all, implicit assumptions need to be explicated, which are often taken for granted by human readers on the basis of natural language descriptions, like the one given in section 3.1. First, there are disjointness and coverage conditions for entities. Concerning (4), remember that CPO omits relators and roles.

$$\neg\exists x \ \ (\mathsf{Cat}(x) \wedge \mathsf{Ind}(x)) \tag{2}$$

$$\forall x \ \ (\mathsf{Cat}(x) \vee \mathsf{Ind}(x)) \tag{3}$$

$$\forall x \ \ (\mathsf{Ind}(x) \rightarrow \mathsf{Obj}(x) \vee \mathsf{Att}(x)) \tag{4}$$

Meta-level subsumption relations need to be expressed, which are indicated in table 1 by indentation.

$$\forall x \ \ (\mathsf{Obj}(x) \rightarrow \mathsf{Ind}(x)) \tag{5}$$

$$\forall x \ \ (\mathsf{P}(x) \rightarrow \mathsf{Cat}(x)) \tag{6}$$

Regarding the meta-level relations, similar axioms have to be stated (7). In addition, an account of the argument restrictions from table 1 is needed (8). In an opposite manner, claims of existence can be expressed (9).

$$\neg(\exists xy \;\; (x :: y \wedge \mathsf{inh}(x, y))) \tag{7}$$
$$\forall xy \;\; (\mathsf{inh}(x, y) \rightarrow \mathsf{Att}(x) \wedge \mathsf{Obj}(y)) \tag{8}$$
$$\forall x \;\; (\mathsf{Obj}(x) \rightarrow \exists y(\mathsf{Att}(y) \wedge \mathsf{inh}(y, x))) \tag{9}$$

Moreover, definitions may reveal "reducible" entities or new, definable notions which are also interesting from an ontological point of view. In the case of CPO, categories can be used to define their individual counterparts (10-11), but not vice versa due to the possibility of categories without instances.

$$\forall x(\mathsf{Ind}(x) \leftrightarrow \neg\mathsf{Cat}(x)) \tag{10}$$
$$\forall x(\mathsf{Obj}(x) \leftrightarrow \exists y(\mathsf{OCat}(y) \wedge x :: y)) \tag{11}$$
$$\forall x(\mathsf{Att}(x) \leftrightarrow \exists y(\mathsf{P}(y) \wedge x :: y)) \tag{12}$$
$$\forall x(\mathsf{PrimCat}(x) \leftrightarrow \mathsf{Cat}(x) \wedge \exists y(y :: x) \wedge \forall z(z :: x \rightarrow \mathsf{Ind}(z))) \tag{13}$$
$$\forall x(\mathsf{AttCat}(x) \leftrightarrow \mathsf{Cat}(x) \wedge \exists y(y :: x) \wedge \forall z(\mathsf{catp}(z, x) \rightarrow \mathsf{P}(z))) \tag{14}$$

Note that (10) already follows from (3) and (2). The newly defined notions of a *primitive category* (13) and an *attributive category* (14) are relevant for section 4. They reveal branching points for the ACO. For instance, the question arises whether one should assume that all categories are primitive. On the other hand, categories of higher types may be admitted. Similarly, attributive categories allow for new constraints: the equality of attributive categories may be defined (15), and, as we decided for CPO, attributive categories may be the only admissable categories (16). From the latter follows that catp is constrained to properties in its first argument.

$$\forall xy \;\; (\mathsf{AttCat}(x) \wedge \mathsf{AttCat}(y) \rightarrow (x = y \leftrightarrow \forall z(\mathsf{catp}(z, x) \leftrightarrow \mathsf{catp}(z, y)))) \tag{15}$$
$$\forall x \;\; (\mathsf{Cat}(x) \rightarrow \mathsf{AttCat}(x)) \tag{16}$$

3.3 Categories, Relations, and Objects

The second abstract core ontology herein is denoted by CRO, and it comprises all notions which were introduced in table 1, in particular roles and object-level relations. This yields the following signature:

$$\Sigma^{CRO} = (\mathsf{Cat}, \mathsf{OCat}, \mathsf{P}, \mathsf{RCat}, \mathsf{R}, \mathsf{Ind}, \mathsf{Obj}, \mathsf{Att}, \mathsf{Rol}, \mathsf{Rel}, =, ::, \mathsf{inh}, \mathsf{role}, \mathsf{catp}) \tag{17}$$

To make our approach to relations clear, we insert an example that illustrates the formalization of an object-level relation. On the object level, let John be a parent of Mary. According to the framework described, John and Mary are objects, and there is a relator which instantiates the parent-of relationship and which connects John and Mary. This connection is established via two roles, one

of which being a role of John, the other being a role of Mary. Let j, m, po, p, c be constants for John, Mary, the parent-of relationship, and the parent and the child role categories, respectively. Then, the following formalizes the example:

$$\mathsf{Obj}(j) \wedge \mathsf{Obj}(m) \wedge \mathsf{R}(po) \wedge \mathsf{RCat}(p) \wedge \mathsf{RCat}(c) \tag{18}$$

$$\exists q_1 q_2 r(r :: po \wedge q_1 :: p \wedge q_2 :: c \wedge$$
$$\mathsf{inh}(q_1, j) \wedge \mathsf{inh}(q_2, m) \wedge \mathsf{role}(q_1, r) \wedge \mathsf{role}(q_2, r)) \tag{19}$$

Of course, similar axioms as those presented for CPO belong to the axiomatization of CRO, where not all of those specified for CPO can be reused directly. For instance, (4) and (8) need to be modified to include roles. We will not specify these modifications as they are easily found. Rather, we introduce some further axioms pointing to advantages and problems with this role-based account of relations. A discussion of these issues beyond that is available in [28], including an analysis of the advantages of this approach in comparison to viewing relations as having tuples of entities as instances.

$$\forall x \ \ (\mathsf{Rel}(x) \rightarrow \exists yz(y \neq z \wedge \mathsf{Rol}(y) \wedge \mathsf{Rol}(z) \wedge \mathsf{role}(y, x) \wedge \mathsf{role}(z, x))) \tag{20}$$
$$\forall xyz \ \ (\mathsf{inh}(x, y) \wedge \mathsf{inh}(x, z) \rightarrow y = z) \tag{21}$$

Axiom (20) enforces that relators are really mediating entities, i.e., more than one role is involved, which is even the case if an object is related to itself. (21) is known for attributes as the *non-migration principle*. However, note that in CRO it also refers to roles which can inhere in a single entity only. This is not to be confused with an entity in which several roles of the same role category inhere. For example, John can be a parent of Mary as well as of Fred. Another question concerns the arguments of relations, i.e., whether one should restrict the second argument of inherence. For the sake of simplicity one would often restrict it to objects (22).

$$\forall xy(\mathsf{inh}(x, y) \rightarrow \mathsf{Obj}(y)) \tag{22}$$

However, in a more expressive system, relators between other entity types may be required. In that case, it would be a common approach to classify relators according to the types of entities their roles inhere in. An equivalent classification could be done for the corresponding relations. Before we can give an example, we need to enforce that every relation is bound to certain role categories (23-24), (cf. also [28], sect. 3.3.3). Then, for example the parent-of relation can be claimed to satisfy (26).

$$\forall xy \ \ (\mathsf{R}(y) \wedge \mathsf{catp}(x, y) \rightarrow \mathsf{RCat}(x)) \tag{23}$$
$$\forall x \ \ (\mathsf{R}(x) \rightarrow \exists y(\mathsf{catp}(y, x))) \tag{24}$$
$$\forall x \ \ (\mathsf{ObjRCat}(x) \leftrightarrow \forall yz(y :: x \wedge \mathsf{inh}(y, z) \rightarrow \mathsf{Obj}(z))) \tag{25}$$
$$\forall xy \ \ (\mathsf{ObjR}(x) \leftrightarrow \forall y(\mathsf{catp}(y, x) \rightarrow \mathsf{ObjRCat}(y))) \tag{26}$$

The introduction of axiomatic fragments for two ACOs, CPO and CRO, shows that there are many branching points even for these fairly small ontologies. In connection with a formal account of ACOs, these subtleties need to be explicated and related to each other. This is one of the reasons why we expect that it is more appropriate to start with a restricted number of primitives on the ACO level of our meta-ontological architecture.

Apart from the use of ACOs as meta-ontology for object-level ontologies, their application to general representation formalisms has been mentioned. In the next section we show by example how CPO is employed for the ontological foundation of one such formalism, namely *formal concept lattices*. A stronger evaluation of the work presented here in quantifiable terms remains as future work.

4 Formal Concept Lattices

Formal Concept Analysis (FCA, [1]) is a mathematical formalism with a central notion of *concept lattices* which can be used for conceptual data analysis and knowledge processing.[4] It has gained wide spread from its earliest beginnings in the late 1970s. We show that the theory of concept lattices can be interpreted in our framework of abstract core ontologies, thus giving them an ontological foundation. Moreover, concept lattices can be applied in diverse ways which can be clarified by explicit reference to ACOs. Accordingly, we believe that each use of formal concept analysis should be coupled with an appropriate ontological analysis in order to gain clarity and to use the results of these formal techniques adequately.

The following are the basic definitions of FCA [1] to which we refer herein.

Definition (*Formal Context and Concept*)
A formal context $\mathcal{K} = (G, M, I)$ consists of two sets G and M and a relation I between G and M. The elements of G are called the *objects*, and elements of M are called the *attributes* of the context. $I(g, m)$ means that the object g has the attribute m.
For a set $A \subseteq G$ of objects and a set $B \subseteq M$ of attributes, the set of attributes common to A is $Attr(A)$, the set of objects with all attributes in B is $Objt(B)$:

$$Attr(A) = \{m \mid m \in M \text{ and } I(g, m) \text{ for all } g \in A\} \tag{27}$$

$$Objt(B) = \{g \mid g \in G \text{ and } I(g, m) \text{ for all } m \in B\} \tag{28}$$

A *formal concept* of $\mathcal{K}$ is a pair (C, D) with $C \subseteq G$, $D \subseteq M$, and $C = Objt(D)$, $D = Attr(C)$. C is called the *extent* and D the *intent* of the concept (C, D).

This definition is an example for the specification of a formalism by direct reference to the meta-meta-level of our approach, where set theory serves as abstract top ontology in this case. However, the use of terms like attribute, object,

[4] See also `http://www.math.tu-dresden.de/~ganter/fba.html`

intent and extent exemplifies hidden ontological assumptions. As examples in [1] indicate, there are several possibilities of interpreting the formalism on the ACO level. That means, it can be used with different ACO level readings which we will now analyze. In the sequel it is assumed that CPO is sufficiently expressive for every of these interpretations. The adequacy of this assumption is reconsidered at the end of this section.

In the first interpretation, G is a set of objects (in the sense of section 3.1, i.e., understood as individuals with attributes), M is a set of properties (categories whose instances are attributes). Then the relation I is interpreted by a relation I_1 defined in terms of inherence and instantiation as follows:

$$I_1(g, m) \leftrightarrow \exists x(x :: m \wedge \mathsf{inh}(x, g)) \tag{29}$$

However, in most examples in the applications of concept lattices the objects are not individuals, but themselves categories. For instance, one example states "A frog needs water to live" for an object "frog" and an attribute "need water to live". Then frog is an object category whose instances are individual frogs, where every individual frog needs water to live. "Needs water to live" has to be understood as a property in the terms of CPO. Hence, in the second interpretation of I, denoted by I_2, the expression "A frog needs water to live" reads as "For every instance of the 'category frog' ($g \in G$) there is an instance of the property 'to need water to live' ($m \in M$) which inheres in that individual frog." This ontologically correct reading of that sentence can be formalized as:

$$I_2(g, m) \leftrightarrow \forall y(y :: g \rightarrow \exists x(x :: m \wedge \mathsf{inh}(x, y))) \tag{30}$$

A third interpretation of I is given by I_3, which in the example corresponds to "the category frog has as a categorial part the property 'to need water to live'":

$$I_3(g, m) \leftrightarrow \mathsf{catp}(m, g) \tag{31}$$

Apparently, it is not the category frog which needs water to live but the individual frogs. Thus, the right-hand side of (31) may entail the right-hand side of (30). In order to achieve this, a suitable ACO should include to following axiom:

$$\forall xy(\mathsf{catp}(x, y) \rightarrow \forall u(u :: y \rightarrow \exists w(w :: x \wedge \mathsf{inh}(w, u)))) \tag{32}$$

Moreover, if the third reading (I_3) is assumed, one may claim that for concept lattices all expressable categories are attributive categories, i.e., axiom (16) of CPO applies. Altogether, CPO provides at least three exact and ontologically adequate interpretations of the relation I defined for concept lattices.

The analysis continues with the question for the nature of the notion of a formal concept (C, D), which depends on the relation between formal objects and attributes. Assuming the third reading (I_3) from above, the extent C of (C, D) is a set of attributive categories, whereas the intent D is a set of properties. More precisely and following the definition of a formal concept, every property in D is a categorial part of each category in C. Ontologically, it seems that (C, D) is

best understood as a category $\hat{c}$ with all properties in D as its categorial parts and *subsuming* all categories in C:

$$x \in C \rightarrow \mathsf{AttCat}(x) \wedge \forall y(y :: x \rightarrow y :: \hat{c}) \tag{33}$$

$$x \in D \leftrightarrow \mathsf{P}(x) \wedge \mathsf{catp}(x, \hat{c}) \tag{34}$$

Notice here that the membership relation between the set C and its elements requires an ontological interpretation which differs from the one required for D and its elements, and either differs from the naive view of membership. The latter would assume instantiation as its ontological counterpart ($\forall xy(x \in y \leftrightarrow x :: y)$), but then (C, D) would refer to a higher-order category. Instead, we propose to understand the set C as reflecting a category $\hat{c}$ which subsumes C's elements without C being exhaustive regarding *all* subcategories of $\hat{c}$ (hence the implication in (33)). D may either be viewed as a property $\tilde{d}$ which subsumes D's elements, or as an object category $\hat{d}$ determined by D's elements as categorial parts. Looking at formal concepts, the way in which these are ordered to form concept lattices seems to suggest the latter interpretation ($\hat{d}$). In this case, the duality of formal concepts by extent and intent can be explained by the requirement that $\hat{c}$ and $\hat{d}$ are two categories with the same extension. Furthermore, this explains the Duality Principle for Concept Lattices [1, p. 22] which states that an exchange of formal objects and attributes induces the dual concept lattice. In the ACO reading this could mean that object categories may be categorial parts of properties, or that a reinterpretation of formal objects and attributes as one type of category is needed.

Discontinuing the analysis, we need to return to the initial assumption that CPO is sufficiently expressive for every ontological interpretation of formal concept lattices. Even considering the examples in [1], this does not appear to be adequate for all of these. For instance, one example introduces countries as formal objects and organizations of countries as formal attributes. Another deals with business services as formal objects and types of business machines to which such services apply as formal attributes. Here, we admit that the ontological interpretations can be much more specialized. Moreover, the assignment to object categories and properties alone may not be suitable for all cases. Nevertheless, analyses like those above should be worthwhile for at least two reasons. Firstly, abstract core ontologies are more widely applicable, and one may refer to an existing ontological scheme when using a formalism, such as one of those above. Secondly, in the future one may expect that the small size and the applicability of abstract core ontologies leads to more extensively formalized and implemented versions than those which are available for very specialized domains. Then ACOs may be used in the very best sense of an ontology: to contribute to the interoperability of distinct formalisms.

Finally, one may ask whether an understanding of such precision is needed in applications. In our opinion, hidden imprecisions or ambiguities may cause problems or unexpected "behavior" of a formalism. For instance, the consequences of mixing the three initial interpretations I_1 (29), I_2 (30), and I_3 (31) from above remain unclear if they are not analyzed in terms of the underlying ACOs.

5 Conclusion

5.1 Summary and Discussion

Based on an analysis of meta-language issues we have provided a three-layered meta-ontological architecture which can be supported on every level by means of natural language. The architecture centers on the notion of an *abstract core ontology* (ACO), which has been discussed in detail, comprising a selection of entities and including some axiomatic fragments for two ACO variants. Roughly speaking, the CPO variant is based on objects and attributes only, whereas the CRO ontology in addition includes relations.

The last part of the paper uses CPO for the ontological foundation of basic notions of Formal Concept Analysis [1] and thus demonstrates an application of the framework apart from meta-ontological considerations. In particular, due to their limited number of selected categories ACOs are well suited for the ontological foundation of general representation formalisms, whereas rich foundational ontologies may provide too many or too special distinctions for this task. Nevertheless, the main application of the presented framework is to provide a meta-account for ontologies, including foundational ontologies.

Three important aspects of our proposal should be emphasized. Firstly, our architecture provides a clear border between ontological and formal contents. This border is situated in the transition from the meta-level to the meta-meta-level, due to which the relations between the three levels are different. In many approaches this distinction is not made, leaving it unspecified to which extent ontology plays a role on each level.

Secondly, the overall architecture is laid out with a strong interest in clarity and, in some sense, minimality in the number of notions. It is our hope that this will be beneficial in two respects. On the one hand, conceptual transparency may promote the use of the architecture in its intended areas of application. On the other hand, this shall allow for an extensive formal treatment, where experience suggests that many overlooked subtleties appear in the process of the formalization of natural language specifications. For the latter reason the formalization itself is considered valuable, as yet independent of computational applications or automated reasoning with ACOs as a primary concern.

Thirdly, during the formalization various branching points become explicit which give rise to different ACOs (with differences beyond their underlying entity types). Thus far it is not intended to promote exactly one of these ACOs. Instead, for distinct cases we expect several ACO variants to be appropriate, which can be formally related within the architecture, however.

5.2 Related Work

Two meta-level architectures shall be compared with ours. We start with the *SUO Information Flow Framework* (SUO IFF; formerly known as IFF Founda-

tion Ontology) which appears closest in its motivation, viz. the provision of a meta-architecture for the Standard Upper Ontology (SUO) initiative:[5]

> The IFF Foundation Ontology represents metalogic. It provides a principled foundation for the metalevel (structural level) of the Standard Upper Ontology (SUO). The SUO metalevel can be used as a logical framework for manipulating collections of object level ontologies.

This quote from [29] already shows some rather strong differences between the SUO IFF and our work. First, SUO IFF pursues a meta-logical approach rather than a meta-ontological. The objects it refers to and which are manipulated on the meta-level are theories and logics, rather than single categories.[6]

SUO IFF offers a sophisticated structure which is divided into three interior levels, called lower, upper and top metalevel. It integrates a large number of mathematical concepts and appears to be based on category theory. Though this may be a powerful approach, it is fairly hard to comprehend. Overall, the SUO IFF is rather a deeply elaborated ATO than an ACO. We are not aware of a level between the SUO IFF and the proposals for SUO itself, i.e., there is no ACO level in the SUO framework.

The second meta-modeling approach to be discussed is the *Meta Object Facility* (MOF, [31]) which provides the meta-modeling architecture for UML [24]. The MOF meta-modeling architecture is derived from the "classical framework for metamodeling" [31, p. 2-2], based on four layers:

1. The *information layer* comprises data which are to be described.
2. The *model layer* describes the data which may occur in the information layer in terms of meta-data, i.e., it defines a language for some information domain. Specific UML models belong to this layer.
3. The *meta-model layer* provides descriptions of meta-data, i.e., the structure and the semantics of meta-data on the model layer. The specification of UML itself belongs to this layer.
4. The *meta-meta-model layer* describes meta-models and relates to the meta-model layer just as the meta-model layer relates to the model layer. This layer is commonly used to close by convention the regress of meta-levels.

Assuming these layers[7], the MOF approach appears fairly similar to ours, at least from an architectural point of view. In this connection it is relevant that the lowest level of our approach is concerned with ontologies, which comprise entities of both the model and the information layer of MOF. This is due to the

[5] Cf. `http://suo.ieee.org/IFF/` and `http://suo.ieee.org`, respectively.

[6] With this focus, it reminds one on other works using category theory for relating different logics, in particular the institutions of Goguen [30].

[7] Note that in MOF these layers are considered as conventions for understanding rather than being of any definitive character, i.e., the interpretation of the MOF architecture (e.g. the number of levels) may vary in dependence of the environment in which it is applied.

understanding of ontologies as specifications of conceptualizations [2], and it is also apparent from works discussing UML and ontology languages (cf. [32] and section 6 of [9]). Accordingly, the ACO level corresponds to the MOF meta-model layer, whereas the ATO level matches the MOF meta-meta-model layer.

However, there are also a number of differences compared to our approach. First, the relationships between the levels are different in the two approaches. In particular, in MOF the relations between the levels appear to be equal, namely some form of "instantiation", in contrast to the relationships between the levels as described in section 2.3. Secondly, differences originate from the object-oriented approach taken in MOF, which assumes objects with object identity, state and behavior. As far as we are aware of ontologies, in particular the dynamic aspects in the sense of software systems are not yet integrated in ontologies. A third source of differences are the details with respect to static aspects, i.e., entities and their characteristics in the formalism. For instance, MOF only allows for binary relations and has a built-in inheritance mechanism for MOF classes, whereas we have a more flexible approach regarding relations (sect. 3.1), but no inheritance. Moreover, there is a greater number of MOF entities (cf. [31, p. 3-12]), but a less formal approach is taken as regards logical formalizations. A detailed technical comparison with MOF remains to be elaborated elsewhere.

In summary, there appear to be more commonalities between our architecture and MOF than compared to the SUO IFF. Nevertheless, a closer analysis reveals rather strong differences to either of the approaches discussed, some of which seem to be justified regarding differences in scope and historical development.

5.3 Future Work

The paper concludes with indicating various ways of extending the present work. From a theoretical and philosophical point of view, further analysis of the interface between the informal and the formal levels of the architecture is needed, for which one may draw on the existing body of philosophical literature. Further, the comparison to other meta-modeling approaches as started in section 5.2 should be extended. Within the ACO level, the axiomatic fragments are to be enlarged and put into relation to each other, for which purpose the SUO IFF [33] may be used.

A major field of application which is not discussed herein are ontology languages for the Semantic Web, primarily RDF and OWL [16, 10]. For these languages one would first choose an appropriate set of ACO entities, where CRO seems to provide sufficient expressiveness[8]. However, concerning the axiomatization modifications are expected. In this connection it will be crucial to find an interpretation of the feature of RDF to allow for a self-application of RDF properties. Another important issue is the integration of datatypes. From an ontological point of view one may assume that datatypes do not require further entity types in an ACO. Rather, their treatment is expected to involve the

[8] Possibly one could even omit attributes.

denotation relation, possibly within the meta-level. Besides interpreting single features of RDF and OWL, different forms of their usage may demand different ACO interpretations. The reification of RDF properties to overcome the lack of expressiveness of these languages with respect to n-ary relations[9] is a good example in this respect.

Finally, machine-processable versions of ACOs need to be implemented. First-order theorem provers may be used to support formal investigations of established ACOs, whereas implementations, e.g. in OWL, would allow for a direct use in connection with ontologies in the Semantic Web.

References

1. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin (1999)
2. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5** (1993) 199–220
3. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: Wonderweb Deliverable D18: Ontology library (final). Technical report, Laboratory for Applied Ontology – ISTC-CNR, Trento (2003)
4. Heller, B., Herre, H.: Ontological categories in GOL. Axiomathes **14** (2004) 57–76
5. Pease, A., Niles, I.: IEEE Standard Upper Ontology: A progress report. Knowledge Engineering Review **17** (2002) 65–70
6. Seibt, J.: Free Process Theory: Towards a typology of processes. Axiomathes **14** (2003) 23–55
7. Sowa, J.F.: Knowledge Representation: Logical, Philosophical and Computational Foundations. Brooks/Cole, Pacific Grove (2000)
8. West, M., Sullivan, J., Teijgeler, H.: ISO/FDIS 15926-2: Lifecycle integration of process plant data including oil and gas production facilities. (Data Model), ISO TC184/SC4/WG3N1328 (2003) Available from: http://www.tc184-sc4.org/wg3ndocs/wg3n1328/lifecycle_integration_schema.html.
9. DSTC, Gentleware, IBM, Sandpiper Software: Ontology Definition MetaModel: Preliminary revised submission to OMG RFP ad/2003-03-40. Technical report, Object Management Group (OMG) (2004)
10. W3C: Web Ontology Language (OWL) Specifications. W3C Recommendation, World Wide Web Consortium (W3C), Cambridge (Massachusetts) (2004) Available from: http://www.w3.org/2004/OWL/.
11. Frege, G.: Die Grundlagen der Arithmetik. Felix Meiner Verlag, Hamburg (1988) edited by C. Thiel.
12. Wittgenstein, L.: Tractatus logico-philosophicus. Routledge & Kegan Paul, London (1922) transl. by Ogden, C. K.
13. Wittgenstein, L.: Philosophische Untersuchungen. Suhrkamp, Frankfurt/Main (1967)
14. Russell, B.: On denoting. Mind **14** (1905) 479–493
15. Tarski, A.: Logic, Semantics, Metamathematics: Papers from 1923 to 1938. 2. edn. Hackett, Indianapolis (1983) ed. by Corcoran, John.

[9] Cf. `http://www.w3.org/TR/swbp-n-aryRelations`

16. W3C: RDF Semantics. W3C Recommendation, World Wide Web Consortium (W3C), Cambridge (Massachusetts) (2004) Available from: http://www.w3.org/TR/rdf-mt/.
17. Hayes, P., Menzel, C.: A semantics for the Knowledge Interchange Format. In: Workshop on the IEEE Standard Upper Ontology at IJCAI, Aug 6, 2001. (2001)
18. Menzel, C.: Common Logic. Available from: http://philebus.tamu.edu/cl/ (2005)
19. Devlin, K.: The Joy of Sets: Fundamentals of Contemporary Set Theory. 2. edn. Springer, Berlin (1993)
20. Barr, M., Wells, C.: Category Theory for Computing Science. 2. edn. Prentice-Hall International Series in Computer Science. Prentice-Hall, London (1995)
21. Guizzardi, G., Herre, H., Wagner, G.: On the general ontological foundations of conceptual modeling. In Spaccapietra, S., March, S.T., Kambayashi, Y., eds.: Conceptual Modeling – ER 2002, Proceedings of the 21st International Conference on Conceptual Modeling. Volume 2503 of Lecture Notes in Computer Science., Berlin, Springer (2002) 65–78
22. Pan, J.Z., Horrocks, I.: Metamodeling architecture of web ontology languages. In Cruz, I.F., Decker, S., Euzenat, J., McGuinness, D.L., eds.: Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, Jul 30 - Aug 1,2001. (2001) 131–149
23. Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D., Rice, J.P.: Open Knowledge Base Connectivity 2.0.3. Specification, Artificial Intelligence Center, SRI International and Knowledge Systems Laboratory, Stanford University (1998)
24. OMG: Unified Modeling Language Specification. Version 1.5, Object Management Group (OMG), Needham (Massachusetts) (2003)
25. Gracia, J.J.E.: Metaphysics and Its Tasks: The Search for the Categorial Foundation of Knowledge. SUNY series in Philosophy. State University of New York Press, Albany (1999)
26. Gracia, J.J.E.: Individuality: An Essay on the Foundations of Metaphysics. SUNY Series in Philosophy. State University of New York Press, Albany (1988)
27. Swoyer, C.: Properties. In Zalta, E.N., ed.: The Stanford Encyclopedia of Philosophy. Winter 2000 edn. Stanford University, Center for the Study of Language and Information (2000) Available from: http://plato.stanford.edu/archives/win2000/entries/properties/.
28. Loebe, F.: An analysis of roles: Towards ontology-based modelling. Onto-Med Report 7, Research Group Ontologies in Medicine, University of Leipzig (2003)
29. Kent, R.: Distributed conceptual structures. In de Swart, H.C.M., ed.: Relational Methods in Computer Science: 6th International Conference, RelMiCS 2001 and 1st Workshop of COST Action 274 TARSKI, Oisterwijk, The Netherlands, October 16-21, 2001, Revised Papers. Volume 2561 of Lecture Notes in Computer Science., Berlin, Springer (2002) 104–123
30. Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the ACM **39** (1992) 95–146
31. OMG: Meta Object Facility (MOF) Specification. Version 1.4, Object Management Group (OMG), Needham (Massachusetts) (2002)
32. Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Mieczyslaw, K., Smith, J.: UML for ontology development. The Knowledge Engineering Review **17** (2002) 61–64
33. Kent, R.: The IFF foundation for ontological knowledge organization. In Williamson, N.J., Beghtol, C., eds.: Knowledge Organization and Classification in International Information Retrieval. Volume 37 of Cataloging & Classification Quarterly. Haworth Press, New York (2004) 187–203

Web Image Semantic Clustering

Zhiguo Gong, Leong Hou U, and Chan Wa Cheang

Faculty of Science and Technology, University of Macau, Macao, PRC
{zggong, ma36575, ma36600}@umac.mo

Abstract. This paper provides a novel Web image clustering methodology based on their associated texts. In our approach, the semantics of Web images are firstly represented into vectors of term-weight pairs. In order to correctly correlate terms to a Web image, the associated text of the Web image is partitioned into semantic blocks according to the semantic structure of the text with respect to the Web images. The weight of a term in the vector of an embedded Web image is calculated with respect to both its local occurrence in semantic blocks and the distances of the blocks to the image. With this method, 'Web image clustering' is transformed into 'term vector clustering'. And a feature based solution is employed in our solution. To reach this objective, we define the associate relations between two terms based on their co-occurrence in the associated text of the Web images. Thus, a term semantic network (*TSN*) is constructed with terms as the nodes and associate relationships as the edges. To cluster terms in *TSN*, *CHAMELEON* algorithm is utilized. In order to determine the significances of terms in each cluster, *HITS* algorithm is applied. Finally, web images are assigned to different clusters based on the similarity between image term vectors and the term vector of the clusters.

1 Introduction

Currently, Web images have been becoming one of the most important information types on the Web. Thus, how to effectively gather, manage and reuse this valuable resource are among the most attractive research topics in the area of Web information retrievals. However, many differences exist in comparison with traditional images in their creation purposes, amount, and semantic coherence. Firstly, Web images are generally created or selected by Web authors as visually supports for their Web page presentations, and no standard exists for the image size and visual qualities. Some images are highly qualified and much valuable for reuse while many others may not have any reuse values. Secondly, the amount of Web images is huge, therefore, manually semantic annotations for Web images, as for many traditional image management systems, is too hard or even impossible. Thirdly, the traditional image management systems are usually dedicated for specific domains which have limited scopes of users and image topics. And domain-based image processing techniques can be exploited for automatic feature extracting and indexing of the images. Web images, however, are produced by authors from different domains. And domain-independent algorithms for feature extracting and indexing of Web images can only be limited to some special features, such as color histogram or textures. As a result, it

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1416 – 1431, 2005.

is hard to couple such lower features with higher semantic descriptions of the Web images. In order to overcome such shortages caused by the heterogeneous and semantic diverse intrinsic of Web images, an effective and efficient algorithm for Web image automatic clustering or classification is definitely desirable.

In [2], we have addressed our techniques on Web image indexing and searching. The semantic extraction of Web images in [2] is based on the HTML structure (DOM tree), the user feedback and Web page segmentations. In this paper, we provide our categorization method of web images based on the term clustering techniques.

In general, topic directory system suffers from lower performance of manual classification of newly collected documents. For example, Yahoo, the largest directory system on the Internet, has more than 1.5 million links in its topic hierarchy and needs as more as 100 editors in its classifying work of the Web documents. Up to now, most commercial categorization work of directory systems is performed manually. Therefore, to find an effectively automatic solution for Web document classification is much desirable and draws many attentions from both academic and industrial communities.

There are two ways for automatic document categorization. One is called *supervised learning techniques*--classification. Such kind of techniques consists of two steps. First, categories are predefined and described by experts. Then, all images are assigned to corresponding categories based on automatic similarity measurements between Web images' subject and the predefined category descriptions. The obvious weakness of such classification methods lies in that large percentage of the heterogeneous Web documents may not be fit to any given predefined category. In classification models, a training set should be manually defined at first, and the size of it may directly affect the precision of the classification result. So it is not suitable for huge amount of Web documents. The other method for categorization is called *unsupervised learning techniques* – Clustering. The main difference between the two is that clustering does not need to predefine the category beforehand, thus reduce manual labors dramatically. With such methods, subject experts can only need to define the label for categorization.

In this paper, we provide a solution for Web image clustering based on the associated text. The semantics of Web images are extracted by partitioning the textual parts of the Web pages with respect to the images. Then, Web images are represented into vectors of term-weight pairs. We suppose that each cluster of Web images can be described with a feature set of term-weight pairs too. The principle behind our approach is based on the observation that co--occurrence words or terms are often closely related to each other in describing the same topics for the Web images. To formalize our discussions, for any term pair t_1 and t_2, we define term relations with $Sup(t_1, t_2)$ (*Support*) and $Conf(t_1, t_2)$ (*Confidence*) which are extracted from our Web document/image database [2]. $Sup(t_1, t_2)$ indicates the absolute co-occurrence of term t_1 and t_2 in the collection, and $Conf(t_1, t_2)$ reflects their co-occurrence relative to t_1's appearances in the collection. Then, a term semantic network (*TSN*) is created with the terms as the nodes and $Sup(t_i, t_j)*Conf(t_i, t_j)$ as the directed edges(t_i, t_j). With this definition for *TSN*, it is obvious that the more value of edges(t_i, t_j), the more possible for t_i and t_j in the same cluster. In our approach, we suppose that terms with strong edges connected in *TSN* are likely to be used together in describing the same topics. In order to cluster Web image topics, we then partition *TSN* into several subgraphs

using algorithm *CHAMELEON* [1]. And we consider each subgraph of *TSN* as one feature cluster for one subject. As a matter of the fact, terms in each cluster are weighed differently in support the cluster topic. For example, the more edges linked to term *t*, the more significance of term *t* as one of the clustering features. We assign weights to terms by using algorithm *HITS* [21]. Finally, we assign Web images to corresponding clusters by the similarity between term vectors of the images and the feature vectors of the clustering. Our experimental results show that retrieval precisions can be improved in comparison with the methodology without using *HITS* or K-Mean partitioning approach.

The reminder of the paper is organized as follows: Section 2 provides a review on related works. Section 3 gives Web image representations. Section 4 addresses our techniques in constructing the term semantic network (*TSN*). And we provide our approaches for Web image clustering in section 5. In section 6, we show our experimental result. Finally in section 7, we conclude this paper.

2 Related Work

In this section, we introduce some past works on general clustering methodologies, as well as on document clustering.

2.1 Works on Fundamental Clustering Techniques

Clustering, which is the process of grouping the data into classes or clusters so that objects within a cluster have high similarity in comparison to one another, is one kind of method in data mining technology. The idea of clustering is that the intra-cluster similarity is maximized and the inter-cluster similarity is minimized. Typical pattern clustering activity involves the following steps [5]:

- pattern representation (optionally including feature extraction and/or selection),
- definition of a pattern proximity measure appropriate to the data domain,
- clustering or grouping,

Feature selection is the process of identifying the most effective subset of the original features to use in clustering. And *Feature extraction* is the use of one or more transformations of the input features to produce new salient features. Either or both of these techniques can be used to obtain an appropriate set of features to use in clustering. In this paper, we consider all terms, which appear in the associated text of the Web image, as the semantic features of the image.

Pattern proximity is usually measured by a distance function defined on pairs of patterns. A variety of distance measures can be found in use in the different communities [5][6][7]. And a simple distance measure like Euclidean distance can always be effective to reflect dissimilarity between two patterns.

The *grouping* step can be performed in a number of ways. The output clustering (or clusterings) can be hard (a partition of the data into groups) or fuzzy (where each pattern has a variable degree of membership in each of the output clusters). Hierarchical clustering algorithms produce a nested series of partitions based on a criterion for merging or splitting clusters by similarity. In general, a partitional clustering algorithm is trying to find one partition that optimizes (usually locally) a clustering criterion.

Actually, based on these three fundamental steps, clustering can be implemented in different models, such as Single Link, Complete Link, Minimal spanning tree, and K-means. In the following, we will introduce some typical methods.

2.1.1 Single Link

Single Link is a simple one of the Hierarchical clustering algorithms. In the single-link method, the distance between two clusters is the *minimum* of the distances between all pairs of patterns drawn from the two clusters respectively (one pattern from the first cluster, the other from the second). The single-link algorithm suffers from a chaining effect [8]. It has a tendency to produce clusters that are straggly or elongated.

2.1.2 Complete Link

In the complete-link algorithm, the distance between two clusters is the *maximum* of all pairwise distances between patterns in the two clusters. And two clusters are merged to form a large cluster if the distance below some predefined threshold. The complete-link algorithm often produces tightly bound or compact cluster [9].

2.1.3 Minimal Spanning Tree (MST)

Most clustering models make all object link together such that it looks like a graph. So graph-theories can be used to solve the clustering problem. One typical technique in graph-theoretic is called Minimal spanning tree (MST). To cluster the nodes in the graph, MST is firstly obtained from the original graph, then, edges with the largest lengths are deleted recursively until ideal clusters are reached. The problem of MST is that the clusters may be quite unbalanced since the algorithm always chose the most weakness edge to cut. It often finds the large cluster group only.

2.1.4 K-Means

K-means is one kind of important partitional clustering algorithms which obtain a single partition of the data instead of a clustering structure. K-means uses the following step to get the cluster:

I. Choose k cluster centers to coincide with k randomly-chosen patterns or k randomly defined points.
II. Assign each pattern (object) to the closest cluster center
III. Re-compute the cluster centers using the new cluster membership
IV. If a convergence criterion is not met, go to step II. Typical convergence criteria are: no (or minimal) reassignment of patterns to new cluster centers, or minimal decrease in squared error.

The weakness of K-means is that it just cut the cluster into simple shape. It is not suitable for many situations.

2.1.5 CHAMELEON

CHAMELEON belongs to the kinds of hierarchical clustering algorithm. It modeling data items as graph and its sparse graph representation of the data items is based on the commonly used *k*-nearest neighbor graph approach. (Figure 1)

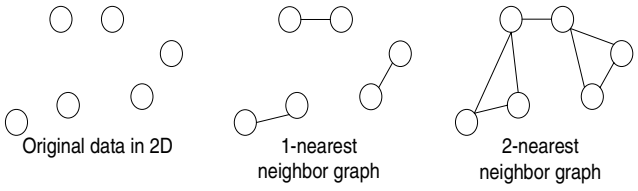


Fig. 1. *k*-nearest graphs from an original data in 2D

It measures the similarity of two clusters based on a dynamic model. In the clustering process, two clusters are merged only if the inter-connectivity and closeness (proximity) between two clusters are high enough relatively to the internal ones of these two individual clusters. The methodology of dynamic cluster modeling used in *CHAMELEON* is applicable to all types of data as long as a similarity metric can be defined. *CHAMELEON* is a successful algorithm to overcome many limitations of the existing hierarchical schemes [14][15][16].

On the other hand, *CHAMELEON* can also be carried out reversely to partition a data set or graph. In fact, *CHAMELEON* utilizes multilevel graph partitioning algorithms to find the initial sub-clusters. In particular, it uses the graph partitioning algorithm which is provided as programming API called *hMETIS* library [17]. *hMETIS* has been shown to quickly produce highly-qualified partitions for a wide range of unstructured graphs and hypergraphs [18][19][20]. *CHAMELEON* uses *hMETIS* to split a cluster C_i into two sub-clusters $C^A{}_i$ and $C^B{}_i$ such that the edge-cut between $C^A{}_i$ and $C^B{}_i$ is minimized and each one of these sub-clusters contains at least 25% of the nodes in C_i. This requirement is often referred to as the *balance constraint* 1.

2.2 Some Works on Document Clustering Model

Document clustering can be implemented either with object cluster model or word cluster model.

In the former model, documents are represented into vectors of term-weight pairs. Therefore, document clustering is transformed into vector clustering in the multi-dimension space. And many research works existed in this model. For example, [11] provided a clustering algorithm called CBC (Clustering By Committee). This work initially discovers a set of tight clusters (high intra-group similarity), called committees, which are well scattered in the similarity space (low inter-group similarity). The union of the committees is however a subset of all elements. The algorithm proceeds by assigning elements to their most similar committee.

In the latter model, features (words) of the cluster are firstly extracted with respect to their semantic relationships. Then, documents are assigned to the corresponding clusters based on the similarity between features of the cluster and the document. In this respect, the documents clustering is actually topic oriented other than document oriented. That mean, if a document contains multiple topics, it will probably be assigned into several clusters. Work [10], for instance, use the information bottleneck method to obtain a better result by using word cluster model. It first finds *word-clusters* that capture most of the mutual information about the set of documents, and then find *document clusters*, that reserve the information. It is much less sparse and noisy than the object cluster model.

Besides, document clustering, there are also many works on document classification. ACIRD: Intelligent Internet Document Organization and Retrieval 3 is a typical work in this model. *Term Semantic Network* (*TSN*) is one of the main concepts used in ACIRD system. It builds the *TSN* based on the term association rule and supports the data representation model.

3 Web Image Semantic Representation

In our system, Web pages are gathered and converted into Document object model (*DOM*) trees by the system crawler. Then, Web images are filtered out by removing meaningless images (or stop images, such as Web site logo, button) before further processing. In order to extract the semantics of a Web image, we partition the Web page of the image into a sequence of semantic blocks with respect to the distances to the image. And we suppose that different blocks have different semantic relevance to the Web image [2]. A term's semantic relevance to an embedded image is calculated both based on its local occurrences in the individual semantic blocks and the semantic distances of the corresponding blocks to the image.

In our approach, we firstly partition the whole associated text of a Web image into three parts, which include *TM* (the texts from <TITLE> and <META> element), *LT* (the text attached to the image display command), and *BT* (the text of <BODY> element), based on their functions in the text. *TM* is often used as the summary of the page content, and it may provide valuable semantic implications to the images especially in an image containers; *LT* is the closest part of the associated text to the image, and often highlight the semantics of the local Web image; *BT* is big in size, but can provide explanations for the contents of the image in most of cases.

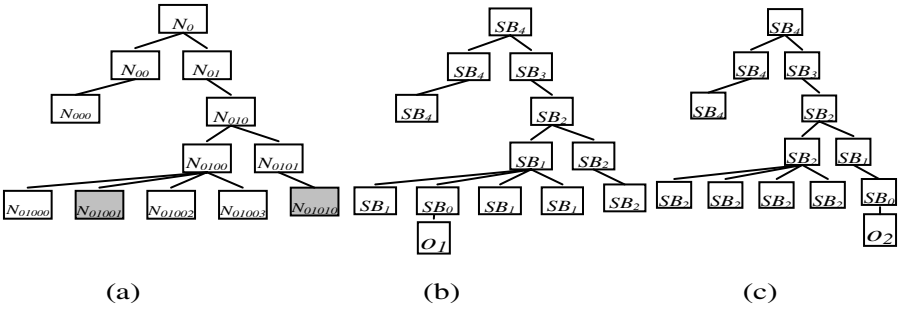


Fig. 2. Semantic Fragmentation

Since the big size of *BT*, it may contain diverse semantics. Therefore, we further partition it into a sequence of semantic blocks with respect to the nested structure of tag elements. Figure 2 is an example for partitioning *BT* blocks. Figure 2(a) represents the original tree structure of a Web page which contains two images o_1 and o_2 in nodes N_{01001} and N_{01010} respectively. Then, Figure 2(b) and (c) shows the semantic fragmentations for image o_1 and o_2 respectively. Formally, for any image o, there exist a sequence of nested nodes E_0, E_1, …E_N of *BT*, and $E_N \supset E_{N-1} \supset \cdots \supset E_0$ such that

E_N=<*Body*> (the root node) and $E_0 \ni o$. And furthermore, no other HTML element exists between E_{j+1} and E_j for any j. Obviously, value N (the number of the nested elements) may be different for different web pages. To simplify our notations, we set N big enough. And if $E_j=BT$, then $E_n=BT$ for any $N \geq n \geq j$. Therefore, semantic blocks are defined as $SB_j=E_{j+1}-E_j$ for $N-1 \geq j \geq 0$ (Figure 2 (b), (c)). Then, we have $BT=\cup_{0 \leq j \leq N-1} SB_j$ with $SB_k \cap SB_l = \Phi$ for any $k \neq l$. Figure 2 illustrates the principle of our Web page fragmentation. In this example, we suppose Web page p has been partitioned into 4 layers of SB_0, SB_1, SB_2, and SB_3, with SB_0 containing Web image o. Then, the semantic relevance of SB_k to o reduces with k increasing from 0 to 3. That is, t_i is more relevant to image o than t_j because of their locations relative to image o. Actually, a term t may appear in several semantic blocks (other than only one). In such a case, t's overall relevance to o should be taken into account of both its importance degrees in all blocks and block's relevances to image o.

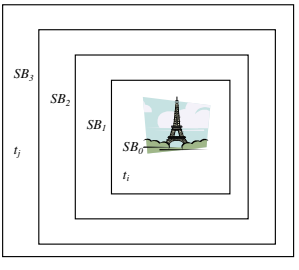


Fig. 3. Semantic Fragmentation for a Web Image

In general, we suppose SB_j ,$0 \leq i \leq N$, are semantic blocks with respect to Web image o, the local occurrence of a term t in block SB_j is calculated as

$$ntf(t)|_{SB_j} = \frac{tf(t)|_{SB_j}}{|SB_j|} \tag{1}$$

where $ntf(t)|_{SB_j}$ is called normalized frequency of term t over SB_j, $tf(t)|_{SB_j}$ is the frequency of term t in SB_j, and $|SB_j|$ is the size of SB_j. Thus, term t's overall semantic relevance to image o through page p is obtained as

$$ttf(t)|_p = \sum_{1 \leq i \leq N-1} w_j * ntf(t)|_{SB_j} \tag{2}$$

where N is the total number of the semantic blocks, and w_j is the weight of SB_j's semantic relevant to the embedded image o. Without loss of generality, we suppose w_j's are normalized, such that $\sum_{1 \leq i \leq N-1} w_j = 1$. In order to determine the values of those w_j's, a greedy algorithm, called *Two-Way-Merging*, is used. In our approach, a binary tree is constructed bottom upward, with SB_j's as all the leaves (Figure 4). Then, merging is carried out recursively, from bottom to top. For each step, a pair of factors

(α^l_i,β^l_i) is obtained, where l stands for the level of merging, α^l_i and β^l_i are merging factors from left and right sub-branches of the processing node respectively, and they are normalized as $\alpha^l_i+\beta^l_i=1$. In the calculation, we choose average precision of retrievals [9] as the objective function for determining (α^1_i,β^1_i). In other words, the values should maximize retrieval precisions of Web images on average. Then, w_j is finally gotten by multiplying all corresponding level factors of SB_j in the tree. For example, in Figure 4, $w_1=\alpha^1_1*\alpha^2_1$, $w_2=\beta^1_1*\alpha^2_1$, $w_3=\alpha^1_2*\alpha^2_1$, $w_2=\beta^1_2*\beta^2_1$.

With above processing, in our approach, a Web image o, which is embedded in page p, can be semantically represented as a vector

$$o=<(t_1, ttf|_p(t_1)), (t_2, ttf|_p(t_2)), \ldots., (t_n,ttf|_p(t_n))>. \quad (1)$$

Therefore, the problem of Web image clustering is transformed into that of term vector clustering.

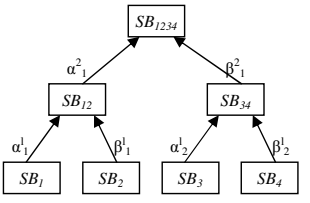


Fig. 4. Two-Way Merging Processing

4 Building Term Semantic Network

To extract topics of Web images, term co-occurrences with respect to Web images are important resource to use. Semantic relationships between terms can be described by the co-occurrences of terms. For example, if two terms often appear together in association with images, they are most likely to be related to the same topics. Term Semantic Network is often used in representing the overall semantic relevances among terms. According to [12], the association rule is formally defined as follows:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. T is a database of transactions. Each transaction t in T is a set of items such that $t \subseteq 1$. An association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \Phi$. The rule $X \rightarrow Y$ holds in the transaction set T with confidence c if c percent of transactions that contain X also contain Y. The rule $X \rightarrow Y$ has support s in the transaction set T if s percent of transactions contain $X \cup Y$.

In this paper, we use a simple mining association algorithm – Apriori [13] -- to mine out the association rules of the words. And we only consider one-to-one term relationship. We define *confidence* (*conf*) and *support* (*sup*) of term association $t_i \rightarrow t_j$ as follows: let

$$D(t_i,t_j) = D(t_i) \cap D(t_j) \quad (2)$$

where $D(t_i)$ stands for the documents include term t_i, $D(t_i) \cap D(t_j)$ stands for the documents that include both t_i and t_j, then, confidence from t_i to t_j is:

$$Conf(t_i, t_j) = \frac{\| D(t_i, t_j) \|}{\| D(t_i) \|} \tag{3}$$

where $\| D(t_i, t_j) \|$ stands for the total number Web documents which contain both t_i and t_j, and $\| D(t_i) \|$ is the total number of Web documents containing term t_i. And

$$Sup(t_i, t_j) = \frac{\| D(t_i, t_j) \|}{\| D \|} \tag{4}$$

where $\|D\|$ stands for the number of document in our Web document database. Above definitions for *Confidence* and *Support* between two words reflect their co-occurrences in the collection. In detail, $Conf(t_i,t_j)$ indicates t_i and t_j's co-occurrence relative to t_i, and $Sup(t_i,t_j)$ indicates their absolute co-occurrences in the collection. In other words, these two functions can get higher values if t_i and t_j are often used together in Web documents. Therefore, in this paper we suppose that two terms are likely to be features for the same topic if they have higher values of *Confidence* and *Support*. And a term semantic network (*TSN*) is created with terms as nodes and $Sup(t_i,t_j)*Conf(t_i,t_j)$as the weighted edge from t_i to t_j.

The initial *TSN* is quite complex because of the huge amount of edges. In order to simplify the graph, we set the minimum value of *support* and *confidence* and try to remove the edges whose *support* and *confidence* values below the thresholds. Currently, in this paper, we empirically set the minimum support to 0.05 and minimum confidence to 0.1. With this filtering, the edges of *TSN* reduce to 4 millions in number.

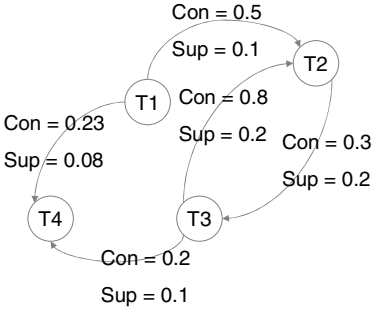


Fig. 5. *TSN*: *Con* and *Sup* Annotated to the edges

Figure 5 shows our *TSN*. In this figure, each node represents a term, and each edge is annotated with two values (*support* and *confidence*) to represent its weight. In our implementation, we use a single value *Rank* = *support* * *confidence* to represent the edge weight. We will discuss how to use this graph for term clustering in following section.

5 Web Image Clustering

To cluster Web images, we firstly mine out term clusters using *TSN*. And in our system, we suppose each term cluster is on the same topics. And we split terms of *TSN* into sub-graphs using *CHAMLEON* algorithm [1] and extract the weights of terms in each cluster by using *HITS* algorithm [21]. And finally we assign Web images to corresponding topic clusters by calculating the similarities between Web image features and topic features.

5.1 Term Clustering

CHAMELEON is an attractive algorithm due to its adaptivity to different clustering applications. And the algorithm includes two steps- splitting and merging.

In splitting phase, *TSN* is partitioned into a set of sub-graphs. Actually, it splits the graph based on the hMETIS algorithm. hMETIS can split a cluster into two sub-clusters such that the edge-cut between these two clusters is minimized and each one of these sub-clusters contains at least 25% of the nodes in the original cluster (balance constraint) [17]. This balance constraint in *CHAMELEON* algorithm is critical to prevent outliers when splitting *TSN* and generate natural clusters.

And the second phase recursively merge some sub-clusters with respect to two indicators *RI* (Relative Inter-connectivity) and *RC* (Relative Closeness). Two clusters are merged if the *RI* and *RC* are over some thresholds. The relative inter-connectivity between a pair of clusters C_i and C_j is given by

$$RI\ (C_i, C_j) = \frac{\left|EC_{\{Ci,Cj\}}\right|}{\frac{\left|EC_{Ci}\right| + \left|EC_{Cj}\right|}{2}} \tag{5}$$

Where $EC_{\{Ci,Cj\}}$ is the set of edges crossing cluster C_i and C_j. And EC_{Ci} and EC_{Cj} are the size of its min-cut bisector (i.e., the weighted sum of edges that partition the graph into two roughly equal parts) in C_i and C_j respectively.

$$RC(C_i,C_j) = \frac{\overline{S}_{EC_{\{Ci,Cj\}}}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{ECCi} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{ECCj}} \tag{6}$$

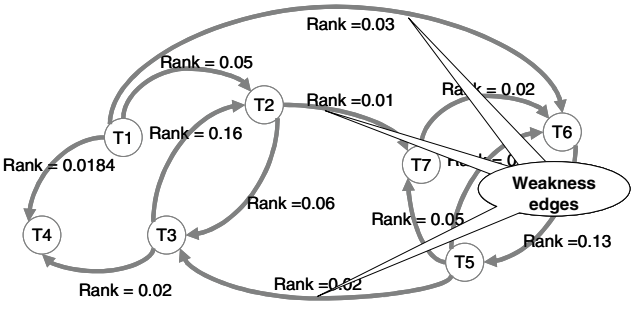


Fig. 6. *TSN*: Choosing Weak Edges to Split

where $\overline{S}_{ECCi}$ and $\overline{S}_{ECCj}$ are the average weights of the edges that belong in the min-cut bisector of clusters *Ci* and *Cj* , respectively, and $\overline{S}_{EC_{\{Ci,Cj\}}}$ is the average weight of the edges that connect vertices in *Ci* to vertices in *Cj* .

Figure 6 shows an example of how to calculate the *RI* and *RC*.

In splitting phase, *CHAMELEON* utilizes *hMETIS* algorithm to split *TSN* based on the weight of each edge. In this example, it chooses to cut the edge $E_{T1\rightarrow T6}$, $E_{T2\rightarrow T7}$ and $E_{T5\rightarrow T3}$ due to their weaknesses comparing with others. After this process, terms of *TSN* are categorized into two clusters: Term Cluster 1 (*T1, T2, T3, T4*) and Term Cluster 2 (*T5, T6, T7*) (Figure 7).

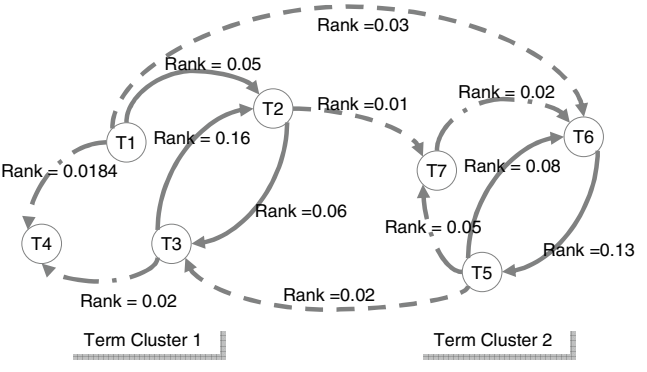


Fig. 7. *TSN*: $EC_{\{Ci,Cj\}}$ and $\overline{S}_{EC_{\{Ci,Cj\}}}$ Calculation

Figure 7 provides an illustration on how to calculate equation (4) and (5) in *CHAMELEON* algorithm. In above figure, we represent edges with different line types. And lines of (⟶) represents edges between two terms from the same cluster, lines of (- - -▶) represent edges between two terms across different clusters, and lines of (−·▶) stand for edges between two terms from the cluster's min-cut bisector. Table 1 shows the calculations and corresponding values.

Table 1. Paramiter Calculations in *CHAMELEON* Algorithm

$EC_{\{Ci,Cj\}}$	$\lvert E_{T1\rightarrow T6}\rvert+\lvert E_{T2\rightarrow T7}\rvert+\lvert E_{T5\rightarrow T3}\rvert$	0.06
EC_{Ci}	$\lvert E_{T1\rightarrow T4}\rvert+\lvert E_{T3\rightarrow T4}\rvert$	0.0384
EC_{Cj}	$\lvert E_{T7\rightarrow T6}\rvert+\lvert E_{T5\rightarrow T7}\rvert$	0.07
$\overline{S}_{EC_{\{Ci,Cj\}}}$	$(\lvert E_{T1\rightarrow T6}\rvert+\lvert E_{T2\rightarrow T7}\rvert+\lvert E_{T5\rightarrow T3}\rvert)/3$	(0.03+0.02+0.01)/3=0.02
$\overline{S}_{ECCi}$	$(\lvert E_{T1\rightarrow T4}\rvert+\lvert E_{T3\rightarrow T4}\rvert)/2$	(0.0184+0.02)/2=0.0192
$\overline{S}_{ECCj}$	$(\lvert E_{T7\rightarrow T6}\rvert+\lvert E_{T5\rightarrow T7}\rvert)/2$	(0.05+0.02)/2=0.035
$\lvert C_i\rvert$	Number of terms in *Ci*	4
$\lvert C_j\rvert$	Number of terms in *Cj*	3

With all the values in table 1, we can easily calculate out the values of *RI* and *RC* as 1.107 and 0.488 respectively. With these two functions, two different merging schemes can be implemented in *CHAMELEON*. The first scheme merges only those pairs of clusters whose relative inter-connectivity and relative closeness are both above some user specified thresholds. *CHAMELEON* visits each cluster C_i , and checks to see if any one of its adjacent clusters C_j satisfy the following two conditions:

$$RI(C_i, C_j) \geq T_{RI} \cap RC(C_i, C_j) \geq T_{RC} \tag{7}$$

where T_{RI} and T_{RC} are the user specified thresholds.

In the second scheme, *CHAMELEON* uses one function as the objective indicator which is actually a combination of *RI* and *RC* as

$$RI(C_i, C_j) * RC(C_i, C_j)^{\alpha} \tag{8}$$

where α is a user specified parameter which can be tailored with respect to different applications. If α > 1, then *CHAMELEON* gives a higher importance to the relative closeness, and when α < 1, it gives a higher importance on the relative inter-connectivity. In this paper, we use the second scheme with α =1. And the term set *C* of *TSN* is partitioned into several clusters C_0, C_1, …, C_n, such that

$$C = \bigcup_{0 \leq i \leq n} C_i \tag{9}$$

And each term cluster *Ci* is assumed to dedicate on one topic.

5.2 Extracting Semantic Significances of Terms

For each cluster *Ci*, all terms in *Ci* is taken as its features which well reflect the topic of the cluster. However, their significances to the cluster may not be the same in general. In this paper, we also want to determine term's weights based on *TSN*. To reach this objective, we calculate the weights of terms in cluster (C_i) by using *HITS* algorithm [21]. Actually, *HITS* is a popularly used link analytic algorithm, it extracts the degrees of node importance from the link structure of hyperlinked environment (Web). In this paper, we use it to extract the significances of terms with respect to the semantic links among them. In our context, terms in each cluster *Ci* are linked with direct edges. The ideas behind *HITS* algorithm is that a node is important in the directed graph if more links are either from or to this node. In fact, for each node *HITS* uses two concepts—*Authority* and *Hub*—in its analysis. *Authority* indicates the degree of links coming to the node, while *Hub* indicates the degree of links going from this node. A good *Authority* means it is pointed by many good nodes and a good *Hub* means it point to many good nodes. The relationship between these two factors is shown in Figure 8.

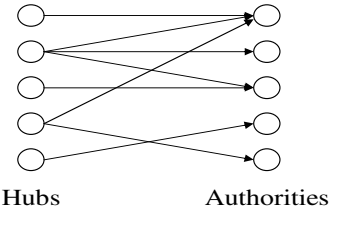


Fig. 8. A densely linked set of hubs and authorities

Because we have already obtained the values of support and confidence between different terms, we define the relationship between term t_i and term t_j as follows:

$$R(t_i, t_j) = Sup(t_i, t_j) \times Conf(t_i, t_j) \tag{10}$$

We initially assign values for *Authority* and *Hub* as follows:

$$Authority(t_i) = \sum_{t_j:(t_j,t_i)\in E} R(t_j, t_i) \tag{11}$$

$$Hub(t_i) = \sum_{t_j:(t_i,t_j)\in E} R(t_i, t_j) \tag{12}$$

Then, *HITS* uses an Interative Algorithm to get the final values for them with:

$$Authority(t_i) = \sum_{t_j:(t_j,t_i)\in E} Hub(t_j) \tag{13}$$

$$Hub(t_i) = \sum_{t_j:(t_i,t_j)\in E} Authority(t_j) \tag{14}$$

Because *HITS* has already shown that the nodes with higher values of *Authority* and *Hubs* are important in the semantic network, we define the weight of each term in cluster *C* by the following formula:

$$Rank_c(t_i) = Authority(t_i) + Hub(t_i) \tag{15}$$

Then, cluster C can be described with vector as

$$C = \langle Rank_C(t1), Rank_C(t2), \ldots, Rank_C(tn) \rangle. \tag{16}$$

5.3 Assigning Web Images to Corresponding Clusters

In the previous discussions, both Web images and topic clusters are represented into vectors of term-weigh pairs (equation 1 and 16). Then, Web image assignments to the clusters are based on similarity measurement between the image vectors and the cluster vectors. The following equation shows our similarity measurement:

$$Sim(o,c) = \frac{\sum \left(Rank_c(t_i) \times ttf(t_i)\right)}{\|o\|_2 + \|c\|_2} \tag{17}$$

where $o = \langle (t_1, ttf|_p(t_1)), (t_2, ttf|_p(t_2)), \ldots, (t_n, ttf|_p(t_n)) \rangle$ is a Web image and c is a cluster, $\|o\|^2$ is the norm of the image object, and $\|c\|^2$ is the norm of the cluster feature. These two values are defined as:

$$\|c\|^2 = \sqrt{(rank_c(t_1))^2 + (rank_c(t_2))^2 + \ldots + (rank_c(t_n))^2} \tag{18}$$

$$\|o\|^2 = \sqrt{ttf(t_1)^2 + ttf(t_2)^2 + \ldots + ttf(t_n)^2} \tag{19}$$

With this method, a Web image may be conceptually related to several clusters. So if the $Sim(o,c) > \gamma$, we assign the object o to cluster C.

6 Experiments

In this section, we compare our proposed approach with other two feature based clustering methods: (1) *CHAMELEON* algorithm without using *HITS* and, (2) a variant of K-Means feature clustering algorithm.

The web crawler in our system crawled more than 1000000 web pages from internet and only about 50000 pages are filtered out for the experiment. Here, all useful pages are the ones which contain image link(s). Our crawler guarantees that Web page are fully crawled (all child links from the page are crawled) in order to support our semantic extractor's [2]. To carry out the experiment, we defined labels for each testing image. And Recall/Precision is used as the objective function.

K-Means is one of the most popularly used clustering algorithms. The basic idea of traditional K-Means algorithm is to recursively re-compute the cluster centers using new cluster membership until no more change. The distance between two objects is symmetric. However, this traditional method can not be directly used to slit our *TSN*. In our situation, firstly, *TSN* is a directed graph. Therefore, the distance implied in *TSN* between two terms is asymmetric. We modify *TSN* into an undirected graph by simply defining the distance between two terms as:

$$Dist(t_i, t_j) = R(t_i, t_j) + R(t_j, t_i) \tag{18}$$

Where $R(t_i,t_j)$ and $R(t_j,t_i)$ are as in equation (10). Secondly, if t_i does not have a direct edge to t_j, we need to use *Prefect Term Support* (*PTS*) algorithm to uniquely define the distance between them [3]. After above two preprocessing steps, we apply K-Means algorithm to split all terms into clusters. Then, we use the same similarity measurement function to assign all images to corresponding clusters. We call this clustering method as 'Variant of K-Means'.

In Section 5.2, we used *CHAMELEON* algorithm to split the *TSN*. Then, *HITS* algorithm is employed to extract the weights of terms in the clusters. In our experiment, we try to compare it with the one without using *HITS* algorithm processing.

In this experiment, we start with 32 seed points for Variant of K-Means algorithm and use *CHAMELEON* to split the *TSN* network into 32 clusters too. As a result, we find only 17 out of 32 of the clusters are meaningful. The other 15 clusters are hard to be identified as coherence clusters. Then we manually combine some clusters that have the same meaning together, after this adjustment, we got 13 meaningful clusters from the K-Means algorithm (table 2).

With *CHAMELEON* algorithm, we can get about 23 meaningful (coherence) clusters. After manually merging clusters which have the same semantics, we got 17 meaningful clusters. This result reveals that *CHAMELEON* algorithm can always produce much finer clusters that the Variant of K-Means.

Table 2. The meaningful clusters number for each method

Method Name	Meaningful Clusters Num	
	After running the algorithm	After human adjust
K-Means	17	13
Without HITS	23	17
With HITS	23	17

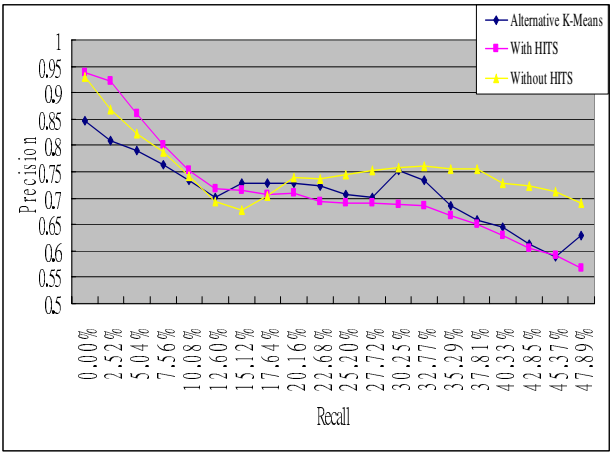


Fig. 9. The statistical result of our system

In Figure 9, we show performances of all the three methods. As we see in the figure, K-Means has the lowest performance in these three methods.

When we used *CHAMELEON* algorithm to split the *TSN*, the result is better than K-Means. If we used *HITS* to weigh cluster features, the query performance is better within the first 17.64% of recall rate than that of the one without running *HITS*. Although the improvement is not very high, we still think it is very useful since the large amount of the Web query result. As a matter of the fact, most Web users can only browse the result within top 10% of the query result. Therefore, it is worth of running *HITS* for weighing terms in clusters.

7 Conclusions and Future Work

In this paper, we have addressed our approach for Web image clustering. Our model is feature based (other than object based). Therefore, one image may be assigned into multiple clusters if it semantically related to more topics. This property is more effective in the Web information retrieval systems since the topic comprehensive of the Web information. Our model is based on the combination of several existing machine learning and data mining techniques with some necessary modifications.

We firstly represent Web images with vectors of term-weight pairs. Then a term semantic network (*TSN*) is constructed by using semantic associations between terms. The semantic associations in our system is described with two functions—*Confidence* and *Support*. Thirdly, we cluster terms by using *CHAMELEON* algorithm. Finally, in order to extract the weights of terms in each cluster, *HITS* algorithm is used. We assign Web images into clusters with respect to the similarity measurements between image vectors and cluster features. Our experimental result shows that our approach can rank relevant result earlier in the query result. That is necessary in the Web query environment.

References

1. George Karypis, Eui-Hong (Sam) Han, Vipin Kumar, CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, computer, Volume 32 , Issue 8 (August 1999) 68-75
2. Zhiguo Gong, Leong Hou U and Chan Wa Cheang, An Implementation of Web Image Search Engine, In Proceedings of the 7th International Conference on Asian Digital Libraries, ICADL, (2004)
3. Lin, S. H., Chen, M. C., Ho, J. M., and Huang, Y. M., ACIRD: Intelligent Internet Documents Organization and Retrieval, IEEE Transactions on Knowledge and Data Engineering (2002)
4. K.S. Jones and D.M. Jackson, The Use of Automatically-Obtained Classifications for Information Retrieval, Information Processing and Management (IP&M), vol. 5, (1970) 175-201
5. JAIN, A. K. AND DUBES, R. C. Algorithms for Clustering Data. Prentice-Hall advanced reference series. Prentice-Hall, Inc., Upper Saddle River, NJ. (1988)
6. ANDERBERG, M. R. Cluster Analysis for Applications. Academic Press, Inc., New York, NY. (1973)
7. DIDAY, E. AND SIMON, J. C. Clustering analysis. In Digital Pattern Recognition, K.S. Fu, Ed. Springer-Verlag, Secaucus, NJ, (1976) 47–94
8. NAGY, G. State of the art in pattern recognition. Proc. IEEE 56, (1968) 836–862
9. BAEZA-YATES, R. A. Introduction to data structures and algorithms related to information retrieval. In Information Retrieval: Data Structures and Algorithms, W. B. Frakes and R. Baeza-Yates, Eds. Prentice-Hall, Inc., Upper Saddle River, NJ, (1992) 13–27
10. N Slonim, N Tishby, Document Clustering using Word Clusters via the Information Bottleneck Method, SIGIR, (2000)
11. P Pantel, D Lin, Efficiently Clustering Documents with Committees, PRICAI, 2002
12. R. Agrawal, T. Imielinski, and A. Swami, Mining Association Rules between Sets of Items in Large Databaseds, Proc. ACM SIGMOD Int'l Conf. Management of Data, (May 1993)
13. R. Agrawaland R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int'l Conf. Very Large Data Bases, (VLDB), (Sept. 1994)
14. A.K. Jain and R. C. Dubes. Algorithms for Clustering Data. Prentice Hall, (1988)
15. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: a robust clustering algorithm for categorical attributes. In Proc. of the 15th Int'l Conf. on Data Eng., (1999)
16. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. In Proc. of 1998 ACM-SIGMOD Int. Conf. on Management of Data, (1998)
17. G. Karypis and V. Kumar. hMETIS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, (1998)
18. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48(1) (1998) 96–129
19. G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In Proceedings of the Design and Automation Conference, (1999)
20. C. J. Alpert. The ISPD98 circuit benchmark suite. In Proc. of the Intl. Symposium of Physical Design, (1998) 80–85
21. J.M. Kleinberg. Authorative sources in a Hyperlinked Environment. J. ACM 46:5, (1999) 604-632

Biomedical Retrieval: How Can a Thesaurus Help?

Leonie IJzereef[1], Jaap Kamps[1,2], and Maarten de Rijke[1]

[1] Informatics Institute, University of Amsterdam
[2] Archives and Information Studies, Faculty of Humanities, University of Amsterdam

Abstract. Searching specialized collections, such as biomedical literature, typically requires intimate knowledge of a specialized terminology. Hence, it can be a disappointing experience: not knowing the right terms to use and being unaware of synonyms or variations in terminology might result in low recall scores. We study the role of a thesaurus in the biomedical information retrieval process. We start by giving a description of vocabulary mismatch problems between natural language queries and relevant documents in biomedical literature search; we provide a detailed case study and observe the impact of vocabulary mismatch problems on retrieval effectiveness. Additionally, we analyze the associated MeSH thesaurus terms used to index the documents in the collection. Based on our observations, we propose a method for exploiting the MeSH thesaurus to improve retrieval effectiveness and, more specifically, to increase recall. We carry out a series of thesaurus-based retrieval experiments that show substantial performance improvements. We conclude with a detailed analysis of the retrieval results.

1 Introduction

In the rapidly growing domain of biomedicine, large numbers of papers are published every day. The resulting information overload makes it hard for scientists to stay up-to-date on the latest findings. Therefore, researchers resort to online databases to identify only that part of the literature that is relevant to their own research focus.

To be able to effectively use a bibliographic search engine, a good and detailed understanding of the topic is necessary to choose the right query terms that retrieve all and only the relevant literature. If a scientist lacks domain knowledge when looking for literature on a specific topic, the retrieval process can be a real challenge: not knowing the right terms to use and being unaware of synonyms or variations in terminology might result in low recall scores.

The classic approach to overcome the mismatch between natural language queries and documents relevant to the user's information need is to use controlled vocabularies. Since the early 1960s, controlled vocabularies such as thesauri have been used to improve the retrieval process [13]. A controlled vocabulary dictates what are the preferred terms to use; selected terms are assigned to each publication by a human indexer, and since search requests are also formulated using

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1432–1448, 2005.

the controlled terminology, there is no vocabulary mismatch. This method is often called *manual indexing*, to contrast it with *automatic indexing* that uses (selected terms in) the free-text of publications as indexing terms. The task of a searcher boils down to locating the appropriate controlled terms, a task that turns out to be highly non-trivial in practice [18, 10]. Perhaps professional search intermediaries or experienced users are well equipped to select the right search term, but the effectiveness of average end-users varies greatly.

The effectiveness of controlled vocabularies for information retrieval has been extensively studied in the literature, dating back to the seminal work at Cranfield [3, 4]. Intuitively, it seems obvious that thesauri can overcome vocabulary mismatch problems; however, previous experimental studies have shown that it's highly non-trivial [21]. Below, we discuss some of the relevant research; an encyclopedic overview is beyond the scope of this paper, however. All in all, the literature gives, at best, mixed results on the effectiveness of controlled vocabularies for information retrieval.

Our aim is to better grasp how a thesaurus can help improve the retrieval effectiveness of ordinary, natural language queries. Our strategy is the following. First, we focus on the potential vocabulary gap in biomedical literature retrieval: we provide a detailed study of vocabulary mismatch problems between natural language queries and relevant documents and show its impact on retrieval performance. In addition, we analyze the thesaurus terms manually assigned to the documents in the collection. Based on our observations, we carry out retrieval experiments and discuss how a thesaurus can be used to improve retrieval effectiveness.

The main contributions of this paper are two-fold:

- A detailed analysis of retrieval queries and relevant documents showing that vocabulary mismatch problems have a negative impact on retrieval effectiveness. This analysis together with the analysis of the assigned thesaurus terms suggests that the semantic knowledge provided by a thesaurus can be useful for biomedical retrieval in two ways: its lexical information can be used as a controlled vocabulary to overcome problems with synonymy and lexical variance and its relational knowledge is potentially useful for identifying relevant related terms.
- We demonstrate the use of thesaurus terms assigned to documents for blind and relevance feedback and provide an analysis of the results. We find that using thesaurus-based feedback can improve both precision and recall. However, for the relevance feedback methods to be successful some effort on the part of the user is needed. Nevertheless, for a scientist interest in high recall values, e.g., looking for all relevant literature on a topic, this investment may be worthwhile.

The remainder of this paper is organized as follows. In Section 2 we describe the thesaurus and evaluation data we use. In Section 3 we provide a detailed case studies of the queries, relevant documents and assigned thesaurus terms of four actual retrieval topics. Section 4 presents the results and an in-depth analysis

of some thesaurus-based retrieval experiments. Finally, in Section 5, we draw conclusions and present directions for further research.

1.1 Related Work

For the open domain, it has been shown that it's hard to use controlled vocabularies due to ambiguity of query words [23]. For more restricted domains, such as biomedicine, there is renewed interest in using controlled vocabularies and semantic knowledge sources due to expanding domains and increasing information needs. In the field of biomedicine, more than 100 different controlled vocabularies (including thesauri and ontologies) are available [16]. Moreover, these vocabularies are already being used for cataloging, classifying, and indexing literature.

Srinivasan [20] compares query expansion based on a statistical thesaurus with expansion via retrieval feedback. She concludes that combining both term selection methods gives the best results, but that the improvement is relatively small in comparison with standard free-text based blind feedback methods.

The term selection method used by French et al. [6] is comparable to the method of Srinivasan: for every word/phrase a list of associated thesaurus terms is computed based on co-occurrences in a training set. However, query augmentation is done by selecting those terms of the list that have been assigned to the greatest number of documents relevant to the query. This gold standard experiment showed that adding one or more suggested terms to the query can potentially improve retrieval effectiveness significantly. Nevertheless, the automatic term selection procedure still has to be defined.

Kostial and Paralic [17] describe a thesaurus-based document boosting procedure (using MeSH and a medical document collection). They combine a basic retrieval procedure with a simple formula based on overlap between thesaurus terms assigned to the query and the documents. The results are promising, but they also circumvent the term selection procedure by assuming that terms relevant to the query are known.

There are also more recent, and more positive, results. Kraaij et al. [12] use thesaurus based relevance feedback for their TREC Genomics 2004 ad hoc task [22] experiments. After a first basic retrieval run, the MeSH thesaurus headings of the top 3 documents are used for a second MeSH retrieval run. They show that a combination of the results of both runs outperforms the basic run, but that the added value of the MeSH run is not convincing. Shallow analysis showed that it only seems to improve precision.

Using a bibliographic database, Savoy [19] evaluates and compares the retrieval effectiveness of various free-text and (human controlled) controlled vocabulary search models. He concludes that the best mean average precision is obtained when both free-text and controlled vocabulary retrieval are combined.

Another feedback technique is described by Kamps [11]: he suggests reranking of the set of initially retrieved documents based on controlled vocabulary terms assigned to documents. He reports a significantly improved retrieval effectiveness based on evaluation on two different domain-specific bibliographic collections, above and beyond the use of standard Rocchio blind feedback.

2 Thesaurus and Data Collection

For our detailed case study and experiments we use the National Library of Medicine's MeSH [15]. This choice is based on the features of our data collection: the MEDLINE [14] bibliographic database we use contains citations that are indexed with controlled vocabulary terms from the MeSH thesaurus. Before giving a more detailed description of our data collection, we recall the main features of the MeSH thesaurus.

2.1 The MeSH Thesaurus

The MeSH thesaurus is used by the National Library of Medicine for indexing biomedical journals and cataloging books, documents and audiovisuals. The core of the MeSH thesaurus is a hierarchical structure that consists of sets of terms naming descriptors. At the top level we find 15 general category headings, such as *Diseases* and *Chemicals and Drugs*. At deeper levels we find more specific headings such as *Brain infarction* (sixth level of *Diseases* branch) or *Dissociative Anesthetics* (ninth level of *Chemicals and Drugs*).

The hierarchy is an eleven-level tree structure that contains over 22,500 headings. Besides the hierarchical structure there are many cross-references that map headings to each other. The main cross-reference fields that can be included in a descriptor's record are the following.

Scope Note Provides additional information about the MeSH heading, which can include related MeSH terms.
See also Contains related terms that may be of interest.
Previous Indexing Contains the MeSH term used before the current descriptor became available.

Together with a descriptor, one or more qualifiers (83 in total) can be used to specify a particular aspect of the descriptor. For example: the qualifier *complications* can be used with diseases to indicate conditions that co-exist or follow.

In addition to the hierarchical structure, there is a separate database with over 139,000 Supplementary Concept Records that consists of chemicals mainly. These supplementary headings are mapped to one or more headings in the main MeSH tree.

It is well known that any thesaurus of the size of MeSH has problems with completeness and consistency [1, 2]. Therefore, it would be useful to analyze MeSH to determine its strengths and weaknesses, and their influence on retrieval. However, for our experiments we take the MeSH thesaurus at 'face value.'

2.2 TREC Genomics Data Collection

To be able to study vocabulary mismatch problems between queries and documents and to look into the potential role of a thesaurus, we use the TREC 2004 Genomics Track ad hoc task [22] data collection. This collection consists of a

selection of 10 years (1994–2003) of MEDLINE citations containing over 4.5M abstracts, 50 retrieval topics and accessory gold-standard data.

Every document in the collection is manually indexed with one or more MeSH headings (from the main tree) and additional qualifiers. For our document collection, this results in 2.6 million unique descriptor-qualifier(s) combinations. In our study, we only take the descriptors into consideration and therefore we ignore the qualifiers. Furthermore, we excluded some frequent but in this context not content-bearing headings such as *Support, Non-U.S. Government* and *Comparative Study*, treeless headings such as *Male* and *Female* and headings with low discriminating values such as *Human* and *Animals*. This leaves us with a total of 21,930 unique headings assigned to the documents in our collection. There is some variation in the number of MeSH headings assigned to each document. Figure 1(Left) shows the distribution of the number of headings assigned to documents. For every document, one or more headings can be marked as main topic of the document. Every heading is placed at one or more nodes in the thesaurus. Figure 1(Right) shows the distribution of unique headings, as well as the distribution of heading occurrences, over the number of nodes at which

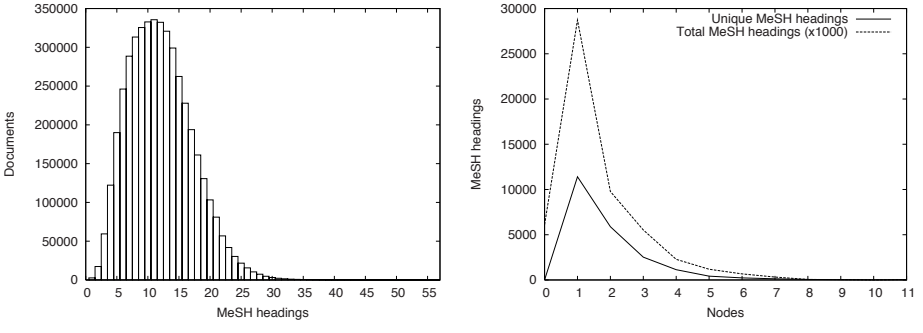


Fig. 1. (Left): Number of MeSH headings assigned to the documents. (Right): Nodes per MeSH heading.

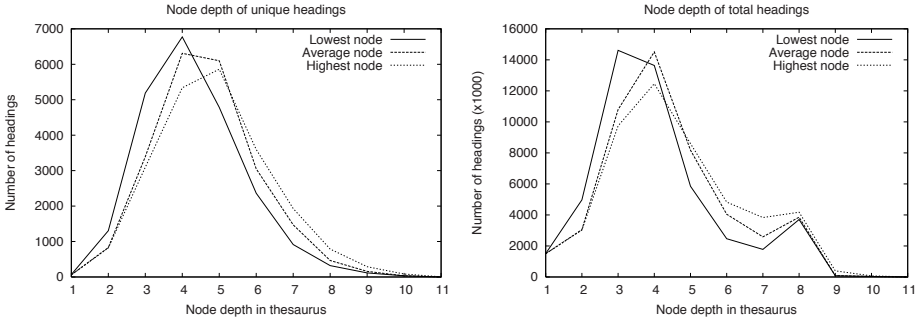


Fig. 2. Node depth of MeSH headings. (Left): for every unique heading (types). (Right): for all headings in collection (tokens).

they are placed in the thesaurus. The depth of the nodes in the thesaurus is related to the specificity of the heading; the deeper the heading is placed in the thesaurus, the more specific it is. Since a heading may be placed on multiple nodes, we can define the depth of a heading by the minimal, maximal, or average depth of its placements in the thesaurus. Figure 2 shows (Left) the distribution of unique MeSH headings and (Right) all heading occurrences over their depth in the thesaurus.

2.3 Evaluation Topics

We selected four topics for our study of vocabulary gaps between the query (the topic's title) and the document collection. The selection is based on the outcome of a retrieval run (with the topic title as the query) with a baseline vector space-based retrieval system. Many other, possibly better performing approaches could have been chosen here [8]. For the TREC 2004 Genomics track ad hoc task systems using stemming and feedback methods turned out to be the most effective. Systems attempting to map controlled vocabulary terms did not fare as well. Given that we are focusing on the role of the thesaurus in the retrieval process, we decided to use a rather basic retrieval system. With a mean average precision (MAP) score of 0.1716 over all 50 topics, our retrieval score is somewhat lower than the mean MAP of the TREC Genomics 2004 ad hoc task participants. However, recall that we only use the short topic statement of the title field, whereas most other participants use the given additional information about the information need too.

We selected one well performing topic (Topic 9 requesting "mutY"), one average performing topic (Topic 21 asking for "Role of p63 and p73 in relation to DNA damage") and two poorly performing topics (Topic 1 and 14 targeting "Ferroportin-1 in humans" and "Expression or Regulation of TGFB in HNSCC cancers," respectively). The four selected topics can be found in Table 1; we include the MAP and recall at 1,000 documents for the title only-based run.

3 Case Studies

In this section, we compare the vocabulary used in natural language queries, the textual content of relevant documents and the MeSH headings assigned to these documents.

3.1 Queries and Relevant Documents

For every topic we take the topic title as our query. We realize that these queries might not perfectly reflect the information need, but since these titles have been formulated by real biologists, we assume that they closely approximate genuine search queries. In this section, we compare the queries and the textual content of the relevant documents.

As can be seen in Table 1, the topic with the shortest title (topic 1) achieves the highest mean average precision of the four. In the relevant documents for this

Table 1. Selected topics with mean average precision and number of retrieved relevant documents (max. 1000 docs retrieved). For the topics 1, . . . , 4 listed below, the original TREC topic IDs are 9, 21, 1, and 14, respectively.

Topic		MAP rel_ret
1. Title:	mutY	
Need:	Find articles about the function of mutY in humans	0.8676
Context:	mutY is particularly challenging, because it is also known as hMYH. This is further complicated by the fact that myoglobin genes are also typically located in search results.	113/115
2. Title:	Role of p63 and p73 in relation to DNA damage	
Need:	Do p63 and p73 cause cell cycle arrest or apoptosis related to DNA damage?	0.1910 40/80
Context:	DNA damage may cause cell cycle arrest or apoptosis. p63 and p73 may play a role in mediating these sequelae of DNA damage.	
3. Title:	Ferroportin-1 in humans	
Need:	Find articles about Ferroportin-1, an iron transporter, in humans.	0.0000 1/79
Context:	Ferroportin1 (also known as SLC40A1; Ferroportin 1; FPN1; HFE4; IREG1; Iron regulated gene 1; Iron-regulated transporter 1; MTP1; SLC11A3; and Solute carrier family 11 (proton-coupled divalent metal ion transporters), member 3) may play a role in iron transport.	
4. Title:	Expression or Regulation of TGFB in HNSCC cancers	
Need:	Documents regarding TGFB expression or regulation in HNSCC cancers	0.0000 0/21
Context:	The laboratory wants to identify components of the TGFB signaling pathway in HNSCC, and determine new targets to study HNSCC.	

query we find that the query word *mutY* occurs in almost every relevant document. As that there are only 168 documents in the corpus that contain *mutY*, this single word query gives very good results. However, if a less frequent synonym such as *hMYH* had been chosen as keyword, scores would have decreased dramatically. An example of this can be seen in topic 3: The main keyword *ferroportin-1* has many synonyms (e.g. *IREG1* and *SLC11A3*) and these can all be found in the relevant documents. By using only *ferroportin-1* together with the very frequently occurring term *human*, only a single relevant document is retrieved. A similar problem occurs in topic 4. Here, two acronyms are used: *TGFB* for *Transforming Growth Factor beta* and *HNSCC* for *Head and Neck Squamous Cell Carcinoma*. Since both terms occur only as spelled out terms, this effects the retrieval score dramatically.

However, choosing 'wrong' synonyms or acronyms is not the only reason for poor retrieval scores. For all four topics we see that besides the keywords (or their synonyms/acronyms) one or more semantically related words occur frequently in the relevant documents. As we can see in the description of topic 3, *iron*

transport plays a central role in this topic. If we look at the most frequently occurring words/phrases in the relevant documents, we find many closely related terms that are not directly expressed in the query. Some examples: *iron overload*, *iron metabolism*, *transferrin*, *iron deficiency*, *ferritin*, *iron homeostasis*. This is also the case for topic 2, where we see various terms that are closely related to the query, such as *p53*, *cell death*, *tumors*, *transcription* and *transactivation*. Now, it comes as no surprise that we find synonyms and semantically related terms of the keywords in the text of the relevant documents. The key question is how to identify these important terms and how to use them to improve retrieval performance. We hope that detailed analyses such as done in this paper will give us answers to these questions.

3.2 Relevant Documents and Thesaurus Terms

All documents in our collection are indexed with MeSH headings, hence we can use this meta-data in our retrieval process. To gain more insight in the role these thesaurus terms and the thesaurus itself could play in the retrieval process, we studied the relation between textual content of the documents and MeSH headings. Before describing this, we show how the MeSH headings assigned to the documents are related to each other.

For all four topics, we created frequency lists for MeSH headings assigned to the relevant documents. Next, we took all MeSH headings that occur in at least 10% of the relevant documents for a topic. This resulted in 19 to 27 MeSH headings per topic. For every heading on the list, we identified its relations with other headings based on the thesaurus.

Among all relations present in the thesaurus, we focus only on *direct* relations between two headings. These come in two kinds: hierarchical parent-child relations and cross-referential relations. Cross-references are relations to headings mentioned in the *Scope*, *Previous Indexing* or *See also* field of a descriptor's record.

For three of the four topics we find many direct relations (approximately 20) between the frequent MeSH headings for that topic. These direct relations are both hierarchical and cross-reference relations. The cross-reference relations are most often not bidirectional: a heading such as *DNA* is often referred to in the *Scope Note* or *Previous Indexing* field of other headings, but only has five *See Also* references itself. If we look at the information need of topic 2, for example, we find that it can be divided into three aspects: Certain proteins, DNA damage and the relation between these two. These three aspects can be seen in the relations between the MeSH headings. One group of seven related headings is focused on the type of proteins and genes involved (e.g., *DNA-binding proteins* and *Tumor suppressor genes*). Another small group of headings contains relations between DNA damage related headings (e.g., *Mutation* and *Apoptosis*). The last group of nine related headings is related to interactions and processes (e.g., *Gene expression regulation*, *Trans-activation* and *Genetic transcription*).

Topic 4, however, shows a different pattern than the other three topics: although most of its 19 main MeSH terms seem to be related, only a few direct relations can be found based on the thesaurus. For example, *Transforming Growth*

Factor Beta and *TGFB receptors* do not have a direct relation in the thesaurus. The same holds for *Head and Neck Neoplasms* and *Squamous Cell Carcinoma*. This can either mean that the thesaurus is not really consistent when it comes to cross-references or these relations are relatively 'new' or quite uncommon, and hence they do not appear in the thesaurus.

All four topics express a quite general information need that does not ask for very specific characteristics likely to be found in only one or a few articles. For these general topics, both the text of the relevant documents and as well as their assigned MeSH-headings have a sufficient level of specificity. This is confirmed when looking at the topics: if we compare frequently occurring words/phrases in the text with frequently occurring MeSH headings, we find a clear relation between text and MeSH headings for all four topics. Almost all frequently occurring nouns and compounds in the text are lexical variants or synonyms of one or more frequent MeSH headings. For example, *IREG1*, *ferroportin-1*, and *SLC11A3* all refer to the MeSH heading *metal transporting protein 1*.

For nouns or phrases that are instances of a heading that is on the Supplementary Concept Headings list, which are not used for indexing documents, we see something interesting. In most cases we find one of the frequently assigned headings in the *Heading Mapped to* or *Previous Indexing* field of the Supplementary Concept heading: for *MutY*, we find the two most frequent headings for topic 1, *DNA Glycosylases* and *N-Glycosyl Hydrolases*, in the two mentioned fields of the Supplementary Concept record of heading *MutY adenine glycosylase*.

The last issue to discuss here is the role of the MeSH headings marked as main topic of a document. For topic 1 there are two MeSH headings (*DNA Glycosylases* and *DNA Repair*) that occur in respectively 87 and 43 of the 115 documents as main topic. For the other three topics the main focus is less clear. If we look at the headings marked as main topic with respect to the relations with the other frequently occurring headings, we see that they are not necessarily headings that have many relations or headings that are at the 'center' of a group of related headings.

3.3 Queries and Thesaurus Terms

Besides examining the relation between the text and the MeSH headings, we can study the relations between the queries and the relevant documents. As said before, we use the title of the topics as our search query, assuming that most scientists will start their search process by entering just a few keywords. When we manually identify the MeSH headings that are most closely related to the keywords of our queries, we find that many of these headings are part of the Supplementary Concept Headings list. Note that the documents have only the preferred MeSH terms (i.e., descriptors) assigned to them. Non-preferred terms such as synonyms (i.e., non-descriptors) can be found in the Supplementary Concepts list. Again we find that in most cases the *Heading Mapped to* and the *Previous Indexing* field refer to MeSH headings frequently used to index the relevant documents. All other content-bearing keywords used in the four queries are instances of MeSH headings that occur frequently in the relevant documents.

3.4 Summary of Our Observations

We conclude this section by summarizing our main observations:

- Low or average retrieval scores are likely to be caused by vocabulary mismatch problems between the query and the relevant documents. This vocabulary gap is often caused by using low frequent synonyms or related terms as keywords.
- MeSH headings that are frequently assigned to the relevant documents of a topic are likely to be directly related to each other in the thesaurus; these relations can either be hierarchical or cross-referential.
- MeSH headings tend to have the same specificity as the frequently occurring words/phrases in the titles and abstracts of the relevant documents.
- Query keywords and frequently occurring words/phrases in the title and abstract of relevant documents can often be mapped to headings on the Supplementary Concept Headings list. The *Heading Mapped to* and the *Previous Indexing* fields on the record of these headings often refer to MeSH headings that occur frequently in the relevant documents.
- MeSH headings frequently marked as main topic that are assigned to the relevant documents do not necessarily play a central role in the information need of the topic.

These observations suggest that the semantic knowledge provided by a thesaurus can be useful for biomedical retrieval in two ways: its lexical information can be used as a controlled vocabulary to overcome problems with synonymy and lexical variance, and its relational knowledge is potentially useful for identifying relevant related terms.

4 Retrieval Experiments

In the previous section we provided a detailed comparison of queries, textual content and MeSH headings assigned to the relevant documents of our four TREC Genomics topics. In this section we take a closer look at the potential of a thesaurus for biomedical retrieval; to this end, we carry out a number of retrieval experiments. Besides comparing precision and recall scores averaged over all 50 TREC Genomics 2004 topics, we zoom in on the four selected topics to gain further insight in the retrieval features that cause retrieval (in)effectiveness.

4.1 Baseline Results

Our baseline run is based on the TREC Genomics 2004 ad hoc task. We use the topic titles as queries and the title, abstract and MeSH-heading fields as retrieval fields. Documents and queries are not stemmed, but stop-words are removed. Our vector space-based retrieval system achieves a mean average precision (MAP) of 0.1716 measured over a maximum of 1000 retrieved documents per topic. Of the total of 8268 relevant documents, 2762 were retrieved.

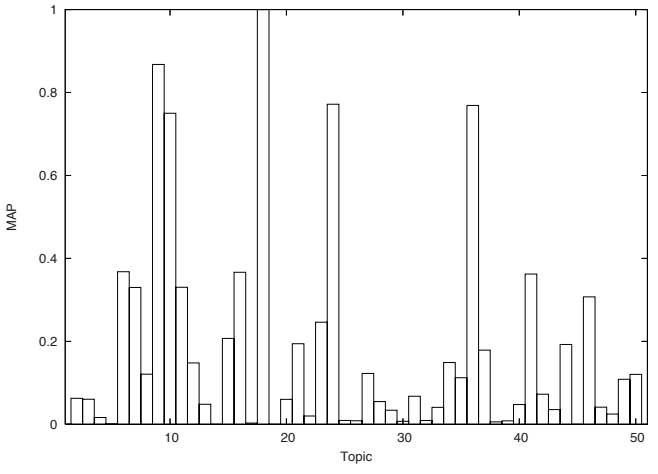


Fig. 3. Average precision per topic for the baseline run

Figure 3 shows the average precision per topic for our baseline run. As can be seen, there is a huge variety in average precision per topic. This same variety can be found in the average scores per topic of the TREC Genomics participants [8].

4.2 Thesaurus-Based Experiments

In our experiments, we will try to reverse engineer the role a thesaurus can play to improve retrieval using a natural language query. Recall from our analysis above that the relevant documents typically have closely related MeSH terms assigned to them. Hence, these MeSH terms provide useful retrieval cues. So if we select MeSH terms frequently assigned to relevant documents, we can use them to improve retrieval effectiveness.

In reality the set of relevant documents is unknown. So how should we select the relevant MeSH terms? We could ask the user to select the relevant documents in the set of initially retrieved documents. That is, we could use relevance feedback to obtain a set of relevant documents and, again, select the most frequently assigned MeSH terms. Finally, since relevance feedback still requires interaction with a user, we could simply assume that the first few, initially retrieved documents are relevant. That is, we could use pseudo-relevance feedback to obtain a set of pseudo-relevant documents and, again, select the most frequently assigned MeSH terms.

Methods. We carried out several feedback experiments. Based on the output of the baseline run, we used the following documents as input for the feedback algorithm:

1. First 10 retrieved documents (this amounts to blind feedback)
2. All relevant documents within the first 10 documents

Table 2. Retrieval results based on a maximum of 1000 retrieved documents

Run	MAP	Precision@30	Recall@1000
0 Baseline run	0.1716	0.3220	2762/8268
1 Baseline + blind feedback (top 10)	0.1801	0.3127	2896/8268
2 Baseline + relevance feedback (top 10)	0.1876	0.3267	2888/8268
3 Baseline + relevance feedback (top 100)	0.1996	0.3667	2933/8268
4 Baseline + 10 relevant docs	0.2011	0.3453	2971/8268

3. Relevant documents within the first 100 documents, with a maximum of 10
4. 10 random chosen relevant documents

In a real life retrieval situation, information about relevance can be provided by real users: they can be asked to judge initial retrieval results on their relevance, and based on this selection a new retrieval run can be done. Since we do not have access to real users to give this feedback, we simulate them by using the gold-standard data of the TREC Genomics ad hoc task for feedback.

Our first retrieval method does not involve this feedback and works with completely blind feedback. For the second and third method the first 10 and 100 retrieved documents were compared with the gold-standard data. The last method is completely artificial: to be able to get an idea of the potential of thesaurus-based relevance feedback, we choose ten relevant documents from the gold-standard collection.

For all document sets selected for feedback, we created frequency lists for MeSH headings assigned to the documents. Experiments showed that selecting the 35 headings that are most frequently assigned to the selection was optimal for feedback purposes.[1]

The lists of selected MeSH headings were used as queries for a new retrieval run on the MeSH heading fields of the collection. Combining this run with the baseline run resulted in a new ranked list of retrieved documents. Experiments showed that using the CombMNZ method [5] for combining both runs with a relative weight of 0.9 on the baseline run gives the best results. Evaluation of the results is based on the TREC Genomics 2004 ad hoc task gold-standard data.

Results. Table 2 shows the results of our feedback experiments. All four feedback runs show a significant improvement in MAP compared to the baseline run.[2] The most important reason for this improvement is the large increase in recall: for all four feedback runs, over a 100 more relevant documents are retrieved. Although runs 2 and 3 were set to a maximum of 10 feedback documents, the average number of documents was respectively 3.86 and 7. For run 2, no relevant feedback documents were found for 10 topics.

[1] MeSH headings occurring very frequently (more than 100,000 times) or infrequently (less than 5 times) in the total collection were not taken into consideration.

[2] For all runs, significance was proved with over 98% confidence. To determine statistical significance we used the bootstrapping method, a non-parametric inference test that has previously been applied to retrieval evaluation by, e.g., Wilkinson [24].

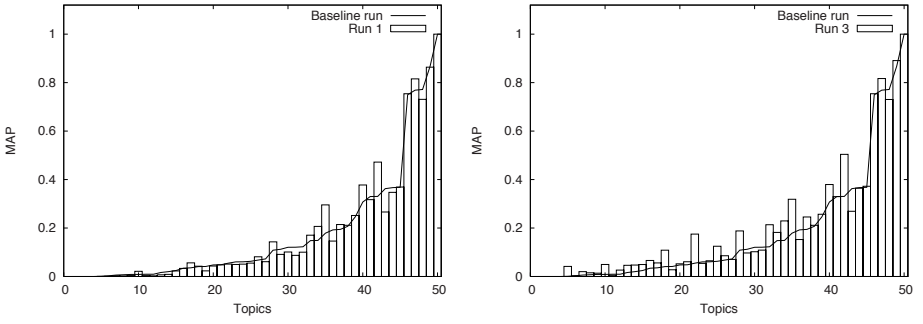


Fig. 4. (Left): Average precision scores for the baseline run vs. run 1. (Right): Average precision scores for the baseline run vs. run 3. In both plots, topics are ordered by increasing average precision score, not by topic ID.

In run 3 the top 100 retrieved documents could be used for feedback; however, for 19 of the 50 topics the 10 relevant documents were found in the top 30. For 6 topics, no relevant documents were found. The degree of improvement seems to be strongly related to the quality of the feedback documents: the more relevant documents are used, the better the MAP score is.

We compared the average precision score per topic for the baseline run, run 1 and run 3 (Figure 4(Left) and (Right), respectively). For both non-baseline runs, we see a big variety in the change of average precision scores. Although the MAP score increases when blind feedback is used (run 1), the graph shows that for a majority of the topics average precision does not improve (Figure 4(Left)). In run 3, when relevant documents are used for feedback, MAP increases for most topics. Especially topics with low initial MAP scores benefit. This is likely due to the fact that for most of these topics, most documents used for feedback in run 1 were irrelevant, whereas in run 3 only relevant documents, if available, are used.

In general we can conclude that thesaurus-based feedback improves mean average precision and especially recall. In real retrieval scenarios, asking a user for feedback is needed to identify relevant documents. While this may seem a time consuming job, for a scientist searching for all available literature on a certain topic, i.e., interested in boosting recall, this might be a good investment.

Case Studies. For two of the four selected topics, topic 3 and 4, no relevant documents can be found in the top 100 results of our baseline run. As a consequence, their retrieval scores do not improve by the first 3 feedback methods. For poorly performing topics, feedback is only useful if a user takes the time to find some relevant documents to use in the feedback procedure. Nevertheless, we hope to be able to define a method for the automatic assignment of MeSH headings to queries (such as [6] or [7]) in future research.

A comparison of the effectiveness of the feedback methods for the other two topics can be found in Table 3. For topic 2, recall is improved for all feedback runs. However, for this topic blind feedback (run 1) works better than the other

two relevance feedback runs. Topic 1, which already has a very good retrieval score, is not hurt by adding feedback to the retrieval process. To conclude our feedback analysis, we take a closer look at the results of topic 2 ("Role of p63 and p73 in relation to DNA Damage"). We see that for this topic, feedback does improve MAP and recall, but that the different feedback approaches do not show big differences in scores.

Let us take a look at MeSH headings used for the feedback runs. When we compare the 35 feedback headings for every run with the 10 most frequently occurring MeSH headings of the gold-standard relevant documents, we find that there is an overlap of at least 6 (see Table 4). Only run 4, whose feedback headings are created based on gold-standard data only, shows less overlap. However, this has no effect on the retrieval results, since MAP and recall stay relatively stable for all feedback runs.

When comparing the 35 feedback headings with the list of MeSH headings that occur in at least 10% of all relevant documents (24 headings in total), we find an overlap of 11 for run 1, 15 for run 2, 11 for run 3, and 7 for run 4. For all four runs, most other MeSH headings on the feedback lists are closely related to the 24 frequent headings of the relevant documents. For many of these headings a direct relation, either hierarchical or cross-referential, can be found in the thesaurus.

Hence, this suggests that using a larger number of relevant MeSH headings for feedback does not necessarily improve retrieval. Yet all improvements are obtained by feedback lists containing at least six MeSH headings frequently

Table 3. Retrieval scores for topic 1 and 2

Topic	Measure	**Baseline run**	**Run 1**	**Run 2**	**Run 3**	**Run 4**
1	MAP	0.8676	0.8638	0.8908	0.8910	0.8989
	Recall	113/115	113/115	113/115	113/115	115/115
2	MAP	0.1944	0.2145	0.2209	0.2255	0.2302
	Recall	40/80	45/80	44/80	44/80	44/80

Table 4. Occurrence of top 10 MeSH headings of relevant documents in feedback heading lists

MeSH headings	**Run 1**	**Run 2**	**Run 3**	**Run 4**
DNA-Binding Proteins	X	X	X	X
Nuclear Proteins	X	X	X	X
Apoptosis	X	X	X	
Protein p53	X	X	X	X
DNA Damage	X	X	X	
Phosphoproteins	X	X	X	X
Trans-Activators	X	X	X	
Cultured Tumor Cells		X		
p53 genes	X	X	X	X
Tumor Suppressor Genes	X	X	X	X

occurring in the relevant documents. In future research, we will look deeper into our feedback mechanisms and feedback results to see what the optimal settings for thesaurus-based feedback are.

5 Conclusion and Discussion

We studied the role of a thesaurus in biomedical retrieval. The relative effectiveness of controlled and natural languages is one of the longest standing debates in information retrieval, dating back to the original Cranfield experiments [4]. In particular, the use of controlled vocabularies to better articulate natural language queries, usually through some form of query expansion, has received a great deal of attention. For example, for automatic query expansion with thesaurus terms, Srinivasan [20] reports moderate improvement, but the improvement is overshadowed by the improvement due to standard text-based blind feedback. Based on the manual assignment of controlled terms to natural language queries, Hersh et al. [9] report a drop in retrieval effectiveness for a wide range of query expansion methods. A recurring pattern in the literature is that expanding natural language queries with controlled terms pays off for some fraction of the queries, but is detrimental for a larger fraction of the queries.

In light of the inconclusive evidence in the literature, we opted for a somewhat different approach to the question of how to select controlled terms to be added to a natural language query. Traditionally, the selection is based on the topic statement and the goal is to select those terms that are topically relevant for the information need. Our hypothesis is that this selection process should also be based on the role that the controlled terms play in the retrieval process, i.e., whether they are good retrieval cues for the search engine. Hence, to better grasp how a thesaurus can help improve the retrieval process, we performed a detailed analysis of a number of queries. In particular, we tried to analyse vocabulary mismatch problems, the related thesaurus terms, and their influence on retrieval. Our detailed analysis of four retrieval topics showed that vocabulary mismatch problems between queries and relevant documents have a negative impact on retrieval effectiveness. That is, there is a range of queries for which the natural language statement fails to be effective. In our thesaurus-based experiments, we found that using thesaurus terms for blind and relevance feedback can improve precision as well as recall. In general, the improvements increase with the amount of true relevance feedback provided to the system. However, the fully automatic runs using only pseudo-relevance feedback also led to improved retrieval effectiveness. The inherent shortcoming of feedback-based techniques, as highlighted by our success/failure analysis, is the failure to improve topics for which no relevant document is initially retrieved. To minimize the detrimental effect of query expansion on the fraction of queries for which the natural language query is effective, we use a combination of runs based on the original and on the expanded query.

Our results may contribute to a better understanding of the role of controlled vocabularies in information retrieval [21]. Our study is still limited and we plan

to extend it in a number of ways. First, a similar analysis should be performed for a larger set of queries. Second, we plan to experiment with other methods of selecting thesaurus terms based on initially retrieved documents. Third, we want to study different ways of incorporating controlled terms in the retrieval model, and the relation to models of text-based blind feedback. Fourth, we plan to analyse the intrinsic properties of the MeSH thesaurus, including its completeness, coherence, and consistency, and test the robustness of our approaches against imperfect resources. A better understanding of the effectiveness of thesaurus terms as retrieval cues is crucial for the selection of controlled terms. This may also influence our view of the goal of thesaurus-based expansion in the first place. If the natural language queries provide excellent retrieval cues for a large fraction of the queries, we can only hope to improve when the original query fails. That is, we could envision offering thesaurus-based query expansion as a query refinement option: in case a user is unsatisfied with the set of documents returned, she may choose to use the expanded query.

Acknowledgments. Leonie IJzereef's work was carried out in the context of the Virtual Laboratory for e-Science project (`www.vl-e.nl`). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ). Jaap Kamps was supported by a grant from the Netherlands Organization for Scientific Research (NWO) under project numbers 612.066.302 and 640.001.501. Maarten de Rijke was supported by grants from NWO, under project numbers 017.001.190, 220-80-001, 264-70-050, 365-20-005, 612.000.106, 612.000.207, 612.069.006, and 612.066.302.

References

[1] W. Ceusters, B. Smith, and L. Goldberg. A terminological and ontological analysis of the NCI thesaurus. *Methods of Information in Medicine*, 2005, in press.

[2] W. Ceusters, B. Smith, A. Kuman, and C. Dhaen. Mistakes in medical ontologies: Where do they come from and how can they be detected? In *Ontologies in Medicine: Proceedings of the Workshop on Medical Ontologies*. IOS Press, Amsterdam, 2003.

[3] C. W. Cleverdon. Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems. Technical report, College of Aeronautics, Cranfield UK, 1962.

[4] C. W. Cleverdon. The Cranfield tests on index language devices. *Aslib*, 19:173–192, 1967.

[5] E. A. Fox and J. A. Shaw. Combination of multiple searches. In D. K. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 243–252. National Institute for Standards and Technology. NIST Special Publication 500-215, 1994.

[6] J. C. French, A. L. Powell, F. Gey, and N. Perelman. Exploiting a controlled vocabulary to improve collection selection and retrieval effectiveness. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 199–206, New York, NY, USA, 2001. ACM Press.

[7] N. Grabar, P. Zweigenbaum, L. Soualmia, and S. Darmoni. Matching controlled vocabulary words. In G. Surjan, R. Engelbrecht, and P. McNair, editors, *Proceedings of MIE 2003, Eighteenth International Congress of the European Federation for Medical Informatics*. IOS Press Publisher, 2003.

[8] W. Hersh, R. T. Bhuptiraju, L. Ross, P. Johnson, A. Cohen, and D. Kraemer. Trec 2004 genomics track overview. In *The Thirteenth Text Retrieval Conference: TREC 2004*, Gaithersburg, MD, 2004. National Institute of Standards and Technology.

[9] W. Hersh, S. Price, and L. Donohoe. Assessing thesaurus-based query expansion using the UMLS metathesaurus. In *Proc. of the 2000 American Medical Informatics Association (AMIA) Symposium*, pages 344–348, 2000.

[10] M. Iivonen. Consistency in the selection of search concepts and search terms. *Information Processing and Management*, 31:173–190, 1995.

[11] J. Kamps. Improving retrieval effectiveness by reranking documents based on controlled vocabulary. In S. McDonald and J. Tait, editors, *Advances in Information Retrieval: 26th European Conference on IR Research (ECIR 2004)*, volume 2997 of *Lecture Notes in Computer Science*, pages 283–295. Springer-Verlag, Heidelberg, 2004.

[12] W. Kraaij, M. Weeber, S. Raaijmakers, and R. Jelier. MeSH based feedback, concept recognition and stacked classification for curation tasks. In *Proceedings of TREC 2004*. NIST, 2005.

[13] F. Lancaster. *Vocabulary Control for Information Retrieval*. Information Resources Press, Arlington, Virginia, second edition, 1986.

[14] National Library of Medicine. Medical Literature Analysis and Retrieval System Online (MEDLINE). `http://www.nlm.nih.gov/pubs/factsheets/medline.html`, May 2005.

[15] National Library of Medicine. Medical Subject Headings (MeSH). `http://www.nlm.nih.gov/mesh/`, May 2005.

[16] National Library of Medicine. Unified Medical Language System (UMLS). `http://www.nlm.nih.gov/pubs/factsheets/umlsmeta.html`, May 2005.

[17] J. Paralic and I. Kostial. Ontology-based information retrieval. In *Proceedings of the 14th Int. Conference on Information and Intelligent Systems - iis2003*, pages 23–28, 2003.

[18] T. Saracevic and P. B. Kantor. A study of information seeking and retrieving. III. searchers, searches, overlap. *Journal of the American Society for Information Science and Technology*, 39:197–216, 1988.

[19] J. Savoy. Bibliographic database access using free-text and controlled vocabulary: an evaluation. *Information Processing and Management*, 41:873–890, 2005.

[20] P. Srinivasan. Query expansion and MEDLINE. *Information Processing and Management*, 32(4):431–443, 1996.

[21] E. Svenonius. Unanswered questions in the design of controlled vocabularies. *Journals of the American Society for Information Science*, 37:331–340, 1986.

[22] TREC Genomics Track. TREC Genomics Track. `http://ir.ohsu.edu/genomics/`, May 2005.

[23] E. M. Voorhees. Using WordNet to disambiguate word senses for text retrieval. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 171–180, New York, NY, USA, 1993. ACM Press.

[24] J. Wilbur. Non-parametric significance tests of retrieval performance comparisons. *Journal of Information Science*, 20:270–284, 1994.

Hybrid Model for Semantic Similarity Measurement

Angela Schwering

Ordnance Survey of Great Britain, United Kingdom, Institute for Geoinformatics, University of Muenster, Germany
angela.schwering@uni-muenster.de

Abstract. Expressive knowledge representations with flexible semantic similarity measures are central for the functioning of semantic information retrieval, information integration, matchmaking etc. Existing knowledge representations provide no or not sufficient support to model the scope of properties. While properties in feature- and geometric models always refer to the whole concept, structured representations such as the alignment model provide a limited support for scope by assigning properties to objects which are part of the whole entity. Network models do not support properties at all. In this paper we propose a hybrid model: a structured knowledge representation combining the relational structure of semantic nets with property-based description of feature- or geometric models. It supports to model properties—features or dimensions—and their scope by taxonomic or non-taxonomic relations between a concept and its properties. The similarity measure computes the similarity in consideration of the scope of each property.

1 Introduction

In this paper we aim to develop a measure to assess the semantic similarity between concepts. According to E. Smith, a concept is "a mental representation of a class or individual and deals with what is being represented and how that information is typically used during the categorization" (Smith 1989, p. 502). To assess similarity between concepts with formal measures, these "mental" concepts must be represented in a computer-readable form: we investigate different knowledge representations based on features, dimensions and relations and propose a new, hybrid model for structured representation of concepts.

The knowledge representations under investigation in this paper all have a cognitive foundation. They are contrary to formal ontologies used in the semantic web: These formal ontologies are based on first order logic and define semantics by stating necessary and sufficient conditions for something to be an instance of a class. Individuals and classes of formal ontologies are processed by syllogistic reasoning (see also (Gärdenfors 1999; 2004) for detailed discussion). These ontologies do not reflect very well the way humans form and process concepts and cognize the world. Here, we investigate knowledge representations which arose from theories about human knowledge structure and human similarity judgment.

Structured knowledge representations are essential for similarity measurement, because they "make explicit the relations between elements in a situation, and allow

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1449 – 1465, 2005.

complex representation to be constructed through the combination of simpler elements" (Markman 1999, p. 124). Reviewing conventional knowledge representations we conclude that they provide no or not sufficient structure to represent the complexity of a concept's semantic, neither include this information in the similarity measure. To specify concepts we require a model that is able to represent relations between concepts, between its elements and the scope of these elements.

The scope of representational elements such as properties is what a property refers to. While some refer to the whole concept, others describe only parts or some specific aspect of the concept. This scope must be reflected not only in the knowledge representation model, but also included in the similarity measure.

Section 2 gives an overview of existing knowledge representations and their similarity measures and evaluates them regarding to their ability to represent relations and scope. In section 3 we define different types of scope and explain how different elements of the hybrid model are combined to represent scope of properties. Section 4 describes the similarity measure and section 5 evaluates the approach in a case study and discusses the results. The final section provides conclusions and directions for future work.

2 Related Work

This section gives an overview of different knowledge representations and their similarity measures.

2.1 Feature Model

The most prominent feature models are Tversky's contrast and ratio model (Tversky 1977; Tversky and Gati 1978; 1982; Sattath and Tversky 1987). Objects (or stimuli)[1] are represented via an unstructured list of features which may correspond to components, concrete or abstract properties of the object. A sunflower may be described by the features 'stalk', 'bloom' and 'root' reflecting the components, 'yellow' for the colour of the bloom and 'tall' for its height.

The contrast model (equation 1) and the ratio model (equation 2) are set theoretic models assessing the similarity of two objects via their ratio of common ($A \cap B$) and distinct features ($A - B$ or $B - A$). While common features increase the similarity, distinct features decrease it. This similarity measure underlies the assumption that similarity is asymmetric and neither minimality nor triangle inequality hold (obtained by parameters α and β in the formulas).

$$S(a,b) = \theta * f(A \cap B) + \alpha * f(A - B) + \beta * f(B - A) \tag{1}$$

$$S(a,b) = \frac{f(A \cap B)}{f(A \cap B) + \alpha * f(A - B) + \beta * f(B - A)} \tag{2}$$

[1] Tversky applies the feature model only to objects or stimuli. Rodriguez (Rodriguez 2000; Rodriguez and Egenhofer 2003, 2004) used Tversky's feature model to measure similarity between entity classes which are concepts about the real world (see also footnote 4).

As illustrated in the example before features refer always to whole object. Although the colour 'yellow' only describes the bloom of the flower, it is assigned to the whole object. The scope of features can only be represented by combining the feature and its scope in one compound feature such as 'yellowBloom'. Compound features are themselves features which either match features of the other object or not. Due to the large set of possible compound features ('yellowBloom' or 'yellowBloomLeaves' etc.) these kinds of features do not lead to good measurements. Feature models assess only entire feature matches and no partial match is detected between 'yellowBloom' and 'yellowBloomLeaves'.

Another disadvantage is the inability of feature models to relate two objects in a structured way, e.g. the fact that a bloom is connected to the stalk of a flower can again only be expressed by the compound feature 'bloomConnectedToStalk'. No partial match would be detected between 'bloomConnectedToStalk' and 'leaveConnectedToStalk'. A relational representation of these features—e.g. connectedTo(bloom,stalk) and connectedTo(leave,stalk)—could detect similarity via an alignment of arguments.

2.2 Geometric Model

Geometric models use the notion of multidimensional vector spaces to represent objects and concepts. Gärdenfors introduced conceptual spaces, a geometric model for representing information at a conceptual level (Gärdenfors 2000). Conceptual spaces are a set of metric quality dimensions where one single or several dimensions form one domain[2]. All dimensions belonging to one domain are integral. In analogy to features, each domain in the conceptual space represents one property. Concepts[3] are represented by an n-dimensional region in the vector space and objects by a point.

Geometric similarity models measure the semantic distance in analogy to the spatial distance: The most prominent distance measure in vector spaces is the Minkowski metric (equation 3). It is a generic formula: For r = 1 the result is the city-block distance and for r = 2 it is the Euclidian distance (Suppes, Krantz et al. 1989).

$$d(q,c) = \left[\sum_{i=1}^{n} |q_i - c_i|^r \right]^{1/r} \tag{3}$$

The similarity is a linear decaying function of distance *d(q,c)* (Attneave 1950; Melara, Marks et al. 1992) and the absolute identification confusability—often taken as

[2] We use the word dimension to refer to the mathematical structure. The values for a property are specified in a domain (modelled by a dimension).

[3] Gärdenfors defines concepts by comparing them to properties: a property "is defined with the aid of a single dimension or a small number or integral dimensions forming one domain" (Gärdenfors 2000, p. 60). A concept is a more general case of a property and is defined by a set of dimensions or domains. Therefore a concept is represented by a region in a conceptual space. His notion of concepts is based on the definition by Sloman, Love, and Ahn: A "concept is an idea that characterizes a set or category of objects" (Gärdenfors 2000, p. 60). Objects, in the philosophical sense also called individuals, are specified by a set of internally consistent properties. One object cannot have two disjoint properties. An object is represented by a point in the conceptual space (Gärdenfors 2002).

indirect measure for similarity—is an exponentially decaying function of distance *d(q,c)* (Shepard 1957; Shepard 1958 a; Shepard 1958 b).

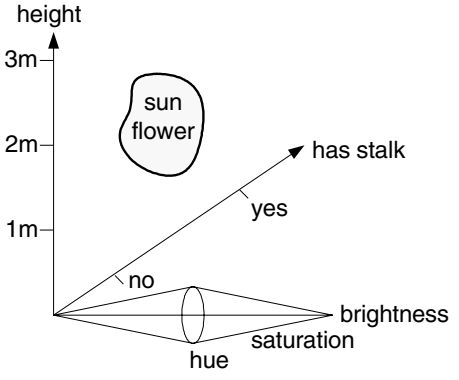


Fig. 1. A geometric model represents concepts in vector spaces. The concept 'sunflower' is modelled in a conceptual space with the domains 'height', 'has stalk' and 'colour'.

Like features, it is not possible to specify the scope of domains in a conceptual space. They always refer to the whole concept: The whole concept 'sunflower' is represented as yellow, although only its bloom is yellow. Properties describing the components of the concept can be represented by Boolean dimensions such as the 'has stalk' dimension, but the components themselves cannot be further described.

Conceptual spaces as introduced by Gärdenfors can represent relationships between concepts only as "compound dimensions", which resemble the notion of compound features in the feature model. Schwering and Raubal (Schwering and Raubal 2005 forthcoming) proposed an extension to conceptual spaces to model relations as dimensions to overcome this disadvantage. However, these "compound dimensions" suffer of the same problem that the relational structure is lost and cannot be used for similarity judgement.

2.3 Network Model

Network models are built upon semantic nets. Rodriguez and Egenhofer (Rodríguez 2000; 2003) proposed a semantic similarity measure between concepts[4] based on semantic neighbourhoods. Subclass-superclass relations and partOf relations between concepts are represented by directed arcs in a semantic net. A semantic neighbourhood of a concept a^o is defined as the set of concepts $\{c_i^o\}$ whose distances to a^o is equal or less than *r* which is the radius of the neighbourhood. The distance between concepts represented by nodes is the length of the shortest path between two

[4] Rodriguez uses the notion of entity classes which are concepts about the real world in the way Dahlgren defines concepts: "These concepts about the real world are cognitive representations that people use to recognize and categorize entities or events in the real world." (Rodriguez 2000, p. 4).

nodes in a semantic net. Rodriguez does not consider the direction of arcs neither distinguishes between types of relations. Equation 4 (Rodríguez 2000) gives the formal definition of a neighbourhood N.

$$N(a^o, r) = \{c_i^o\} \forall i \; d(a^0, c_i^o) \leq r \tag{4}$$

Rodriguez uses the distances in the semantic net only for defining the neighbourhood. Once a neighbourhood is set, the similarity is computed based on word matching or a slightly modified version of Tversky's feature matching (for further information see (Rodríguez and Egenhofer 2003)).

Other approaches compute the semantic similarity based on the shortest path in a network. Rada et. al propose a function to compute the distance between concepts in a semantic net for similarity measurement based on spreading activation (Rada, Mili et al. 1989).

The representation of relations between concepts is the strength of the network approach. Rodriguez restricts relationships to hierarchic and partonomic relations, but it can be extended by other non-taxonomic relations such as spatial relations (Schwering 2004). Network models do not describe concepts any further (e.g. by features or dimensions) though for the similarity measurement it can be combined with the feature model like Rodriguez did.

The main disadvantage of network models is their failure of modelling scope. Pure network models do not include properties at all; Rodriguez allows for features, but does not include their scope in the similarity measurement. Like Tversky, she simply compares sets of features.

2.4 Alignment Model

While the feature model, the geometric model and the network model are unstructured knowledge representations, alignment models are the first to represent knowledge in a structured way by adopting the structural alignment framework (Gentner and Markman 1997). Goldstone (Goldstone 1994; Goldstone and Medin 1994; Goldstone and Son 2004) introduced the Similarity, Interactive Activation, and Mapping (SIAM) model, an alignment model for similarity measurement. It is used for similarity measurement of spatial scenes, though the notion of structural alignment can be applied to concept similarity measurement as well.

The entities—spatial scenes—are described by roles, objects and features: Roles are two-ary relations with objects as arguments, which themselves contain feature slots filled in with particular values. Figure 2 shows a spatial scene and a description of this scene as it could be found in SIAM. Relations describe hierarchical (partOf and isA relations) or propositional representations of objects, such as the spatial relation in the example below (Goldstone 1994).

The similarity measurement is two-step process: at first the correspondences between features, objects and roles are analyzed. All correspondences must be structural consistent which involves two constraints: one-to-one mapping and parallel connectivity (Markman 1999). One-to-one mapping states that each element—feature, object or role—is mapped to exactly one element of the compared entity. Mappings to multiple elements inhibit themselves mutually. Parallel connectivity means that the arguments of corresponding roles must also correspond. The second step of the

similarity measurement is the interactive process to compute the similarity between spatial scenes: aligned matches increase the similarity more than non-aligned matches (for a detailed description of the interactive process see (Goldstone 1994; Goldstone and Medin 1994)).

Representation:

above and left [
(Sunflower: bloom yellow, height tall),
(Rose: bloom red, height medium)]

Fig. 2. The alignment model proposed by Goldstone describes spatial scenes in a structured way with roles, objects and features

SIAM structures scenes into components called objects which are described by the roles they play and the features they have. Since this similarity model measures the perceived similarity by humans, objects are limited to the scene's component and features are limited to perceptual features only. By assigning each feature to one object of the scene, SIAM enables to model the scope of features in the sense that a feature belongs only to one part of the scene (compare different types of scope in section 3). It is not possible to state that a feature refers to one specific aspect of the object, e.g. it is not possible to describe the *smell* of the rose by features 'strong' and 'sweet'. In alignment models both features are assigned directly to the object rose.

One constraint for setting the correspondences is the one-to-one mapping of elements. With spatial scenes having more or less an identical structure (same objects with corresponding features) it is possible to align analogous objects and features, but this does not work for concept similarity: two properties describing one concept may align to one, more generic property describing another concept. A more flexible model allowing for property hierarchies and n:m matches is required to compute concept similarity.

Features in SIAM are atomic and cannot be described any further. There exists no partial match between features (Feature 'bloom yellow' is as dissimilar to the feature 'bloom orange' as to 'bloom black'. Modelling properties as dimensions enables measuring degree of similarity between properties.).

3 Hybrid Model for Structured Knowledge Representation

The hybrid model for structured knowledge representation includes properties (features or dimensions) such as the feature model and the geometric model, but as well relations such as the network model. The main advantages of its flexible

architecture are the ability to explicitly model the scope of properties. This section explains different types of scope (section 3.1) and introduces the elements and structure of the hybrid model (section 3.2).

3.1 Scope of Properties

Structured knowledge representations provide mechanisms to specify explicitly the scope of properties—the perspective or the aspect the property focuses on. In natural language we express the scope of properties for example by genitive constructs (green leaves of a rose) or participles presence (strong smelling flowers), but also by whole sentences (ivy is located in shadowy areas). Sometimes also the scope is not expressed explicitly, e.g. in 'red roses' the property red refers to the rose's bloom, but not to the whole rose).

We distinguish four types of scope:

- Scope 1: The property refers to the whole concept. In this case the property is assigned to the concept just as it is done in the feature- or the geometric model: For example the feature 'tall' of a concept 'sunflower' describes the whole sunflower and is therefore one feature of the set of features describing sunflowers. In geometric models the property 'height' with values ranging form 100 to 250 cm is used to describe the concept 'sunflower'. As property referring to the whole concept it is modelled as one domain of the conceptual space describing 'sunflower'. No relational structure is needed to model scope type 1.
- Scope 2: The property refers to a part of a concept. In this case the concept and its part are modelled as two separate concepts related via a partOf relation. The property is assigned to the part and can be included in the similarity measurement via the relation to the main concept.

  For example the feature 'red' describing a rose's bloom would not be part of the feature set describing roses. Instead, two feature sets—one describing the concept 'rose' and one describing 'rose's bloom'—are created. The feature 'red' is part of the 'rose's bloom' feature set. The same goes for geometric models: Two separate conceptual spaces are created. The colour domain of the 'rose's bloom' conceptual space has the value 'red'. Both feature sets or both conceptual spaces are related via a partOf relation.
- Scope 3: The property refers to a concept which is a superconcept S of the considered concept. In this case the super- and the subconcept are modelled as two separate concepts related via an isA relation. The property is assigned to concept S, but can be included in the similarity measurement via the relation to the subconcept. The property 'photosynthesis' is not describing the metabolism of flowering plants in particular, but is a property that all organisms containing chlorophyll, i.e. all plants (except for some parasitic plants), have. This property is assigned to the feature set or conceptual space of the concept 'plant'.
- Scope 4: The property refers to a specific aspect, e.g. it describes the spatial environment where instances of a concept are usually located in. In this case the aspect is modelled as a separate concept (e.g. location) and related to the original concept via a non-taxonomic relation (e.g. contained within). By this non-taxonomic relation any scope can be represented explicitly.

In the running example the property 'sunny' is assigned to the feature set or conceptual space describing the location, because it is not the sunflower which is sunny, but its preferred location.

The hybrid model uses relations to model the scope of properties. These relations are typically binary, but can be n-ary with $n \geq 2$.

3.2 Elements and Structure of the Hybrid Model

A hybrid model consists of concepts, properties and relations. Concepts are modelled as nodes in a semantic net. They are described by properties modelled as domains in a conceptual space or as features in a feature set. Each node contains a conceptual space respectively a feature set (figure 3). In the following explanations we focus only on concepts described in conceptual spaces, but the similarity measure can be applied to concepts described with feature sets as well.

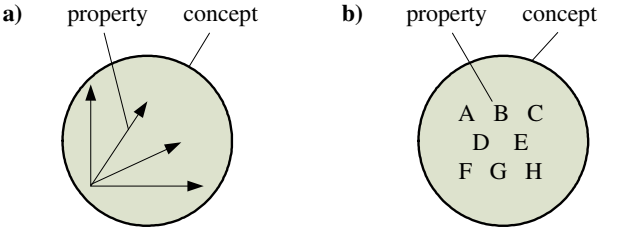


Fig. 3. Hybrid models describe properties of concepts either in a conceptual space (figure a) or with a set of features (figure b)

Relations between concepts are represented as directed arcs between nodes. If a property has a specific scope (which does not equal the whole concept like scope type 1) it is modelled as property of a separate concept with a relation specifying the scope (figure 4).

Properties describing a superconcept of a concept X are represented in the conceptual space of the superconcept (scope type 3). The hierarchic relation between X and its superconcept are used in the similarity measurement to identify inheritance of properties. Properties describing parts of concept X are represented as properties of concept 'part of X' (scope type 2), which is related to X with a partonomic relation. Properties referring to the location of X are assigned to concept 'location of X', which is related to X with a spatial relation such as contained within (scope type 4). Properties referring to the function of X are analogous assigned to a concept 'function of X'. All concepts themselves can be described by properties and again by relations to other concepts.

A hybrid model combines the idea of semantic nets with feature or dimensional approaches to model properties. This structured representation allows for modelling complex semantics and explicitly the scope of properties.

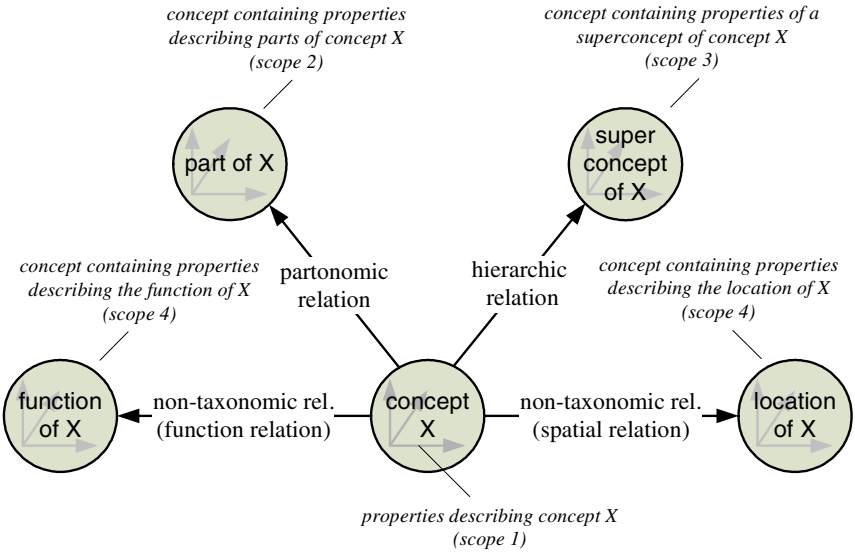


Fig. 4. The elements of a hybrid model are concepts and its properties, relations, the related concepts and their properties. Different types of relations are used to model different types of scope.

4 Semantic Similarity Measurement with a Hybrid Model

Similarity in hybrid models gets calculated in a two step process. At first all properties are identified: for properties modelled as related concepts (scope type 2-4) we compute separate similarity values and in the second step include these values in the overall similarity measure. In this section we outline the particular requirements of a similarity measurement for structured knowledge representation and explain the measurement steps in detail.

4.1 Requirements for the Similarity Measure

A semantic similarity measure of a structured knowledge representation must be sensitive to the argument structure of the representation. This similarity measure must be able to account for properties modelled as features or dimensions *and* properties modelled in related concepts. If properties are connected via relations between two concepts, the type of relation must be considered in the similarity measure:

- Properties belonging to concepts related with isA relations are inherited by the concept. If the concept already contains this property, it is not inherited, e.g. a concept 'flowering plant' is described by the property 'photosynthesis' and height with the values 0 to 300 cm. While the concept 'sunflower' being a subconcept of 'flowering plant' inherits the property 'photosynthesis', it contains more specific information on the height and retains its own height description.
- Properties of concepts related by partOf- or other non-taxonomic relations have to be considered separately. Their domains must not be included directly in the

conceptual space of concept C, because they describe only a part or a specific aspect of C. Parts or aspects of concept C should be compared to corresponding parts or aspects of the other concept Q.

It is not possible to include domains of the conceptual space of parts or aspects in the conceptual space of C, because there may exist domains with the same label in the conceptual space of C, but with different meaning due to their different scope, e.g. the domain colour describing the part 'bloom' of a rose has the value 'red' and the domain colour describing the part 'stalk' of a rose has the value 'green'.

4.2 The Similarity Measure

The similarity measure for hybrid models comprises several steps. The following detailed description refers to a dimensional representation of properties and an Euclidian measure. Other distance measures of conceptual spaces such as the city-block metric can be applied without any problems (as long as the measure is based on property-wise comparison of vectors). If properties are modelled as features, Tversky's feature model is used for similarity measurement.

In the explanation below we refer to a directed similarity measurement task as it occurs in the information retrieval: concept Q denotes the query concept. The similarity value indicates how similar a concept C is to the query concept Q.

$$d(q,c) = \sqrt{\sum_{i=1}^{n} \underbrace{(q_i - c_i)^2}_{\text{dimension-wise difference}}} \tag{5}$$

According to the Euclidian metric (equation 5 is equation 3 with r = 2) the distance between two vectors C and Q in a multidimensional space is computed as the square root of the sum of the dimension-wise squared differences. At first we describe how to compute the dimension-wise difference.

Figure 5 and 6 illustrates the steps of the similarity measurement:

1. Identify all common and corresponding properties of query concept Q and the compared concept C, as well those modelled as concept related to C and Q[5]. Step 1.a in figure 5 illustrates the alignment of domains in the conceptual space of C and Q. Inherited domains are treated as ordinary domains of a conceptual space. Steps 1.b.1 and 1.b.2 in figure 5 show the alignment of related concepts. In the example there are no correspondences for the properties D5 and D6 and for the relation 2. All additional properties and relations of concept C that the query concept Q does not contain—i.e. the user does not search for something having these additional properties or relations—are simply deleted and left out for the similarity measurement.
2. For each related concept C_{rel} and Q_{rel} we compute a semantic distance value analogous to the similarity measure of conceptual spaces, e.g. a semantic distance

[5] For the sake of simplicity we consider only same or comparable properties of C and Q. Properties that do not have a counterpart in the conceptual space of the other concept are not considered in the similarity measurement.

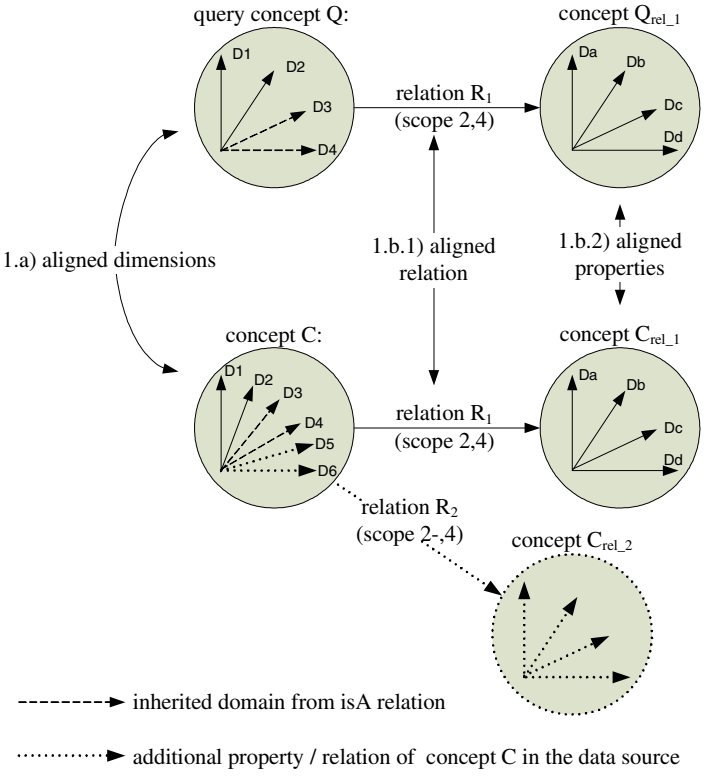


Fig. 5. Step 1 of the similarity measure between query concept Q and concept C

value for the part 'bloom' of a rose to the part 'bloom' of the query concept. The resulting distance value is the Euclidian distance between the bloom of C and the bloom of Q (step 2.a in figure 6).

In the conceptual space of C and Q we introduce a new temporary dimension for each related concept, e.g. a dimension 'partOfBloom' (step 2.b in figure 6). Since the related concept consists of a set of several dimensions we cannot assign a value on a single dimension to C or Q, but we do know the semantic distance of C_{rel} and Q_{rel}. This semantic distance is fed in the formula as shown in equation 6.

3. Compute the difference between C and Q for each domain belonging to their conceptual space (step 3 in figure 6).

The result is an n-dimensional vector (equation 6); n is the sum of the domains of the conceptual space of C and Q plus the domains of its related properties—in this example n = 5: four domains of C and Q and one related for the related concept. Each dimension of this vector represents the dimension-wise distance:

- either it is the difference between concept's C value on this dimension and the query concept's value on this dimension
- or it is the Euclidian distance between the related concept C_{rel} and the related concept Q_{rel}.

$$\begin{pmatrix} C_{D1} - Q_{D1} \\ C_{D2} - Q_{D2} \\ C_{D3} - Q_{D3} \\ C_{D4} - Q_{D4} \\ C_{rel} - Q_{rel} \end{pmatrix} \quad \begin{matrix} \} \text{ dimension-wise distance} \\ \} \text{ semantic distance} \end{matrix} \tag{6}$$

These values are fed into the Euclidian distance formula: each dimension is squared, summed up and the square-root is computed of the sum.

In the query concept each domain may be weighted by a weighting factor. By this can be specified the importance of each domain in the similarity measurement. These weighting factors are included in the similarity measure by a dimension-wise multiplication factor (Raubal 2004).

Step 2 of the similarity measure is the calculation of the Euclidian distance between related concepts. To determine the semantic distance of these related concepts, one can again include the related concepts. To determine this recursive

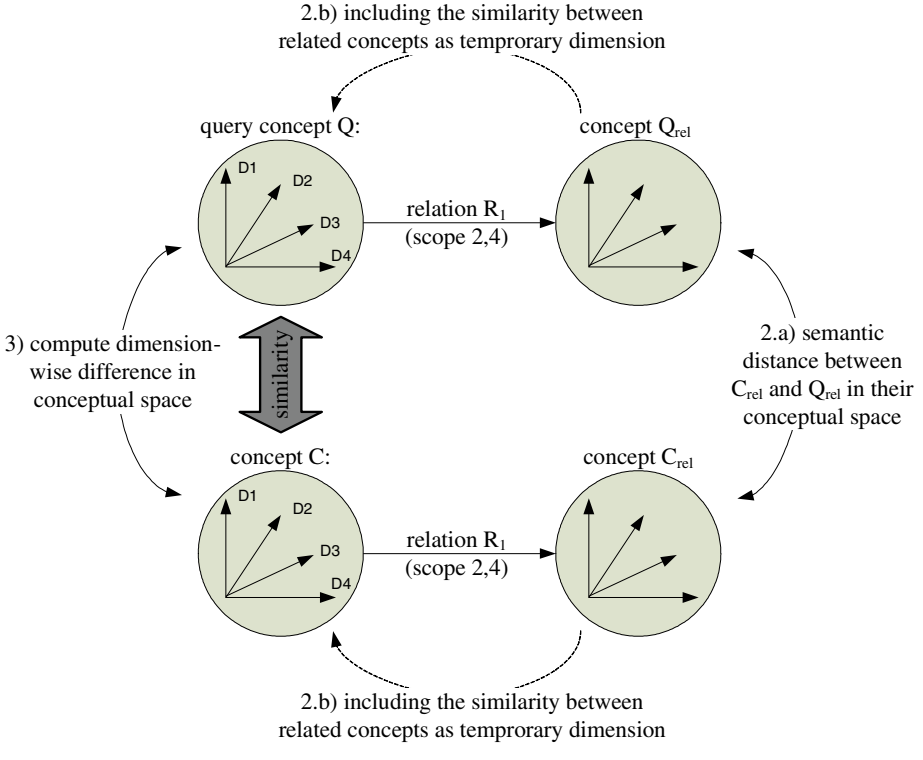


Fig. 6. Steps 2 and 3 of the similarity measure between query concept Q and concept C

process we need to decide on a neighbourhood size—analogous to the radius determining the neighbourhood size in the similarity approach by Rodriguez and Egenhofer (see also section 2.3).

5 Case Study and Evaluation

The case study shows how the hybrid model and its similarity measure can be applied to an information retrieval task. Then we evaluate advantages and disadvantages of this approach.

5.1 Case Study

The following case study is about an information retrieval task searching for appropriate plants to grow in one particular place. Rebecca is a landscape gardener planning the planting of a garden. She wants to replant a flower bed with plants that complement the existing planting. Since there should be plants blooming all over the year in different colours, Rebecca wants to choose a selection of plants with different properties concerning the season of anthesis, the colour of the bloom, the height etc. The flower bed is located in the sun.

Rebecca specifies her query in the information retrieval system by choosing the relevant properties (figure 7) and specifying their scope. In this example she describes the plant she is searching for by restricting the properties of its part bloom and the location where the plant should grow.

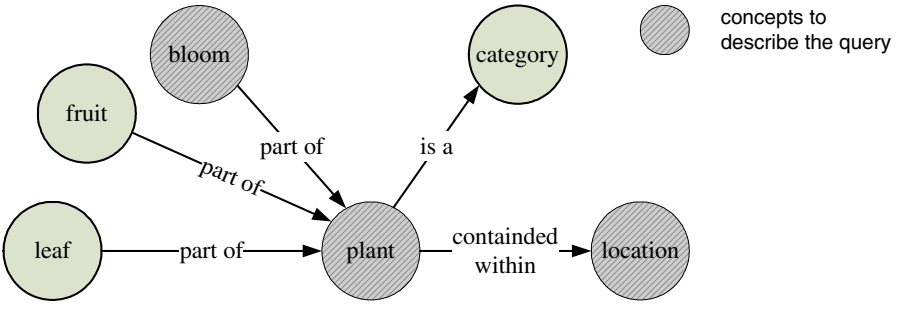
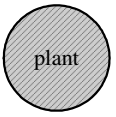


Fig. 7. The information retrieval system contains several concepts and properties to describe plants. To specify the query in the information retrieval system the user chooses the properties relevant to describe the plant she is searching for.

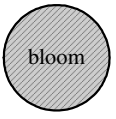
For each property—the related ones and the domains in the conceptual space of plant—she selects the relevant domains in the conceptual space provided by the information retrieval system and specifies the values on these domains. Figure 8 shows the existing domains and the specified values by Rebecca. She leaves out domains such as 'lifetime' of the plant or 'form' of the bloom, because they are irrelevant for her task.

Depending on the importance, Rebecca sets a weighting factor for each domain and each related property[6]. The weighting factors for the domains of 'bloom' and 'location' are included in the calculation of the semantic distance between the blooms and the locations (step 2.a in figure 6). The weighting factors for the whole related properties 'bloom', 'leaf' and 'location' and the weighting factors for the domains of 'plant' are included in the semantic distance calculation in step 3 in figure 6.



query concept:
height: 40-60 cm

conceptual space domains:
lifetime: values {0,1,2, ...} unit: year
height: values {0,1,2, … } unit: cm
form: values {narrow, … , broad} (grounding in cm² possible)



query concept:
colour: red
anthesis season:
April-June

conceptual space domains:
colour: dimensions hue, saturation, brightness
size: values {small, … , big} (grounding in cm³ possible)
form: values {flat, … , ball-shaped}
anthesis season: values {Jan, Feb, … , Dec}

query concept:
sunlight: sunny

conceptual space domains:
sunlight: values {umbra, pen-umbra, sunny, blazing sun}

Fig. 8. The user specifies the values of the query concept's dimensions and domains, which are a subset of the dimensions and domains that are provided by the information retrieval system

In order to calculate semantic distances between concepts it is required that all dimensions of the conceptual space are represented in the same relative unit of measurement. This is ensured by calculating the z scores for these values, also called z-transformation (for more information about the z-transformation please see (Devore and Peck 2001)).

Once specified the query concept, the information retrieval system measures the similarity between query concept and all concepts of plants in the system according to the above described similarity measure. Table 1 shows the description of a subset of flowers and their semantic distance to the query concept. The semantic distance values are transformed into a similarity value according to a linear decaying function of the semantic distance. The exact function must be determined by human subject testing. The complete list of flowers and their semantic distances can be found at http://ifgi.uni-muenster.de/~eidueidu/odbase05.zip.

The information retrieval system does not contain information about a flower that satisfies all restrictions of Rebecca. The most suitable flower is the Begonia: it fits the colour and anthesis restrictions, though it is slightly too little and more appropriate to pen-umbra than to sunny locations. Papaver orientale, achillea millefolium and centaurea dealbata match the height and the location requirements, but the anthesis season does not completely overlap with the query concept. The bloom colour of papaver orientale and the one of centaurea dealbata are similar to red. Rebecca can look at the ranking and chooses one of the best ranked flowers.

[6] To keep the calculation simple we assume a weighting factor of 1 for every domain.

Table 1. The semantic distance from all flower concepts to the query concept is computed. The most similar flowers are recommended to Rebecca.

Flower (short names)	Sem. Dist.	Flower: Height cm	Bloom: Colour	Bloom: Anthesis Season	Location: Sunlight
Begonia	1,37	20-35	red	Mar-Oct	pen-umbra
Papaver orientale	1,97	40-70	light red	May-Jun	sunny
Achillea millefolium	2,54	40-60	red	Jun-Sep	sunny
Centaurea dealbata	2,57	40-70	rose	Jun-Jul	sunny
Campanula glomerata	2,78	40-60	blue-violet	May-Jun	pen-umbra
Doronicum orientale	2,93	40-50	yellow	Apr-May	pen-umbra, umbra
Iris 'blue'	3,04	40-60	blue	May-Jun	sunny
Chrysanthemum (red)	3,11	90	red	Sep-Nov	sunny
Chrysanthemum (yel)	3,33	90	yellow	Sep-Nov	sunny
Helianthus Annuus	3,38	100-250	yellow	Jul-Oct	sunny
…	…	…	…	…	…
Malus sylvéstris	5,58	500-1000	reddish-white	Apr	pen-umbra, umbra
Pýrus commúnis	6,58	500-2000	white	Apr-May	pen-umbra, umbra

Malus sylvéstris (apple tree) and Pýrus commúnis (pear tree) have the highest semantic distance to the query concept and are therefore the least similar concepts. The system has classified them as "least suitable" to plant on a flower bed.

Comparing the hybrid model with unstructured models such as the feature model or the geometric models the main advantage is the higher accuracy due to greater expressiveness. Using non-structured knowledge representations the concept 'rose' may be described as follows: Instead of being able to explicitly state that the bloom of a rose is red, the user adds this property to the whole concept 'rose'. In the data source though the colour of a rose is specified as green, since the data engineer described the stalk of a rose. The limited expressiveness of non-structured approaches leads to inaccurate similarity results.

6 Summary and Future Work

This paper develops a model to describe concepts in a structured way. It is built on existing approaches to represent concepts—either a feature- or a geometric model—and combines them with semantic nets which allow for modelling relations between concepts. These relations—hierarchic, partonomic or any other kind of non-taxonomic relation—enable also an explicit representation of the scope of properties.

Based on this flexible model we extended conventional similarity measures to include the scope of properties in the similarity assessment. The original idea underlying the measures—similarity based on the ratio of common and distinct features for feature models or similarity based on the spatial distance in conceptual spaces—is retained.

Geometric similarity measures compare values in same domains (e.g. two colours in the colour domain) and feature-based similarity measures compare corresponding features (e.g. 'apple tree' and 'cherry tree' have the common feature 'has_fruit', but 'fir' has not fruit, i.e. has the distinct feature '¬has_fruit'). Like the geometric and the feature-based approach, the hybrid model also measures similarity based on comparing corresponding properties. Properties without counterpart in the other concept do not influence the similarity.

Future research needs to investigate whether this missing information is also *some information* or not. We assume that missing counterparts for properties have negative effect on the similarity. The importance can easily be shown by an example: Rebecca is planning a fruit and vegetable garden and specifies a query concept for fruit trees with the property 'has_fruit'. The information retrieval system measures similarity to all other trees and finds a tree X which has identical properties like the query concept 'fruit tree', except it does not have any information on its fruits. There is no counterpart for the property 'has_fruit' in concept X (we do not know whether it 'has_fruit' or ¬'has_fruit'!). Since this property is essential for the query concept, this missing value should have negative impact on the similarity. But neither the feature nor the geometric model take into account missing counterpart properties.

Acknowledgement

We like to thank the members of the Iridium team at Ordnance Survey, the members of MUSIL and the 3 anonymous reviewers for their valuable comments.

References

Attneave, F. (1950). "Dimensions of Similarity." American Journal of Psychology 63: 516-556.

Devore, J. and R. Peck (2001). Statistics - The Exploration and Analysis of Data. Pacific Grove, CA, Duxbury.

Gärdenfors, P. (1999). Some tenets of Cognitive Semantics. Cognitive Semantics: Meaning and Cognition. J. Allwood and P. Gärdenfors. Amsterdam, John Benjamins: 19-36.

Gärdenfors, P. (2000). Conceptual Spaces: The Geometry of Thought. Cambridge, MA, MIT Press.

Gärdenfors, P. (2004). How to Make the Semantic Web More Semantic. Formal Ontology in Information Systems, Torino, Italy, IOS Press.

Gentner, D. and A. B. Markman (1997). "Structure Mapping in Analogy and Similarity." American Psychologist 52(1): 45-56.

Goldstone, R. L. (1994). "Similarity, Interactive Activation, and Mapping." Journal of Experimental Psychology: Learning, Memory, and Cognition 20(1): 3-28.

Goldstone, R. L. and D. L. Medin (1994). Similarity, Interactive Activation and Mapping: An Overview. Advances in Connectionist and Neural Computation Theory. J. Barnden and H. K. New Jersey, Ablex. Vol. 2: Analogical Connections: 321-362.

Goldstone, R. L. and J. Son (2004). Similarity. Cambridge Handbook of Thinking and Reasoning. R. Morrison. Cambridge, Cambridge University Press.
Markman, A. B. (1999). Knowledge Representation. Mahwah, New Jersey, Lawrence Erlbaum Associates.
Melara, R. D., L. E. Marks, et al. (1992). "Optional processes in similarity judgments." Perception & Psychophysics 51(2): 123-133.
Rada, R., H. Mili, et al. (1989). "Development and application of a metric on semantic nets." IEEE Transactions on systems, man, and cybernetics 19(1): 17-30.
Raubal, M. (2004). Formalizing Conceptual Spaces. Formal Ontology in Information Systems, Proceedings of the Third International Conference (FOIS 2004). A. Varzi and L. Vieu. Amsterdam, NL, IOS Press. 114: 153-164.
Rodríguez, A. (2000). Assessing Semantic Similarity Among Spatial Entity Classes. Spatial Information Science and Engineering. Maine, PhD Thesis. University of Maine: 168.
Rodríguez, A. and M. Egenhofer (2003). "Determining Semantic Similarity Among Entity Classes from Different Ontologies." IEEE Transactions on Knowledge and Data Engineering 15(2): 442-456.
Sattath, S. and A. Tversky (1987). "On the Relation Between Common and Distinctive Feature Models." Psychological Review 94(1): 16-22.
Schwering, A. (2004). Semantic Neighbourhoods for Spatial Relations (Extended Abstract). Third International Conference on Geographic Information Science (GIScience), Maryland, USA, Regents of the University of California.
Schwering, A. and M. Raubal (2005 forthcoming). Spatial Relations for Semantic Similarity Measurement. 2nd International Workshop on Conceptual Modeling for Geographic Information Systems (CoMoGIS2005), Klagenfurt, Austria., Springer.
Shepard, R. N. (1957). "Stimulus and Response Generalization: A Stochastic Model Relating Generalization to Distance in Psychological Space." Psychometrika 22(4): 325-345.
Shepard, R. N. (1958 a). "Stimulus and Response Generalization: Deduction of the Generalization Gradient from a Trace Model." Psychological Review 65(4): 242-256.
Shepard, R. N. (1958 b). "Stimulus and Response Generalization: Tests of a Model Relating Generalization to Distance in Psychological Space." Journal of Experimental Psychology 55(6): 509-523.
Smith, E. E. (1989). Concepts and Induction. Foundations of cognitive science. M. I. Posner. Cambridge, MA, MIT Press: 501-526.
Suppes, P., D. M. Krantz, et al. (1989). Foundations of Measurement - Geometrical, Threshold, and Probabilistic Representations. San Diego, California, USA, Academic Press, Inc.
Tversky, A. (1977). "Features of Similarity." Psychological Review 84(4): 327-352.
Tversky, A. and I. Gati (1978). Studies of Similarity. Cognition and Categorization. E. Rosch and B. Lloyd. Hillsdale, NJ, Lawrence Erlbaum: 79-98.
Tversky, A. and I. Gati (1982). "Similarity, Separability, and the Triangle Inequality." Psychological Review 89(2): 123-154.

Ontology-Based Spatial Query Expansion in Information Retrieval

Gaihua Fu, Christopher B. Jones, and Alia I. Abdelmoty

School of Computer Science, Cardiff University, Cardiff, UK
{Gaihua.Fu, C.B.Jones, A.I.Abdelmoty}@cs.cf.ac.uk

Abstract. Ontologies play a key role in Semantic Web research. A common use of ontologies in Semantic Web is to enrich the current Web resources with some well-defined meaning to enhance the search capabilities of existing web searching systems. This paper reports on how ontologies developed in the EU Semantic Web project SPIRIT are used to support retrieval of documents that are considered to be spatially relevant to users' queries. The query expansion techniques presented in this paper are based on both a domain and a geographical ontology. The proposed techniques are distinguished from conventional ones in that a query is expanded by derivation of its geographical query footprint. The techniques are specially designed to resolve a query (such as *castles near Edinburgh*) that involves spatial terms (e.g. *Edinburgh*) and fuzzy spatial relationships (e.g. *near*) that qualify the spatial terms. Various factors are taken into account to support intelligent expansion of a spatial query, including, spatial terms as encoded in the geographical ontology, non-spatial terms as encoded in the domain ontology, as well as the semantics of the spatial relationships and their context of use. Some experiments have been carried out to evaluate the performance of the proposed techniques using sample realistic ontologies.

Keywords: Ontology, Semantic Web, Spatial Search, Query Expansion.

1 Introduction

The WWW holds vast amounts of information. However, users do not always get information they expect when searching the Web. One main reason for this is that existing web documents are rarely augmented with semantic annotation that describe their content, which would make them more easily accessible to automated search facilities. The Semantic Web is one of several proposed solutions to resolve this problem [29]. One aim of the Semantic Web is to enrich the current web documents with some well-defined meaning (meta-data), so that the existing web searching systems can be extended to have more advanced capabilities to find these resources more effectively. It has long been recognized in the Semantic Web research that ontologies play a key role as they can be used as a source of shared and precisely defined terms for such meta-data [14, 19].

Apart from annotating web documents with semantic information, ontologies have also been employed to resolve the mismatch problems between queries and

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1466–1482, 2005.

documents, i.e. a query may not be expressed in terms that match the ones contained in some of the relevant documents. Traditionally this is dealt with using query expansion techniques which expand a query with the terms (as encoded in ontologies or other knowledge resources) that are considered to be related to the ones in the query, so that the relevant documents can be retrieved. Most of these studies use a term-based method [25, 1, 10, 30, 7]. For example, a query expansion method is introduced in [28] which extends a query with the words that are lexically related to the original query words using WordNet. A method is introduced in [15] to expand a query term with the ones that can be reached transitively in a concept network that is built up according to a thesaurus.

While these studies are useful for processing a general query, they provide inadequate support for processing a spatial query. A spatial query is different from a generic one in that it usually includes one or more spatial terms. It is often used by a user when he/she wishes to find Web resources that are related to a place. An example of such a query is *castles near Edinburgh*. Support for this type of query is necessary as most human activities are rooted in geographical space in some aspect, and therefore many documents include references to geographical context, typically by means of place names. Conventional search engines treat spatial terms involved in a query in the same way as other terms and can not always ensure good search results due to the lack of spatial awareness. This has led to research interests in developing spatial search techniques to help users find resources in which the subject matter is related to a place [13, 12, 22].

As with a generic query, there is also a need to expand a spatial query. While query expansion has been studied extensively in the literature, the interest here is how to expand a spatial query so that documents that are considered to be *spatially* relevant can be retrieved. A document can be spatially relevant to a query in different ways. It may be spatially relevant to a query by involving a geographical term that is considered to be an alternative name for the one appearing in the query. A document may also be spatially relevant to a query by involving places which satisfy the specified spatial relationship with the one appearing in the query. An example of this is a query looking for *castles near Edinburgh*. The relevant documents may not only include the ones that describe castles in *Edinburgh*, but also the ones that describe castles in places such as *West Lothian* and *Midlothian*, which are geographically *near* to *Edinburgh*.

Conventional term-based query expansion techniques can be utilised to resolve a spatial query. However, the danger is that they may introduce too many query terms in spatial context, perhaps many thousands, and may therefore become intractable for the query processing facilities. Another challenge in dealing with spatial query expansion is that a spatial relationship involved in a query can be vague. Its interpretation can vary with respect to different users' intentions, as well as depending on the types of spatial and non-spatial terms involved in a query. For example, one user may use *near* to refer to places that are either inside of or adjacent to a place presented in a query, and another user may use it to refer to places that are only adjacent to the specified place. Also, a spatial

relationship may need to be interpreted differently due to different subject matters involved. For example, *near* in the query *lakes near Edinburgh* may need to be treated differently from *near* in *hotels near Edinburgh*.

In this paper we report the spatial query expansion techniques developed in the EU Semantic Web project SPIRIT. The query expansion techniques presented in this paper are based on both a domain and a geographical ontology. Different from term-based query expansion techniques, the proposed techniques expand a query by trying to derive its geographical query footprint, and it is specially designed to resolve a spatial query. Various factors, such as types of spatial terms as encoded in the geographical ontology, types of non-spatial terms as encoded in the domain ontology, the semantics of the spatial relationships, their context of use, and satisfiability of initial search result, are taken into account to support expansion of a spatial query. The proposed techniques support the intelligent, flexible treatment of a spatial query when a fuzzy spatial relationship is involved. Some experiments have been carried out to evaluate the performance of the proposed techniques using sample realistic ontologies.

The remaining part of the paper is organized as following. Section 2 studies related work. Section 3 introduces the background knowledge of this research, discusses various factors that affect spatial query expansion, and presents how SPIRIT ontologies are designed to support spatial query expansion. Section 4 presents our method that supports spatial query expansion. Section 5 reports our experimental results. Section 6 concludes the paper and points out the possible future research.

2 Related Work

Query expansion is traditionally considered as a process of supplementing a query with additional terms as the assumption is that the initial query as provided by the user may be an inadequate representation of the user's information needs [28, 30, 15, 5, 7]. Query expansion techniques can broadly be classified in two categories: those based on the search results and those that are based on some forms of knowledge structure. The former group of techniques depends on the search process and uses relevance feedback in an earlier iteration of search as the resource to identify the query expansion terms [1, 4, 7]. The latter group of techniques is independent of the search process and additional query terms are derived by traversing a semantic network built up according to a knowledge structure. Knowledge structures used by this group of techniques can either be a general-purpose ontology (or thesaurus) [28], or an ontology built for a specific domain [15], or an ontology constructed from document collection based on the term clustering [20]. Work that combines the two approaches is reported in [30], where authors apply term clustering techniques to the local set of documents.

The work reported in this paper belongs to the second group of research, i.e., both a domain ontology and a geographical ontology are utilised to support query expansion. In the literature, there are several search engines that employ ontologies to support spatial query expansion [22, 18]. For example, Mirago has

developed a regional web search facility that provides spatial search services for several European countries including UK, Germany, France and Spain [22]. A user can issue a spatial query by typing a domain term and selecting from available place names (as encoded in a geographical ontology) the one that he/she would like his/her search to focus on, and documents that employ both the domain term and the spatial term in their text are retrieved. Mirago supports some limited spatial expansion by using the spatial containment relationship existing between places (as encoded in the geographical ontology). That is, if no or few documents are found according to a spatial query term, the term is replaced with a place name whose region immediately contains it.

In addition to term-based spatial query expansion research, recently some geographical search systems employ footprint-based spatial query expansion techniques to assist with retrieval of spatially relevant documents (the footprint of a query refers to the spatial search space of a query). For example, the geographical search engine developed by Vicinity [27] allows the user to enter part or all of an address in the USA or Canada, along with a category of interest and a search radius in miles. Google has recently introduced a locational web search system based in the USA [13]. Like the Vicinity search tools it allows the user to specify the name of a place of interest using an address or zip code, which is then matched against relevant documents. Other research which considers the spatial search is that of [8, 2, 9, 3, 21]. All these spatial search engines support the *inside* spatial relationship, and a few of them support the *distance* relationship as well. Though relatively little has been published on the technology that underlies spatial query expansion by these systems, according to authors' investigation of some search results of these systems, it appears they perform query expansion by simply translating a place name into its corresponding coordinate footprint.

The main advantage of footprint-based query expansion is that it avoids introducing too many query terms, which, as discussed in [28], is not as effective as supposed to be. Furthermore, footprint-based query expansion can effectively avoids retrieval of irrelevant documents due to name sharing (according to [24], about 16.6 percent of European place names have multiple uses), which is usually inevitable in term-based query expansion. Finally, footprint-based expansion allows us to perform more accurate spatial relevance calculation by analysing the query footprint and the document footprint, which is not possible with term-based expansion.

The work reported in this paper studies footprint-based spatial query expansion techniques. It is distinguished from previous research in several aspects. First, it supports spatial query expansion especially when a fuzzy spatial relationship term such as *near* is presented in a query, which is largely not considered in other research. A wide range of spatial fuzzy spatial relationship terms are supported by the techniques proposed in this paper. Secondly, the proposed techniques support intelligent and flexible spatial query expansion. This is achieved by taking into account of various factors, e.g. spatial query term, non-spatial query terms, the use context of a spatial relationship etc., when computing a

query footprint. Thirdly, we support iterative spatial query expansion, i.e. a query footprint will be progressively extended when initial search results are not sufficient. This in one aspect ensures search satisfactory. On the other hand, it ensures the most spatially relevant documents will be retrieved first, which is difficult to be achieved with traditional query expansion techniques if not impossible.

3 SPIRIT Queries, Query Expansion and Ontologies

The work reported in this paper is part of the SPIRIT project (Spatially-Aware Information Retrieval on the Internet) [6]. The aim of SPIRIT is to develop Web search technology that is specialised for access to documents relating to places or regions referred to in a query. A primitive spatial query in SPIRIT can be formalised as a triple:

$$\langle what, rel, where \rangle$$

where the *what* term is used to specify a general non-spatial object, which may correspond to a physical or an abstract subject or activity; *where* is used to specify a spatially referenced term; the *rel* term is a spatial relationship which relates *what* and *where*.

The following concepts are used throughout the paper to illustrate our techniques. A spatial term is the one which has a footprint *P-footprint*.

Definition 1. *The footprint P-footprint of a spatial term indicates the geographical location of the intended place, and is specified in terms of map coordinates with a selected reference system.*

A document may have footprint *D-footprint* if it involves one/more spatial terms.

Definition 2. *A document footprint D-footprint defines the geographical coverage of a specified document, and it may consist of multiple P-footprints if more than one place name appears in the document.*

Given a spatial query $\langle what, rel, where \rangle$, the purpose of spatial query expansion in this research is to generate a query footprint (denoted as *Q-footprint*). Ideally, *Q-footprint* should be computed in such a way so that spatially relevant documents of $\langle what, rel, where \rangle$ are those whose document footprints fall in *Q-footprint*.

Definition 3. *A query footprint Q-footprint defines a geographical space that covers the intended spatial search extent of* $\langle what, rel, where \rangle$*, and it is specified in the form of map coordinates.*

Given $\langle what, rel, where \rangle$, deriving *Q-footprint* will start with the *P-footprint* of *where*. The most important information that influences *Q-footprint* is the *rel* term, and it determines what geographical area should be covered by *Q-footprint*.

For example, if *rel* is *near*, the query footprint may be assumed to be the area surrounding *where*. If *rel* is *north*, the geographical area that covers north of *where* should be returned.

Most spatial relationships are fuzzy, and their semantics can vary when used with different combinations of *what* and *where*. Consequently, *Q-footprint* may be different when the same *rel* is used in different contexts. Given $\langle what, rel, where \rangle$, we consider that the interpretation of *rel* is mainly determined by the following factors:

- the *type* of *where*. This is because the search extent is usually assumed differently where different types of *where* are presented in queries. For example, given *near*, we tend to assume a bigger search space when *where* is of type *city* than when it is of type *village*.
- the *P-footprint* of *where*. Some places are of the same type, but the areas they cover can vary. For example, both *London* and $Cardiff$ are type of *city*, but the area of *London* is much bigger than that of $Cardiff$. Therefore it is reasonable to assume a larger neighbourhood region of *London* than that of *Cardiff*.
- the *what* term. Given a geographical area, the distribution densities of different *what* subjects may vary, and therefore some subjects may have more documents describing them than others. For example, there are more *hotels* than *airports* for most places. For a subject which has a sparse distribution density in an area, it tends to require a bigger search space in order to find some relevant documents.
- the user's intention of using a *rel* term. Different users may employ a same *rel* with different meanings in mind. For example, one user may use *near* to refer to a region that covers both the *where* and its surrounding areas, while another user would use *near* only to refer to the neighbouring regions of *where*. Therefore it is desirable that *rel* can be interpreted by taking into account of the user's intention in mind.

We are aware that other factors may also affect the interpretation of a *rel* term, for example, the *population* of *where* if it is an inhabited area. However, most of these factors apply to the specific type of queries (e.g. *population* only needs to be considered for a query whose *where* term represents an inhabited place), whereas the factors considered in this research apply to generic spatial queries. Therefore our techniques support spatial query expansion by mainly considering the generic factors. However, when a query footprint does not produce good search results, our techniques support iterative spatial query expansion (see Section 4 for details).

To support spatial query expansion, the SPIRIT system has incorporated into its architecture an ontology component, of which the primary parts are a domain ontology and a geographical ontology (or geo-ontology)[1]. The domain

[1] SPIRIT ontology design was also driven by other spatial search requirements, e.g. spatial query disambiguation, spatial relevance ranking, spatial index and annotation of web resources, as discussed in [11, 16].

ontology models the terminologies of one application area or domain, and is used to resolve the *what* aspect of a SPIRIT query. Modelling of domain-specific terminology is accomplished using conventional thesauri methods. Equivalent terms or synonyms are represented via `USE` and `USE-FOR` relations. Hierarchical relations whether generic (is-a) or metronymic (part-of) are represented with Broader Term (`BT`) and Narrower Term (`NT`) relations. For each term, the domain ontology maintains a coefficient that indicates the influence of it on the interpretation of a spatial relationship, and this is derived by carrying out some document density studies.

The *where* aspect of the SPIRIT query is dealt with the SPIRIT geo-ontology, which is constructed to provide a knowledge structure of the interested geographic space. Several types of information are encoded in the geo-ontology, including the various names that a place is known by, the place types with which it can be categorised, its topological relationships (such as *partof* and *containing*) with other places, and its geographical footprints (*P-footprint*). For each category of place, the geographical ontology maintains a coefficient that indicates the influence of it on the interpretation of a spatial relationship, and this is derived by carriying out some user studies.

4 A Method for Deriving Spatial Query Footprint

This section describes how spatial query expansion is performed by employing the SPIRIT ontologies. The proposed techniques are mainly designed to handle spatial queries with fuzzy spatial relationships presented, and the group of spatial relationships that can be handled by using techniques proposed in this paper includes *in, near, outside, north-of, south-of, east-of, west-of* and *within a specified distance*[2].

Apart from a domain ontology and a geographical ontology, we assume the availability of the alternative interpretation of a spatial relationship *rel*. For example, for *near*, three options may be available for its interpretation: an area covers only *where*, an area covers both *where* and its surrounding regions and an area covering neighbouring regions of *where*. The statistical data of a spatial relationship in search needs to be maintained to record the option that a user may choose in search processes, and the frequency that an option is chosen for interpreting a spatial relationship [3].

The proposed techniques support iterative spatial query expansion. This is necessary for several reasons. First, for some topics, inadequate documents may exist on Web to describe them. Secondly, some information encoded in the ontologies, such as coefficient data which indicates the influence of a domain term on the interpretation of a *rel* term, may not be as valid as they are supposed to be, especially when experiments for obtaining these parameter values are too expensive to perform exhaustively. Finally, query footprint will be derived by

[2] Other spatial relationships need to be treated differently and our follow up paper will elaborate on this.

[3] This is achieved by maintaining a log file.

taking some generic factors into accounts, while some specific types of query may need to consider other factors, as stated in Section 3. Therefore it is desirable that that spatial query expansion can be performed iteratively when initial search results are not satisfactory.

In what follows, we will use $Q\text{-}footprint_i$ to denote the query footprint generated at the *i-th* iteration of query expansion. We first describe how the initial query footprint $Q\text{-}footprint_1$ is computed, and then describe how query footprint can be incrementally expanded when initial search results are inadequate. We will use the geographical space (which covers the UK county "Oxfordshire" and its surrounding area) shown in Figure 1 to illustrate the techniques proposed.

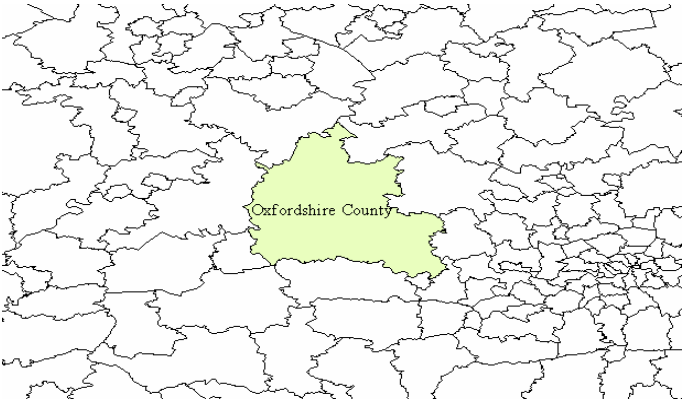


Fig. 1. `Oxfordshire` and its Surrounding Area

4.1 Initial Spatial Query Expansion

The following steps describe how $Q\text{-}footprint_1$ is generated.

1. Though *P-footprint* of the *where* term is the starting point from which $Q\text{-}footprint_1$ is generated, the type of geometric operation that is performed over *P-footprint* for generating $Q\text{-}footprint_1$ is determined by *rel*. For example, if *rel* is *near*, then a buffer operation needs to be performed over *P-footprint* for generating $Q\text{-}footprint_1$. Therefore the first step of computing *Q-footprint* is to determine the type of geometric function required according to *rel*. This is shown by using following function:

$$GeoOp = \beta(rel) \tag{1}$$

where the function β maps a spatial relationship *rel* to a corresponding geometric function name. For example, if *rel* is *near*, the function β will generate value $Buffer$ for *GeoOp*. Different *rel* terms result in query footprints of different orientation and geometries. Figure 2 shows some example query footprints (polygons plotted with bold lines), when *rel* stands for *near, outside-of* and *north-of*. When a query is in the form of $\langle what, near, Oxfordshire \rangle$,

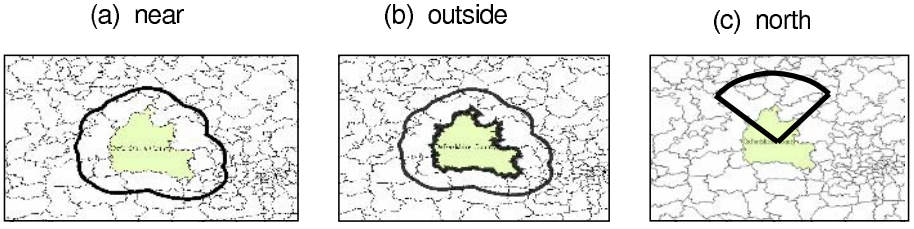


Fig. 2. Different *rel* Terms Resulting in Different Query Footprints

Q-footprint is the space that covers both $Oxfordshire$ and its surrounding areas. *Q-footprint* for $\langle what, outside\text{-}of, Oxfordshire\rangle$ is quite similar to the one for $\langle what, near, Oxfordshire\rangle$, but it only covers surrounding regions of $Oxfordshire$. If a query is in the form of $\langle what, north\text{-}of, Oxfordshire\rangle$, then the area that covers the north and northern part of $Oxfordshire$ is returned as *Q-footprint*.

2. To derive exact geographical coverage of $Q\text{-}footprint_1$, a geometric function *GeoOp* requires the following parameters:
   (a) the *P-footprint* of the *where* term, and it can be retrieved from the SPIRIT geo-ontology. This gives us the initial geometry from which $Q\text{-}footprint_1$ is to be generated;
   (b) a geometric distance d that is required for extending *P-footprint* to generate $Q\text{-}footprint_1$. The group of fuzzy *rel* terms studied in this paper determines that $Q\text{-}footprint_1$ is generated by extending *P-footprint* at a specified distance in a certain way. For example, if *rel* is *near*, $Q\text{-}footprint_1$ may be generated to cover areas extended from *P-footprint* at a specified distance. If *rel* is *north*, $Q\text{-}footprint_1$ may be generated to cove areas extended from the north part of *P-footprint* at a specified distance. The exact distance d for geometric expansion is determined by the following:
      i. the area size of *P-footprint*, and it is used to determine the initial extension distance using the formula shown below:[4]

$$id = \sqrt{\frac{area(P\text{-}footprint)}{\pi}} \tag{2}$$

      That is, the initial extension distance is assigned the approximate radius of *P-footprint*;
      ii. a coefficient p_1 which determines the influence of the *what* term, this can be retrieved from the SPIRIT domain ontology;
      iii. a coefficient p_2 which determines the influence of the *where* term, that can be retrieved from the SPIRIT geo-ontology;

   The exact expansion distance is therefore determined by the following:

$$d = id * p_1 * p_2 \tag{3}$$

[4] This formula is used in our preliminary study, and some more user and performance experiments need to be carried out to validate it.

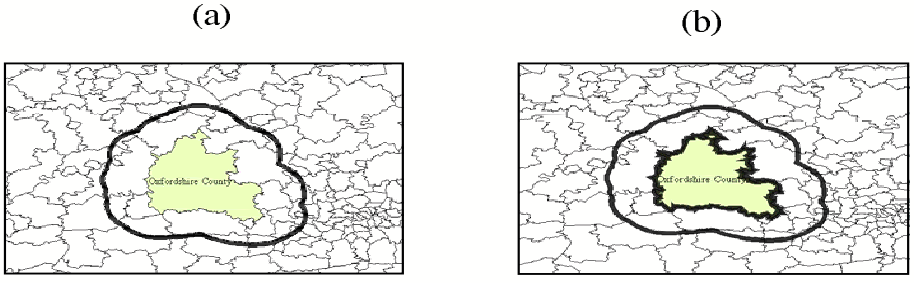


Fig. 3. Different Interpretation of Spatial Relationship *near*

(c) As we mentioned earlier, each *rel* may have different interpretations. This can either be chosen by a user or be derived from the SPIRIT log file which encodes the most frequently used option for a specified *rel*. This is assigned to parameter p_3[5]. For example, Figure 3 shows query footprints when *near* is interpreted differently – one covers both the *where* and its surrounding areas and another just covers the neighbouring regions of *where*.

3. The parameters derived in step 2(a), 2(b) and 2(c) are passed on to the geometric function *GeoOp* generated in Step 1 to derive *Q-footprint*$_1$.

$$\textit{Q-footprint}_1 = \textit{GeoOp}(\textit{P-footprint}, d, p_3) \tag{4}$$

4.2 Iterative Spatial Query Expansion

When an initial search results fail to satisfy the user's query need, our method regenerates *Q-footprint* to cover some regions beyond that of *Q-footprint*$_1$. This section shows how this is achieved. Given $\langle what, rel, where\rangle$, we derive *Q-footprint*$_i$ if the search results of *Q-footprint*$_1, \ldots,$ *Q-footprint*$_{i-1}$ are not satisfactory[6]. The procedure below describes how iterative spatial query expansion is performed:

1. derive *Q-footprint*$_i$ if the iteration criterion is satisfied (e.g. when search result is not satisfactory after *i-1* rounds of iteration). *Q-footprint*$_i$ can be derived largely using spatial query expansion procedure described in Section 4.1. The difference is that we further enlarge the geometric distance d generated in the formula (3) according to:

$$d = d * i \tag{5}$$

[5] That is, the most frequently used interpretation of *rel* is used by the system by default. However, the SPIRIT user interface allows a user to choose other options as well.

[6] Various factors can control iterative query expansion process, e.g. a new iteration can be triggered when no or few documents are retrieved in initial search, and iteration can be interrupted if the allocated search time runs out. This is beyond the topic of this paper and therefore will not be discussed further here.

2. subtract from $Q\text{-}footprint_i$ the area covered by $Q\text{-}footprint_1, \ldots, Q\text{-}footprint_{i-1}$ so that to avoid spatial search redundancy:

$$Q\text{-}footprint_i = Q\text{-}footprint_i - \sum_{1}^{i-1} Q\text{-}footprint_k \tag{6}$$

Figure 4 shows query footprints that are progressively generated in order to find documents for the query $\langle airports, near, Oxfordshire \rangle$, and we can see that it has been spatially expanded three times in its effort to find spatially relevant documents.

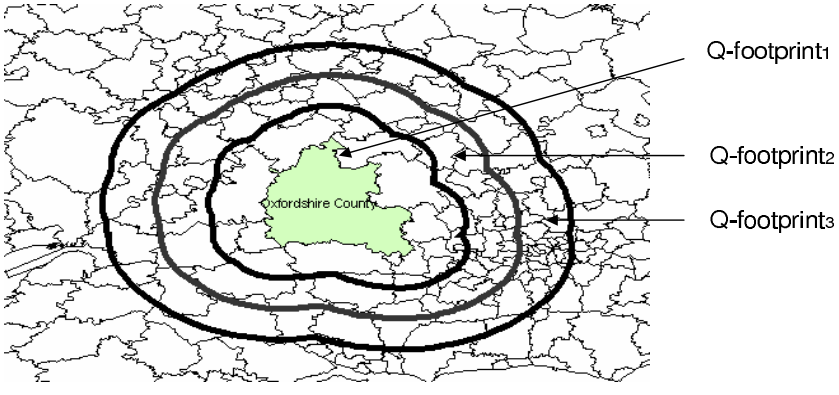


Fig. 4. Iterative Spatial Query Expansion

5 Implementation and Evaluation

To verify the spatial query techniques proposed, we have carried out some experiments. In this section, we will demonstrate how query expansion techniques are used in SPIRIT to improve search results, we also report on the experiments which were carried out to study the time cost for performing spatial query expansion using SPIRIT ontologies.

Query expansion techniques are implemented using Java, and they interact with SPIRIT ontology databases (composed of a domain ontology and a geo-ontology) to compute query footprints. The domain ontology contains the terms that are used in tourism area, and 2223 terms are encoded. The geo-ontology contains geographical places of several European countries, including United Kingdom, France, Germany and Switzerland, and 125,812 places are encoded. Both domain and geo-ontology are stored in Oracle 9.2.0. Once a query footprint is generated, it is feed to SPIRIT search component to retrieve the relevant documents. All experiments were carried out on a Pentium 4 PC with a 2.00 GHz processor and 516 MB of memory, running Microsoft Windows/XP. The SPIRIT adopts a distributed architecture (see [17] for details), and query expansion services talks to other components of the system through Apache SOAP.

5.1 Precision Study

This section demonstrates the effectiveness of the spatial query expansion techniques proposed. This is achieved by comparing the search results obtained by the SPIRIT system when spatial query expansion option is either switched on and off. When spatial query expansion is on, the SPIRIT system performs query expansion using techniques proposed, search is carried out basing on both the spatial and the textual index of web collection, and relevant ranking is performed using techniques proposed in [26]. When spatial query expansion is off, all query terms (including spatial and non-spatial ones) are send to the search component to perform a textual based search, and `BM25` proposed in [23] is used to rank the search results.

Table 1. Search Topics

query 1	⟨castles, inside, Cardiff⟩
query 2	⟨castles, near, Cardiff⟩
query 3	⟨castles, north-of, Cardiff⟩
query 4	⟨castles, outside-of, Cardiff⟩

The experiments were carried out using a set of queries (shown in Table 1). The queries involve *rel* terms *inside, near, north-of* and *outside-of*. Since other *rel* terms such as *south-of* and *east-of* are treated similarly with *north-of* using our techniques, we consider the set of *rel* terms are sufficient for evaluation purposes. The results produced from running these queries were analysed for P10 (precision at 10) accuracy. The top ten results were examined by human users to judge their spatial relevance to the given queries. To help with judging spatial relevance of the retrieved documents, the UK city *Cardiff* and its surrounding areas, which are familiar to the intended users, were chosen for the queries to focus on.

A retrieved document was classified as three types in our experiments: relevant, irrelevant, and partially relevant. The first two types are easy to understand. A document is classified as *partially relevant* it is not designed to describe the search topic but it contains a link that points to a relevant page, e.g. a directory page. We note that a *rel* term can be interpreted differently using our system, however due to human effort required, we were only able to perform experiments for a fixed number of settings. Table 2 shows the experiment results, where columns 3, 4, 5 display the numbers of relevant, partially relevant and irrelevant documents retrieved.

When *rel* term is *inside*, the query footprint *Q-footprint* is *P-footprint* of *where*. It is not obvious that the search system performed better when spatial query expansion option was switched on. However, with query expansion option switched on, we observed that documents, which describe castles in terms of subareas of *Cardiff*, or alternative names of *Cardiff*, i.e. *Caerdydd*, were retrieved. This did not happen when query expansion option was switched off. The main reason for this is that our spatial query expansion is footprint-based, and retrieved documents are the ones whose documents footprints fall in query

Table 2. Experimental Results

query	spatial query expansion	relevant	partially relevant	irrelevant
1	off	3	6	1
	on	2	6	2
2	off	1	3	6
	on	5	5	0
3	off	2	3	5
	on	4	6	0
4	off	0	4	6
	on	5	4	1

footprint. Different documents may have different geographical terms in their text, but if these geographical terms refer to same places, these documents have the same document footprints, which all fall in *Q-footprint*. Documents specified in term of subareas of *where* have footprints which are subsets of *Q-footprint*, therefore are retrieved as well.

When *rel* term is *near*, *Q-footprint* were generated covering *P-footprint* of *where* plus its surrounding areas. The search system performed better with spatial query expansion switched on – the top 10 retrieved documents are either relevant or partially relevant. The footprint-based query expansion enabled us to retrieve documents which describes castles not only in $Cardiff$ but also in places like *Caerphilly, Newport, Dinas Powys, Abergavenny* and *Swansea*. Since these places are geographically close to *Cardiff*, the retrieved documents are spatially relevant to the query. When spatial query expansion was off, it appeared that all retrieved documents involve the terms *castles, near* and *Cardiff*. Unfortunately, many of these documents do not actually describe castles *in* or *near* to $Cardiff$.

When *rel* term is *north-of*, *Q-footprint* were generated covering northern part of *where* plus areas that are north of *where*. From Table 2, we can see that the system performed considerably better when spatial query expansion was switched on. The reason is the same with *near* – the footprint-based query expansion enables us to retrieved documents whose footprints satisfy specified geometric relationship with query footprint. However, when spatial query expansion was off, many documents retrieved are the ones which happen to have the terms *castles, north-of* and *Cardiff* presented, but do not actually describe castles in the northern or north of *Cardiff*.

When *rel* term is *ouside-of*, *Q-footprint* were generated covering only surrounding area of *where*. Same with *near* and *north-of*, the system presented its inability to deal this type queries when spatial query expansion was off, whereas it performed considerably better when spatial query expansion option was on.

5.2 Time Cost Study

Due to the complexity of the original geometric footprint of a place, only two approximation representations of a footprint, MBR and convex hull polygon,

were utilised to deal with spatial query expansion in SPIRIT. An MBR is the minimum bounding rectangle of a geometry object, and a convex hull polygon is the smallest convex polygon that completely encloses a geometry object.

We first compared the time costs of query expansion by using MBRs and convex polygons, and the mean response time of using two types of footprint for query expansion is shown in Figure 5.

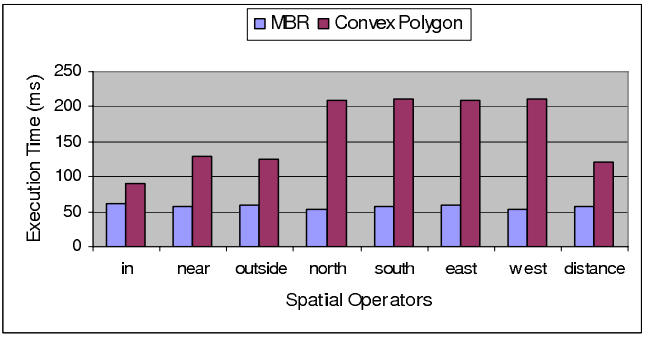


Fig. 5. Response Time of Query Expansion by using MBRs and Convex Polygons

From Figure 5, we can observe that it requires more CPU time to derive query footprint using convex polygon than using MBR. This is mostly due to the complex nature of convex polygons. A MBR is composed of two coordinate points, while a convex polygon can have more coordinate points, ranging from 7 to 38 according to our geographical ontology. However, the CPU time required for deriving query footprints using both MBR and convex polygon are in a range that is acceptable in a SOAP-based distributed search environment, i.e. about 60 milliseconds for MBR and about 210 milliseconds for convex polygon.

We then studied the time cost of query expansion using convex polygons with different complexity, i.e. convex polygons composed of different numbers of coor-

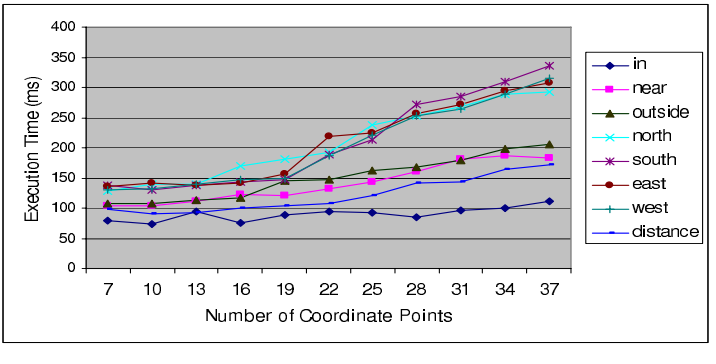


Fig. 6. Impact of Coordidnate Point Number

dinate points, and the result is shown in Figure 6. As we can see, that response time increases with number of coordinate points – the more coordinate points a convex polygon has, the more CPU time is required for deriving the query footprint. This increase is obvious when dealing with spatial relationships *north, south, east* and west. However, for all spatial relationship terms, the increase displays a linear tendency.

6 Conclusions

In this paper we have introduced an ontology-based spatial query expansion method that supports retrieval of documents that are considered to be spatially relevant. The proposed method expands a spatial query by trying to derive its geographical query footprint, and it is specially designed to resolve a query that involves a fuzzy spatial relationship. Both a domain and a geographical ontology are employed to support spatial query expansion. Various factors are taken into consideration for supporting intelligent expansion of a spatial query, and proposed method also supports iterative spatial query expansion when initial spatial searches are not satisfactory. Our experiments show that the proposed method can considerably improve search results when a query involves a fuzzy spatial relationship, and experiments also show that proposed method works efficiently using realistic ontologies in a distributed spatial search environment. The method reported in this paper is proposed to deal with a group of spatial relationships that frequently appear in spatial search, and how to resolve other spatial relationships, e.g. *between*, still requires further investigation.

Acknowledgement

This work is funded by Grant IST-2001-35047 from EC Fifth Framework Programme.

References

1. R. Attar and A. S. Fraenkel. Local Feedback in Full-Text Retrieval Systems. *Journal of the ACM*, 24(3):397–417, July 1977.
2. S. Bressan, B. Ooi, and F. Lee. Global Atlas: Calibrating and Indexing Documents from the Internet in the Cartographic Paradigm. In *Proceedings of the 1st International Conference on Web Information Systems Engineering*, volume 1, pages 117–124, 2000.
3. O. Buyukokkten, J. Cho, H. Garcia-Molina, L. Gravano, and N. Shivakumar. Exploiting Geographical Location Information of Web Pages. In *Proceedings of Workshop on Web Databases (WebDB'99) held in conjunction with ACM SIGMOD'99*. ACM press, 1999.
4. D. Cai, C. J. Rijsbergen, and J. M. Jose. Automatic Query Expansion based on Divergence. In H. Paques, L. Liu, and D. Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM-01)*, pages 419–426, New York, Nov. 5–10 2001. ACM Press.

5. C. Carpineto, R. de Mori, G. Romano, and B. Bigi. An Information-Theoretic Approach to Automatic Query Expansion. *ACM Transactions on Information Systems*, 19(1):1–27, 2001.
6. C.B. Jones and R. Purves and A. Ruas and M. Sanderson and M. Sester and M.J. van Kreveld and R. Weibel. Spatial Information Retrieval and Geographical ontologies: an Overview of the SPIRIT Project. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 387–388, 2002.
7. H. Cui, J. Wen, and M. Li. A Statistical Query Expansion Model Based on Query Logs. *Journal of Software*, 14(9):1593–1599, 2003.
8. D. Egnor. http://www.google.com/programming-contest/winner.html.
9. J. Ding, L. Gravano, and N. Shivakumar. Computing Geographical Scopes of Web Resources. In *Proceedings of the 26th Very-Large Database (VLDB) Conference*, pages 546–556. Morgan Kaufmann, 2000.
10. E. N. Efthimiadis. Query Expansion. In M. E. Williams, editor, *Annual Review of Information Science and Technology*, volume 31, pages 121–187. American Society for Information Science, 1996.
11. G. Fu, C. Jones, and A. I. Abdelmoty. Building a Geographical Ontology for Intelligent Spatial Search on the Web. In *Proceedings of IASTED International Conference on Databases and Applications*, pages 167–172. Spriner Verlag, 2005.
12. GBdirect Ltd. SomeWhere Near. http://somewherenear.com/.
13. Google. Google Location Search. http://local.google.com/lochp.
14. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.
15. K. Järvelin, J. Kekäläinen, and T. Niemi. ExpansionTool: Concept-Based Query Expansion and Construction. *Information Retrieval*, 4(3/4):231–255, 2001.
16. C. Jones, A. Abdelmoty, and G. Fu. Maintaining Ontologies for Geographical Information Retrieval on the Web. In *Proceedings of OTM Confederated International Conferences CoopIS, DOA, and OOBASE*, pages 934–951. Spriner Verlag, 2003.
17. C. Jones, A. I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The SPIRIT Spatial Search Engine: Architecture, Ontologies and Spatial Indexing. In *Proceedings of the 3rd International Conference on Geographic Information Science*, pages 125–139.
18. C. Jones, D. Tudhope, and H. Alani. Augmenting Thesaurus Relationships: Possibilities for Retrieval. *Journal of Digital Information*, 1(8), Jan. 15 2001.
19. O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. W3C Recommendation, 1999. Available at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.
20. R. Mandala, T. Tokunaga, and H. Tanaka. Combining General Hand-Made and Automatically Constructed Thesauri for Query Expansion in Information Retrieval. In D. Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 920–925, S.F., July 31–Aug. 6 1999. Morgan Kaufmann Publishers.
21. K. McCurley. Geospatial Mapping and Navigation of the Web. In *Proceedings of Tenth International World Wide Web Conference*, page Session P7. ACM press, 2001.
22. Mirago: Mirago the UK Search Engine. http://www.mirago.co.uk/.
23. S. E. Robertson, S. Walker, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the 3rd Text REtrieval Conference (TREC3)*.

24. D. A. Smith and G. S. Mann. Bootstrapping Toponym Classifiers. In *Proceedings of the HLT-NAACL 2003 Workshop on Analysis of Geographic References*, pages 45–49.
25. K. Sparck Jones. *Automatic Keyword Classification and Information Retrieval*. Butterworths, London, 1971.
26. M. van Kreveld, I. Reinbacher, A. Arampatzis, and R. van Zwol. Distributed Ranking Methods for Geographic Information Retrieval. In *Proceedings of 11th Int. Sympos. on Spatial Data Handling: Developments in Spatial Data Handling*, pages 231–243.
27. Vicinity.com. http://home.vicinity.com/us/mappoint.htm.
28. E. M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In W. B. Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, pages 61–69. ACM/Springer, 1994.
29. W3C. Semantic Web. http://www.w3.org/2001/sw/, 2004.
30. J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11. ACM Press, 1996.

Security Ontology for Annotating Resources

Anya Kim, Jim Luo, and Myong Kang

Center for High Assurance Computer Systems,
Naval Research Laboratory, Washington, DC 20375
{kim, luo, mkang}@itd.nrl.navy.mil

Abstract. Annotation with security-related metadata enables discovery of resources that meet security requirements. This paper presents the NRL Security Ontology, which complements existing ontologies in other domains that focus on annotation of functional aspects of resources. Types of security information that could be described include mechanisms, protocols, objectives, algorithms, and credentials in various levels of detail and specificity. The NRL Security Ontology is more comprehensive and better organized than existing security ontologies. It is capable of representing more types of security statements and can be applied to any electronic resource. The class hierarchy of the ontology makes it both easy to use and intuitive to extend. We applied this ontology to a Service Oriented Architecture to annotate security aspects of Web service descriptions and queries. A refined matching algorithm was developed to perform requirement-capability matchmaking that takes into account not only the ontology concepts, but also the properties of the concepts.

1 Introduction

In today's network-centric computing environment, automatic discovery of resources and the ability to share information and services across different domains are important capabilities [1]. The first step in providing these capabilities is to markup these resources with various metadata in a well-understood and consistent manner. Such annotation will enable resources to be machine-readable and machine-understandable.

Using metadata to find distributed resources that meet one's functional requirements is only the first step. Resource requestors may have additional requirements such as security, survivability, or quality of service (QoS) specifications. For example, they may require resources to possess a certain military classification level, to originate from trusted sources, or to be handled according to a specified privacy policy. Therefore, resources need to be sufficiently annotated with security-related metadata so that they can be correctly discovered, compared, and invoked according to security as well as functional requirements of the requestor.

In this paper, we introduce a set of security-related ontologies collectively referred to as the NRL Security Ontology. The NRL Security Ontology provides the ability for precisely describing security concepts at various levels of detail. This ontology complements existing ontologies that mainly focus on functional aspects of capability, content, and parameters. Marking up security aspects of resources is a crucial step toward deploying a secure Service Oriented Architecture (SOA) system.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1483 – 1499, 2005.

Other groups have recognized the need for security annotation of services and proposed a set of security-related ontologies [2-4]. However, these ontologies possess certain limitations discussed in Section 2. The NRL Security Ontology was created to address these limitations. We expect this work to serve as a catalyst in the development of standardized security-related ontologies with contributions from both the security community and the semantic Web community.

The rest of the paper is organized as follows. Section 2 examines previous work in security ontology and discusses the need for improvements. Section 3 presents the NRL security ontology, including design objectives, domain and scope, and detailed descriptions. Section 4 gives examples of how to use these ontologies to annotate and query for resources particularly in a Web service context. It also discusses our algorithm for matchmaking between queries and resource descriptions. Section 5 presents future work and our conclusion.

2 Existing Security-Related Ontologies

Realization of the need for security ontologies is not new. Denker et al. have created several ontologies for specifying security-related information in Web services [2] using Daml+OIL [5] and later OWL [6]. We refer to this set of ontologies as the DAML Security Ontology for the rest of the paper. The authors state that the goal of these ontologies is to enable high-level markup of Web resources, services, and agents, while providing a layer of abstraction on top of various Web service security standards such as XML-Enc [7], XML-Dsig [8], and SAML (Security Assertion Markup Language) [9].

Of the set of ontologies that make up the DAML security ontology, the two main ontologies are the Security Mechanisms ontology and the Credential ontology. They describe security mechanisms and authentication credentials respectively. While we realize that these ontologies are works-in-progress and provide a great foundation for describing security-related concepts, we found two issues with them. First, they are not intuitive to understand especially in terms of the organization of subclass relationships. Second, they cannot express all the security information that we want to describe or be easily extended to do so.

The intuitiveness issue is particularly true for the main Security Mechanisms ontology. Figure 1 depicts this ontology in a simplified form where circles denote classes, solid lines represent instances of the classes and dotted lines represent properties[1]. The top class in this ontology is 'SecurityMechanism' with subclasses of 'SecurityNotation', 'Signature', 'Protocol', 'KeyFormat', 'Encryption', and 'Syntax'. Making these unrelated concepts sibling classes does not make sense from either a security perspective or an ontology perspective. Furthermore, some instances are not properly assigned to the correct subclass. For example, Kerberos and SSH are both declared as instances of 'KeyProtocol', however these are not key protocols. Additionally, all properties are defined for the top class. However, those properties do not apply to most of the subclasses. For example, no instance under the 'Syntax' subclass would have a need for the *relSecNotation* (Relative Security Notation), *enc* (Encryption), *sig* (Signature), or *reqCredential* (Required Credential) properties, yet they are all inherited because these properties are defined at the top class.

[1] Some complex concepts they use such as restriction classes are not depicted here.

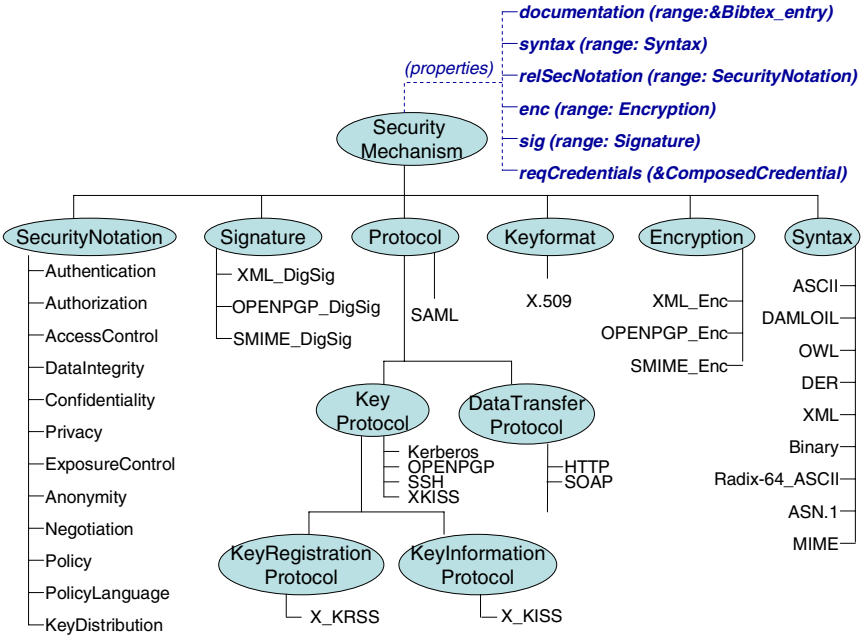


Fig. 1. Simplified DAML Security Mechanisms Ontology

The second issue we mentioned is the lack of expressiveness. The DAML security ontology includes many classes and instances that are not directly relevant for security annotation while lacking others that are necessary. For example, syntax and data transfer protocols are useful concepts in another domain, but are not particularly relevant for describing security-related information. Furthermore, the only encryption instances defined in the ontology are S/MIME, OpenPGP, and XML encryption. We do realize that more instances could be added as the need arises. However, the organization of the class hierarchy should be well developed. For example, there should be classes to represent military as well as commercial security devices and security policies. Currently, there is no appropriate place in the DAML Security Ontology to create a firewall or military security policy instance. There is also a lack of appropriately placed properties that could allow for more detailed refinement of security concepts. For example, it would be useful to define the algorithms supported by a protocol, or the certification status of a mechanism.

Although the authors of the DAML Security Ontology did a great job in recognizing the need for security ontologies and beginning work in security ontologies, we feel that there is still room for improvement. The next section describes the NRL Security Ontology in detail.

3 NRL Security Ontology

The DAML Security Ontology focuses on annotation of Web services rather than resources in general. This is evident not only from their documentation [2], but also

by examining the types of classes and instances in the ontology. We want ontologies that can be used to annotate generic resources from simple documents to interactive services with security-related metadata. We also want to improve upon the limitations of the DAML Security Ontology outlined in the previous section. The NRL Security Ontology was designed with the following objectives in mind:

1. Describe security related information applicable to all types of resources
2. Provide the ability to annotate security related information in various levels of detail for various environments (both commercial and military)
3. Create ontologies that are easy to extend and provide reusability
4. Facilitate mapping of higher-level (mission-level) security requirements to lower-level (resource-level) capabilities

3.1 Domain and Scope of the Ontology

When creating an ontology, one of the most important factors is the domain and scope in which it will be used [10]. While our objectives outlined above are a good starting point, in order to create ontologies that will be truly useful, we need to understand the types of questions that the ontology will be expected to answer.

These ontologies will be used by both the resource provider and the requestor to express their security requirements and capabilities. We must consider the various ways that the same statement can be expressed. Furthermore, we need to consider statements that are unlikely in order to limit the scope of the ontology. Statements that are either too broad or too specific are unlikely to be used and provide no useful information.

Noy et al. [10] state that one of the best ways to determine the scope of the ontology is to list a set of *competency questions* that can be answered using the ontology. For our purposes we did the same by composing a list of security requirements and capabilities for both the resource requestor and the provider. From the requestor's perspective, security requirements can be stated in terms of specific mechanisms or in terms of abstract security objectives. From the resource provider's perspective, security requirements are similar to the notion of policy and can express concepts such as authentication and access control. The provider's capabilities include protocols and mechanisms that the provider possesses and security policies it adheres to. The actual list of the requirements and capabilities statements we created can be found in the extended version of this paper [11].

3.2 Organizational Structure of NRL Security Ontology

We chose OWL to create our ontologies because it provides a rich vocabulary for describing classes and properties [6, 12]. It is widely used in many communities that have begun to develop ontologies of their own knowledge domains [13].

There are seven separate ontologies that make up the NRL Security Ontology:

1. Main Security ontology: an ontology to describe security concepts
2. Credentials ontology: an ontology to specify authentication credentials
3. Security Algorithms ontology: an ontology to describe various security algorithms
4. Security Assurance ontology: an ontology to specify different assurance standards
5. Service Security ontology: an ontology to facilitate security annotation of semantic Web services

6. Agent Security ontology: an ontology to enable querying of security information
7. Information Object ontology: an ontology to describe security of input and output parameters of Web services

The Service Security, Agent Security, and Information Object ontologies are based on some existing DAML Security ontologies while the others are new. The Credentials, Security Algorithms, and Security Assurance ontologies provide values for properties defined for concepts in the Main Security ontology. They enable those concepts to be described in more detail with respect to types of credentials used, supported algorithms, and associated levels of assurance. The Service Security ontology provides the means to use security concepts from the Main Security ontology in the Web services framework. The Agent Service ontology enables creation of security-related queries using security concepts from the Main Security ontology. The Information Object ontology allows for annotation of Web service inputs and outputs using the Security Algorithms ontology. The relationship among these ontologies is represented in Figure 2. The ontology depicted in gray represents OWL-S, a set of core ontologies used to describe Web services.

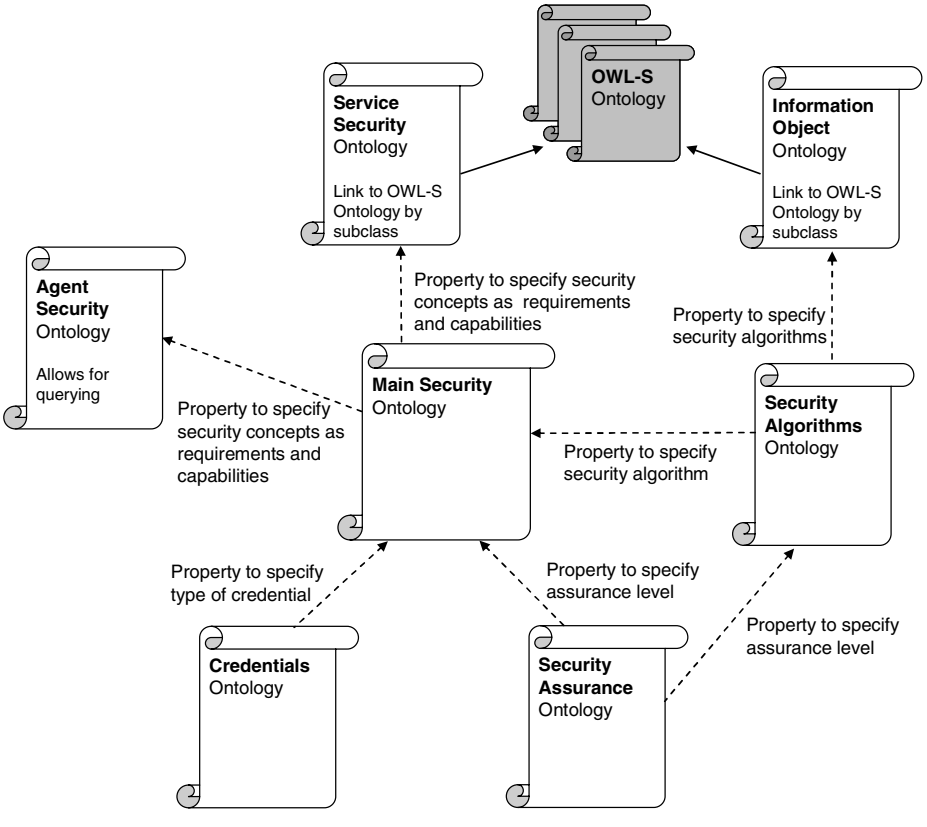


Fig. 2. Graphical Representation of Security-Related Ontologies and Their Relationships

Next, we present a brief explanation of classes, properties and relationships in each ontology. Due to space limitations we do not show all ontologies here. A complete graphical depiction of these ontologies and the OWL files can be found in [11].

Main Security Ontology (securityMain.owl). The core ontology in the NRL security ontology set is the Main Security ontology (Figure 3). It imports the Credentials ontology, Security Algorithms ontology, and Security Assurance ontology as object properties. The top class, 'SecurityConcept' possesses three subclasses: 'SecurityProtocol', 'SecurityMechanism' and 'SecurityPolicy'.

While some may argue that the distinction between security protocols and security mechanisms is blurred, we define security protocols as an agreed upon series of steps to accomplish a task while security mechanisms are implementations of protocols [14]. We specifically differentiate them here to provide the ability to describe security in both manners. Security policies are the set of rules that regulate how information is protected and secured .

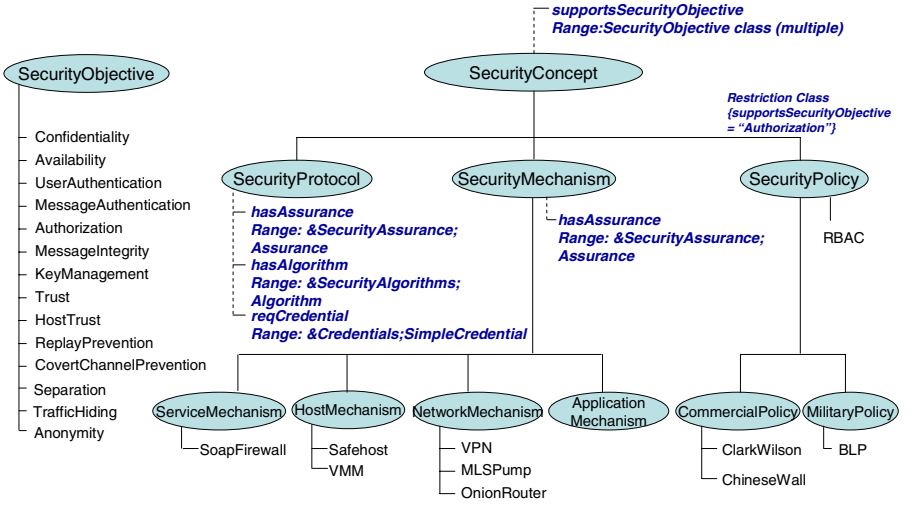


Fig. 3. A Part of the Main Security Ontology

The Main Security ontology also has a separate class called 'SecurityObjective' that enables users to specify security objectives for the 'SecurityConcept' class using the *suppotsSecurityObjective* property. For example, IPSec is declared to have Confidentiality, MessageAuthentication, and TrafficHiding as its *supportsSecurityObjective* property values. Security objectives also enable users to search for protocols, mechanisms, or policies based on the security objective they require. For example, users can query, "find all instances that provide confidentiality" and receive a list of all the security concepts that have a value of Confidentiality in their *supportsSecurityObjective* property.

Another way we can use 'SecurityObjective' is to map high-level mission requirements to low-level service requirements. For instance, assume that a security requirement is specified at the mission level such that Mission 1 and Mission 2 must have separation between them. At this level, the mission planner can use the ontology to specify the security objective of Separation. The mission designer can then search for instances in the 'SecurityConcept' class that provide Separation. In this case, the only one that does is VPN, so he can select VPN as a security requirement at the service level.

Credentials Ontology (credentials.owl). Authentication is one of the most fundamental security requirements in a networked environment. The Credentials ontology allows for specification of credentials used for authentication purposes (Figure 4). Concepts in the Security Main ontology can refer to a specific credential through their *reqCredential* property. While we adopted some of the notations in the DAML Credential ontology, we improved upon it by reorganizing classes to be more intuitive, including more properties and adding more classes to define additional types of credentials. Our Credentials ontology categorizes credentials into physical token, electronic token, and biometric token.

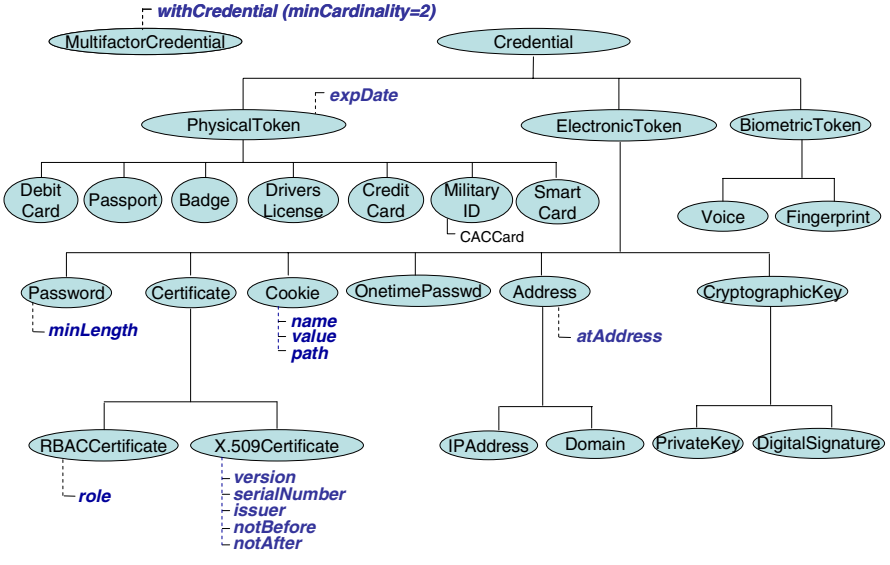


Fig. 4. Credentials Ontology

Under the 'PhysicalToken' class, we kept many of the classes from the DAML Credential ontology under their 'IDCard' class. In addition, we created a class to describe military IDs and an instance to represent CAC (Common Access Card) cards used in the military. The ontology can be extended to add properties such as issuing agency, expiration date, issue date, etc. Under the 'ElectronicToken' class, we

provide subclasses that enable authentication based on host address, certificates, passwords, and cryptographic keys to name a few. Additional properties were added to describe certificates including the issuer, version and serial number under the Certificate class. In order to support role-based (RBAC) certificates [15], an 'RBACCertificate' class was created as a subclass of the Certificate class with a *role* property. The 'BiometricToken' class represents credentials that pertain to human traits. For now, only 'Voice' and 'Fingerprint' subclasses are defined here.

In addition to the three categories of simple credentials, the 'MultifactorCredential" class can be used to describe composed credentials made up of two or more individual credentials. For example, it can describe requirements where both a smart card as well as a password is needed.

Security Algorithms Ontology (securityAlgorithms.owl). The Security Algorithms ontology was created to enable description of various security algorithms (Figure 5).

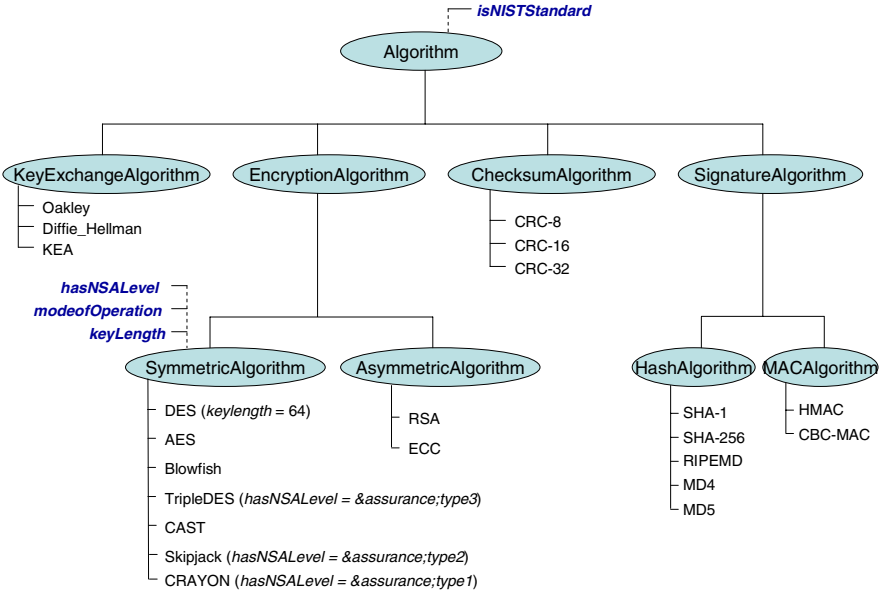


Fig. 5. Security Algorithms Ontology

Security Assurance Ontology (securityAssurance.owl). The Security Assurance ontology provides a way to describe standardized assurance methods for security protocols, mechanisms, and algorithms. They can be described in terms of their assurance level using the *hasAssurance* property from the Main Security ontology. The 'Assurance' class is classified according to different assurance methods: 'Standard', 'Accreditation', 'Evaluation', and 'Certification'. This ontology is the least compete of all our ontologies. However, we have added classes to describe the Common Criteria and TCSEC evaluations, and the FIPS and NSA standards [16].

Service Security and Agent Security Ontologies (serviceSecurity.owl and agentSecurity.owl). OWL-S [17] is an OWL-based semantic markup description language that provides a core set of constructs for describing Web services specifically. It provides a set of ontologies called Profile, Process, and Grounding to describe Web services. The Profile describes services in terms of what the service does, the Process describes how to use it, and the Grounding specifies how to interact with it.

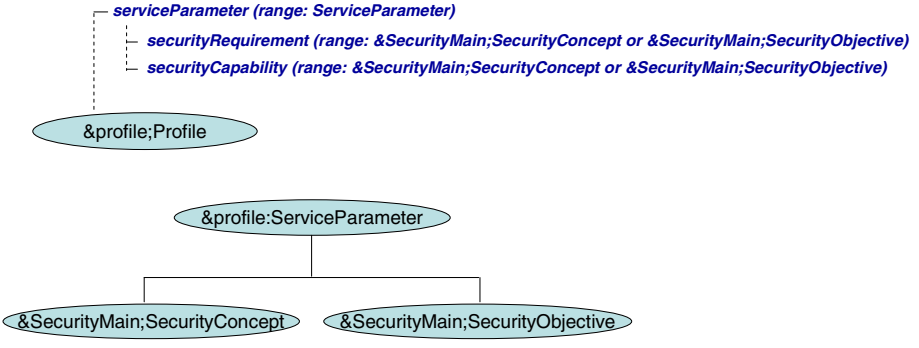


Fig. 6. Service Security Ontology

In order for the NRL Security Ontology to be used in the Web service context, a link must be made to the OWL-S ontologies. The Service Security ontology was developed for such a purpose. In the Service Security Ontology, 'SecurityConcept' and 'SecurityObjective' from the Main Security ontology are defined to be subclasses of the 'ServiceParameter' class in the OWL-S Profile ontology (Figure 6). The OWL-S Profile also contains a *serviceParameter* property that can have ServiceParameter as its value[2]. Declaring two subproperties of the *serviceParameter* property, *securityRequirement* and *securityCapability* enables the OWL-S Profile to include security requirements and security capabilities in its service description. Furthermore, we defined the range for these subproperties as either the 'SecurityConcept' or 'SecurityObjective' classes. This allows security requirements and capabilities to be stated in terms of either a particular security objective, or a specific security mechanism.

The Agent Security ontology allows for querying of resources, in particular Web services with requestor requirements and capabilities. It defines an 'Agent' class to represent the service requestor with the properties *securityCapability* and *securityRequirement* that can hold values from the 'SecurityConcept' and 'SecurityObjective' classes.

Information Object Ontology (InfObj.owl). The Information Object ontology is based on a DAML ontology created to capture encrypted or signed input/output data

[2] Note that the OWL-S Profile ontology has a property and class of the same name, service parameter. However, the property starts with a lowercase letter, while the class starts with an uppercase letter. Thus, *serviceParameter* refers to a property while ServiceParameter refers to a class.

of Web services. It has an 'InfObj' class and two subclasses, 'EncInfObj' (Encrypted Information Object) and 'SigInfObj' (Signed Information Object). The 'InfObj' class is used as the range for input and output parameters of services described with OWL-S. The ontology has the *cryptoAlgUsed* property to specify the algorithm used to encrypt or sign the object. In the original DAML ontology, the *cryptoAlgUsed* property pointed to a set of algorithms defined within the DAML Information Object ontology. However, we felt that the two concepts of information object and security algorithms were so dissimilar that they did not belong within the same ontology file. Hence, in the NRL Information Object ontology, the *cryptoAlgUsed* property points to classes in the Security Algorithms ontology.

3.3 Design Objectives Revisited

At the beginning of Section 3 we outlined a set of objectives expected to be achieved by the NRL Security Ontology. This subsection discusses whether those design objectives were met and to what degree.

1. **Describe security related information not only for Web services, but for all types of resources:** The NRL Security Ontology enables us to describe security information of various types of resources. We can describe security protocols that are specific to Web services such as XML-enc and SAML, but also include many protocols and mechanisms such as IPSec, Kerberos and SSH that are generally applied to any resource.
2. **Provide the ability to annotate security related information in various levels of detail for various environments:** The ontology can provide specific details of security mechanisms through properties such as the types of algorithms supported, required key length, types of credentials used, and expiration dates. Classes and instances were created that enable description of resources relevant to a military environment as well as for commercial use.
3. **Create ontologies that are easy to extend and provide reusability:** The ontologies are created with a class hierarchy that makes sense from a security perspective. New instances when necessary can be added to the ontology in an intuitive manner with out having to alter the class hierarchy.
4. **Facilitate mapping of higher-level (mission-level) security requirements to lower-level (resource-level) capabilities using the ontology:** resources can be described in terms of either security objectives at the abstract level, or security concepts at the concrete level. A mapping was established so that moving between the two methods of specification is possible.

In the next section, we will provide some examples of how to apply these ontologies to annotate resources with security information.

4 Application of NRL Security Ontology to a Service Oriented Architecture

While the NRL Security Ontology can be used to describe security-related information of resources in general, in this section we discuss how to annotate Web services in a Service Oriented Architecture. In particular, we focus on:

- How to annotate Web service descriptions with security requirements and capabilities
- How to create queries for finding Web services with given security requirements and capabilities
- How to perform matchmaking between queries and service descriptions in the SOA context.

4.1 Reasoning and Matching Algorithm

We have stated that both resource requestors and providers have security requirements and capabilities. Matchmaking looks for a two-way correspondence between these requirements and capabilities. In other words, service requirements are compared to requestor capabilities and service capabilities are compared to requestor requirements. In order for a match of security concepts to occur between a service provider and a service requestor, two conditions should be met. First, the provider's security capabilities should satisfy the requestor's security requirements. Second, the provider's security requirements should be satisfied by the requestor's security capabilities. This implies that the requirements should subsume the capabilities (Table 1).

Table 1. The Matching between Requestor and Provider Requirements and Capabilities

Requestor		**Provider**
Requirements	$\subseteq$	Capabilities
Capabilities	$\supseteq$	Requirements

Every single requestor requirement must have a corresponding capability on the provider side to satisfy it, and vice versa. Hence the matchmaker must be able to perform two tasks. First, it must be able to determine the level of match for each specific requirement and a specific capability. Second, it must use those levels of match to determine if the set of requirements is matched by the set of capabilities. In other words, the matchmaker must determine the level at which each requirement is matched to a capability, and then the overall level of match between the requester and the provider. This will be explained in detail later in the section.

Several semantic matching algorithms have been proposed [2, 18, 19]. Two of these [18, 19] support only one-way matching of functional service descriptions to requestor queries as opposed to requirement-capability matching. They do not need to consider two-way matching since their focus is on matching functional aspects; when discussing purely functional requirements there is no functional requirement from the provider-side and no functional capabilities on the requestor-side. The third proposed matchmaking algorithm [2] performs requirement-capability matching for both sides. However, it does not take into account property attributes. Consequently, it will not support cases where both the requirement and capability point to the same concept but the concepts are annotated with different properties. For example, the requestor and provider may both use SSH (stated as a requirement on one side and a capability on the other), but if the requestor requires SSH using TripleDES and the provider is only capable of SSH with AES then these two should not match. Our matchmaker will perform requirement-capability matching, taking into account property annotations.

Specifically, when describing security information of resources, the ability to include properties in the matching algorithm is very important. This is due to the fact that security information, more so than functionality-related information can require detailed descriptions that make extensive use of properties. Complex statements can be made with multiple layers of properties. For example, there could be a security requirement that requires the use of XML-enc (*securityRequirement* property) with a symmetric encryption algorithm (*hasAlgorithm* property) that has been declared a type 3 algorithm from the NSA (*hasNSALevel* property).

For the first task of the matchmaker, there are four possible levels of match for each requirement-capability pair: perfect match, close match, possible match, and no match in decreasing order of matching.

Perfect Match cases. Perfect matches occur when both one's capability and the other's requirement point to the same concept. The same concept can mean the exact same concept, or two concepts declared as equivalent in the ontology. There are two ways this can occur:

- Case 1. Both the requirement and capability specify the exact same ontology concept. The instances and property values specified by both sides are identical. This is the trivial case. For example, if a requestor query states that it requires the service to possess a VPN (Virtual Private Network) that possesses a Common Criteria EAL4 rating and a service describes its capability as possessing a VPN with a Common Criteria rating of EAL 4 then these two are a perfect match.
- Case 2. The requirement and capability refer to equivalent concepts, and if properties are specified, the properties are identical or equivalent. For example, a requestor's requirement specifies SSL and the provider's capability is listed as TLS. In the Main Security ontology, these two concepts are listed as equivalent classes; hence they are identical and will produce a perfect match. We sometimes call this an equivalence match to differentiate from the first case.

Close Match cases. A close match occurs when one's requirement is more general (i.e., described in less detail) than the other's capability. There are three ways this can occur:

- Case 1. The requirement specifies a more general concept at a higher level in the ontological hierarchy. For example, the requestor's capability is stated as DES while the provider's requirement asks for a symmetric encryption algorithm. DES is an instance of the 'SymmetricAlgorithm' class and thus lower in the hierarchy. We assume that the provider specified its requirement as a higher level concept because it does not care which specific algorithm is used as long as it is a symmetric encryption algorithm. Therefore, we can assume a match.
- Case 2. The requirement and capability have the same concept, but the capability is specified in more detail (i.e., property). For example, the requestor's capability is specified as AES with 256 bit keys while the provider's requirement asks for AES (with no properties). AES with 256 bit keys is a more specific instance of AES so we can assume that there is a match.
- Case 3. The requirement is stated in terms of a security objective while the capability is stated in terms of a security concept that supports that specific objective. For example, the requestor's requirement is stated as the objective of

Confidentiality and the provider's capability is given as XML-Enc which has the *supportsSecurityObjective* value of Confidentiality. Since the requirement is looking for anything that supports Confidentiality and XML-Enc does support it, we view this as a match.

Possible Match cases. A possible match occurs when one's requirement is more specific (i.e., defined in more detail) than the other's capability. This is the opposite of a close match. A possible match does not rule out the possibility of a match, but the information available cannot ensure the capability can match the requirement. There are three ways this can occur:

- Case 1. The requirement specifies a more specific concept (lower in the hierarchy). For example, the requestor's capability is stated as symmetric encryption algorithm while the provider's requirement asks for DES. The symmetric encryption algorithm that the requestor is capable of could be DES, but it is not certain. Therefore, it is only a partial match.
- Case 2. The requirement and capability refer to the same concept, but the requirement specifies a more refined concept (i.e. property). For example, the capability is stated as AES while the requirement asks for AES with 256-bit keys. The AES specified in the capability could be possible of 256-bit key encryption, but it is not certain. Therefore, it is only a partial match.
- Case 3. The requirement is stated in terms of a security concept while the capability is stated in terms of a security objective that is supported by the security concept. For example, the requestor's requirement is stated as confidentiality while the provider's capability is stated as XML-Enc which supports confidentiality. The requestor may be capable of using XML-Enc, but it is not certain. All we can deduce is that the requestor is capable of confidentiality. Therefore, it is only a partial match.

No Match cases. No match occurs when one's capability and the other's requirement are disparate without the possibility of matching. There are two ways this can occur:

- Case 1. The requirement and capability point to two unrelated concepts. For example, the requirement states it requires DES and the capability states its capability as RSA. These concepts have no hierarchical relationship to each other and so are unrelated. There can be no match.
- Case 2. The requirement and capability point to the same concept but have different specifics (i.e. properties) with respect to that concept. For example, the requirement points to AES in CBC mode while the capability states AES in CFB mode. The capability and requirement can both use AES, but they require modes of operation; one is a block cipher the other is a stream cipher so they are not compatible.

For the second task of the matchmaker, it must attempt to match every requirement on one side against every capability on the other side. The degree of match for a single requirement is its highest level of match it has against all of the possible capabilities. The overall level of match between the requester and the provider is the same as the lowest degree of match of any of the requirement-capability pairs. There are four possibilities:

- If at least one of the requirements is not matched, then the requestor is not matched to the provider. The requestor will not be able to use the resource.
- If all the requirement-capability pairs are at least possible matches, then there is a possible match between the requester and the provider. This means there is not enough information to determine one way or the other whether the requester can use the resource. Additional information or negotiation will be needed to make that determination.
- If all the requirement-capability pairs are at least close matches, then the requestor can indeed use the resource.
- If all the requirement-capability pairs are perfect matches, then obviously the requestor can use the resource.

In the following section, we will provide an example of the matching process between a service description and a query.

4.2 Application of the Matching Algorithm

In this section we examine how to actually describe services and create queries using the security ontologies, and how to find services using the matching algorithm. In our example, we have a service requestor looking for a book selling service. The service requestor would create queries to find services that match not only the desired functionality, but also the security capabilities and requirements of the requestor.

The following is an example of the requestor's security capabilities and requirements along with the part of their query that pertains to the security capability and requirements:

Requestor's Security Capability
1. Authentication via SAML with an X.509 Certificate signed by VeriSign
Requestor's Security Requirement
1. Authorization
2. SSH with the DES algorithm in CBC mode

```
<credential:X.509Certificate rdf:ID="X.509">
  <credential:issuer rdf:resource="VeriSign"/>
</credential:X.509Certificate>
<securityMain:SAML rdf:ID="Capability1">
    <securityMain:reqCredentials
    rdf:resource="&credential;X.509"/>
</securityMain:SAML>
<securityMain:Authorization rdf:ID="Requirement1"/>
<securityAlgorithms:DES rdf:ID= "Alg">
  <securityAlgorithms:modesOfOperation rdf:resource="CBC"/>
</securityAlgorithms:DES>
<securityMain:SSH rdf:ID="Requirement2">
    <securityMain:hasEncryptionAlgorithm
    rdf:resource="&securityAlgorithms;Alg1"/>
</securityMain:SSH>
<agent:Agent rdf:about="#BookRequest">
  <securityCapability rdf:resource="#Capability1"/>
  <securityRequirement rdf:resource="#Requirement1"/>
  <securityRequirement rdf:resource="#Requirement2"/>
</agent>
```

On the other hand, a book selling service would create an OWL-S profile that includes its functional capabilities, as well as security requirements and capabilities. The following is the example security capability and requirement statements of the book selling service (BookSeller), along with the part of its OWL-Profile that would contain these statements.

BookSeller's Security Capability
1. SOAP Firewall with a Common Criteria level of EAL4
2. SSH with DES

BookSeller's Security Requirement
1. Authenticate via SAML with an X.509 Certificate

```
<securityMain:SOAPFirewall rdf:ID="Capability1">
    <securityMain:hasAssurance rdf:resource="&assurance;EAL4"/>
</securityMain:SOAPFirewall>
<securityMain:SSH rdf:ID="Capability2">
    <securityMain:hasEncryptionAlgorithm
    rdf:resource="&securityAlgorithms;DES"/>
</securityMain:SSH>
<credential:X.509Certificate rdf:ID="X.509"/>
<securityMain:SAML rdf:ID="Requirement1">
    <securityMain:reqCredentials
    rdf:resource="&credential;X.509"/>
</securityMain:SAML>
<profile:Profile rdf:about="#BookSeller1">
<profile:serviceName>BookSeller1</profile:serviceName>
<profile:textDescription>
         This service sells all types of books
</profile:textDescription>
  <securityCapability rdf:resource="#Capability1"/>
  <securityCapability rdf:resource="#Capability2"/>
  <securityRequirement rdf:resource="#Requirement1"/>
</profile:Profile>
```

Given this service description and the above query, the matching algorithm would match the requestor's capabilities to the provider's requirements and the requestor's requirements to the provider's capabilities in the following manner (Tables 2 and 3):

Table 2. Matching Requestor's Capabilities to Provider's Requirements

Requestor Security Capability	Provider Security Requirement	Match Level
Authentication via SAML with an X.509 Certificate signed by VeriSign	Authentication via SAML with an X.509 Certificate	Close Match

Table 3. Matching Requestor's Requirements to Provider's Capabilities

Requestor Security Requirement	Provider Security Capability	Match Level
Authorization	SOAP Firewall with Common Criteria level EAL4	Close Match
SSH with DES algorithm in CBC mode	SSH with DES algorithm	Possible Match

- In Table 2, the requestor's capability and the provider's requirement possess the same concepts, but the capability has more detail. This is Case 2 of the close match situation.
- In the first row of Table 3, the requestor's requirement was that a service provides Authorization. While the security objective of authorization is not explicitly stated in the OWL-S Profile of the provider, the reasoner was able to deduce that the SOAP Firewall supports authorization since it has a value of Authorization in its supportsSecurityObjective property. This is Case 3 of the close match situation.
- In the second row of Table 3, the requestor has a more detailed requirement regarding SSH than the provider has specified as its capability. This is Case 2 of the possible match situation. This could mean that either the provider cannot support the CBC mode of DES or it can support DES in CBC mode but decided not to provide this additional detail.

Since the lowest level of match in the three sets of requirement-capability pairs is possible match, the matchmaker will declare the service to be a possible match. The requester is not certain whether it can use the service. It must obtain additional information or negotiate with the provider to make that decision.

5 Conclusion and Future Work

Annotating resources with metadata enables them to be machine-understandable and facilitates automatic discovery and invocation. Most work in the area thus far has focused on annotation of resources in terms of functionality. However, security is an important issue especially in a network-centric environment. Most resources on the network are protected by some sort of security mechanisms. Satisfying functional requirements alone may not guarantee access to desired resources. As a result, annotation of resources in terms of security is just as important as annotation in terms of functionality.

In this paper, we presented the NRL Security Ontology for making security annotations. It is much more comprehensive than security ontologies previously available in terms of the number of concepts, the properties of the concepts, and the type of resources that can be described. Its organization is also more intuitive so that it is easier to use as well as to extend. New properties and instances can be added without modifying the overall class hierarchy. We demonstrated how the ontology can be applied to the context of Web services in a Service Oriented Architecture to describe security capabilities and requirements. A matchmaking algorithm was presented to perform requirement-capability matchmaking that takes into account not just the concepts, but also the properties of the concept. This is important because security annotations make extensive use of property attributes. The ability to take them into account makes this matching algorithm much more refined than previous work.

The creation of these ontologies is an iterative process. Additional instances and properties will always be needed to express new security statements. Classes and properties may be added and deleted as the security community continues to evaluate and refine the security ontologies. Additional ontologies are still needed to address issues such as privacy policies, access control, survivability, and QoS. We hope this work will serve as a catalyst in the development of standardized security-related ontologies with contributions from both the security community and the semantic Web community.

References

1. IA Architecture and Technical Framework (2004). Executive Summary of the End-to-End IA Component of the GIG Integrated Architecture, National Security Agency Information Assurance Directorate.
2. Denker, G., Kagal, L., Finin, T., Paolucci, M., and Sycara, K. (2003). Security for DAML Web Services: Annotation and Matchmaking. In *Proc. of the 2nd International Semantic Web Conference (ISWC2003)*: Sanibel Island, Florida.
3. Denker, G., Nguyen, S., and Ton, A. (2004). OWL-S Semantics of Security Web Services: a Case Study. In *1st European Semantic Web Symposium*: Heraklion, Greece.
4. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., and Sycara, K. (2004). Authorization and Privacy for Semantic Web Services. In *AAAI Spring Symposium, Workshop on Semantic Web Services*: Stanford, California.
5. W3C (2001). DAML+OIL (March 2001) Reference Description, http://www.w3.org/TR/daml+oil-reference.
6. W3C (2004). OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/.
7. IETF and W3C Working Group (2001). XML Encryption, http://www.w3c.org/Encryption/2001.
8. IETF and W3C Working Group (2003). XML Signature, http://www.w3c.org/Signature.
9. OASIS SSTC (2005). Security Assertion Markup Language (SAML) 2.0 Technical Overview, Working Draft, http://www.oasis-open.org/committees/download.php/12938/sstc-saml-tech-overview-2.0-draft-06.pdf.
10. Noy, N.F., and McGuinness, D.L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory, KSL-01-05.
11. Kim, A., Luo, J., and Kang, M. (2005). Security Ontology for Annotating Resources. pp. 51, Naval Research Lab, NRL Memorandum Report, NRL/MR/5540-05-641: Washington, D.C.
12. W3C Recommendation (2004). OWL Web Ontology Language Guide, vol. 2005, W3C.
13. DAML Ontology Library. http://www.daml.org/ontologies/.
14. Schneier, B. (1996). Applied Cryptography, 2nd Edition (New York: John Wiley and Sons, Inc.).
15. Ferraiolo, D.F., Kuhn, D.R., and Chandramouli, R. (2003). Role-Based Access Control (Norwood, MA: Artech House).
16. Committee on National Security Systems (2003). National Information Assurance (IA) Glossary. pp. 85, http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf: Ft. Meade, MD.
17. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2003). OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/.
18. Jaeger, M., and Tang, S. (2004). Ranked Matching for Service Descriptions using DAML-S. In *Enterprise Modelling and Ontologies for Interoperability (EMOI), INTEROP 2004*: Riga, Latvia.
19. Srinivasan, N., Paolucci, M., and Sycara, K. (2004). Adding OWL-S to UDDI, Implementation and Throughput. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*: San Diego, California.

An Ontology for Mobile Agents in the Context of Formal Verification

Paulo Salem da Silva and Ana Cristina Vieira de Melo

University of São Paulo, Department of Computer Science, São Paulo – Brazil
salem@linux.ime.usp.br, acvm@ime.usp.br

Abstract. The increasing interest of developing and formally studying mobile agents has taken place in the last decade. Several formalisms and tools have been created to aid this enterprise. However, tools are mostly developed in isolation and, therefore, are hard to use together. The present work is an attempt to make such integration easier, through the provision of a common 'language' – an ontology – for verification tools reasoning about mobile agents.

1 Introduction

The interest for the development and formal study of mobile agents has grown in the last decade. With the creation of theories designed for this purpose, such as π-calculus [2, 21], Distributed Join-Calculus [20, 19] and Ambient Calculus [17], it has become possible to build software capable of reasoning about properties related to a given specification. These so called verification tools usually have a base theory and language, as well as some equivalence definitions and proof system.

Even for a single theory, verification tools are frequently developed in isolation and, thus, with different capabilities and notations. When users need to use various verification tools to solve a problem, they have to learn and use each verification tool individually. Many times, users prefer to partially solve their problems instead of learning various verification tools. On the other hand, new verification tools are created from scratch and, then, reimplement services already available in existing tools, instead of reusing them.

We believe that these problems can be partially solved. Since verification tools differ only in their conventions, but not in their application domains, it is reasonable to assume that it is possible to create a *common* 'language' on the top of the native 'languages' of *each* verification tool. Such common 'language', together with the relation among its elements, is called an ontology [3, 4]. From a general point of view, an ontology is a language that specifies an application domain in a way that different tools may exchange information about it. In our case, an ontology for mobile agents verification tools must describe not only the domain of these agents, but also the capabilities of the verification tools. In other words, it must describe the objects being studied (i.e., mobile agents) as well as what the verification tools can reason about (i.e., the class of properties and equivalences the verification tools may check).

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1500–1516, 2005.

Establishing an ontology for mobile agents can help in integrating existing verification tools. It can be used to denote the language and capabilities of verification tools in a mobile agents environment:

- verification tools may be registered into the system using an ontology;
- users[1] may find an appropriate verification tool among the registered ones using ontology based search criteria.

In this paper, we present an ontology for the mobile agents domain, the first step toward the system described above. The ontology describing the verification tools capabilities remains as work in progress and, thus, it is not presented here.

Mobile agents and, therefore, mobility itself, may be defined in several ways. In our work, we use the π-calculus theory and tools as the mobility paradigm. However, as it will soon become clear, our ontologies are extensible in order to work with other mobility theories.

At last, we point out that there are other formal approaches to agent modeling, such as the SMART framework [5]. However, as far as we know, these models have not been used to foster verification tools interoperability.

In the remainder of this paper, we present the notations and tools we used (Sect. 1.1), the ontology itself (Sect. 2) and an usage example (Sect. 3). We also discuss our results and the work that remains to be done (Sect. 4).

1.1 Notation and Tools

We have chosen the Web Ontology Language (OWL) [12] as the notation for our ontologies. We had considered other options, such as UML, but found them to be very limited. OWL, on the other hand, provides a rich logic language based on Description Logic [13] as well as supporting tools, since it is becoming a mainstream industry[2] resource.

Among existing tools for OWL, we opted for Protégé-2000 [14]. Protégé not only supports OWL design, but also provides an API (Application Programming Interface) that allow external software to reuse many of its resources. This characteristic might be useful at a later stage of our work.

To make easier understanding the ontology, in this paper we present a plain English description of it, as well as pictures generated using the OntoViz [15] Protégé plugin. The formal OWL definitions, along with the Protégé project files, can be obtained in the following URL:

`http://www.ime.usp.br/~salem/papers/mobile_agent_ontology.zip`

2 The Mobile Agent Ontology

The purpose of our mobile agent ontology is to describe the possible components of a mobile agent and its environment, and to provide a way to describe actual formalisms structures, such as those from π-calculus.

[1] Both humans and other software systems.

[2] OWL is a new W3C recommendation and W3C is mainly industry oriented.

Notice that the ontology does not aim to describe the actual structure of agents, this would be equivalent to create a new theory for mobility analogous to the π-calculus. Our aim here is to describe the properties of agents structure (e.g., elements that can be used to build an agent, not the actual blueprint of the agent).

Tools using this ontology to exchange information about agents are benefited because:

- They only need to represent the search criteria once, using the ontology. Without the ontology, it could be necessary to specify several search criteria, each one designed to match the proprietary notation employed by each tool it is trying to communicate with;
- They may be developed using other theories rather than π-calculus, as long as the structures from these theories can be mapped into our ontology;
- They may provide a more user-friendly description of agents, since the ontology is designed at a high abstraction level.

The ontology is divided in two subontologies:

- the high level description of the mobile agent domain;
- the description of the formalisms employed by the tools;

A mapping of the first onto the second subontology is also provided (Sect. 2.3).

In the remainder of this section we shall present both subontologies and the mapping in details.

2.1 Subontology 1: Mobile Agents

This subontology describes the domain of mobile agents. It is based on the 'agent' concept found in [1], which, in short, defines an agent as an entity having sensors, actuators and that exists in an environment.

The scope of our ontology is restricted to the *formal methods* area and to an special kind of agent, the *mobile* one. Therefore, there are two important aspects which make our ontology unique:

- First, there are several characteristics from the agents domain which are not useful to us. For instance, it is not of our interest to define 'rationality', since it is not clear if such concept would be useful for formal verification. We have, thus, excluded concepts that are either irrelevant or obscure;
- On the other hand, the general theory about agents found in [1] lacks several elements that could be useful for an appropriate representation of mobility. For example, we crafted the concept of 'message' as a perception.

Now we will describe the classes of this ontology. To make the concepts clearer, we will first consider a cell phone as an example of mobile agent and show how this device is described by the ontology.

Example: Cell Phone. Cell phones are classical examples of mobile agents. To understand why, let us consider the fundamental idea behind them. If you are already familiar with that, you may wish to skip this section.

In order to enable cell phone operation in a certain region, the region is divided in several subregions, or cells. Each cell provides a connection point to the several cell phones inside its area. This access point is usually a station with several antennas to which the cell phones may connect using ordinary radio waves. Once connected, the station takes care to establish communication with the rest of the telecommunication network. See Fig. 1.

Mobility arises when a cell phone changes the cell it is in. When this happens, the phone must terminate the connection to the previous station and start one with the station covering the new cell. See Fig. 2

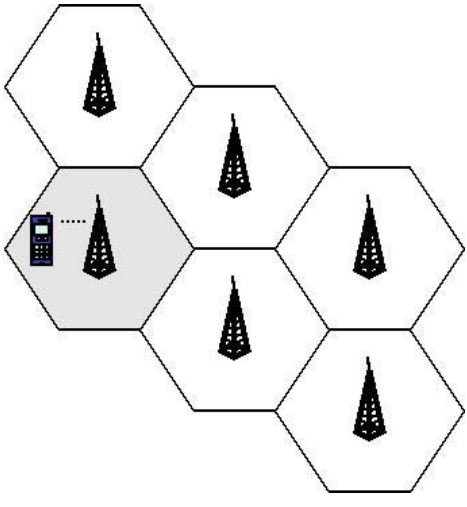


Fig. 1. Cell division of a region. The highlighted cell shows a cell phone connected to the cell's station.

Ontology Classes. As we pointed out earlier, the description of the ontology given in this paper is in plain English.

Fig. 3 depicts the ontology graphically.

Ontology class 1 (Agent). *Agents are entities which perceive and act in a particular environment. In our example, the cell phone itself belongs to this class. They have the following properties.*

hasSensors. *Zero or more instances from the Sensor class.*
hasActuators. *Zero or more instances from the Actuator class.*
isReconfigurable. *Boolean value. May the agent change its internal state during its existence? The cell phone, for instance, is reconfigurable, since at any given time it may be connected to a different antenna and be performing calls to different phones.*

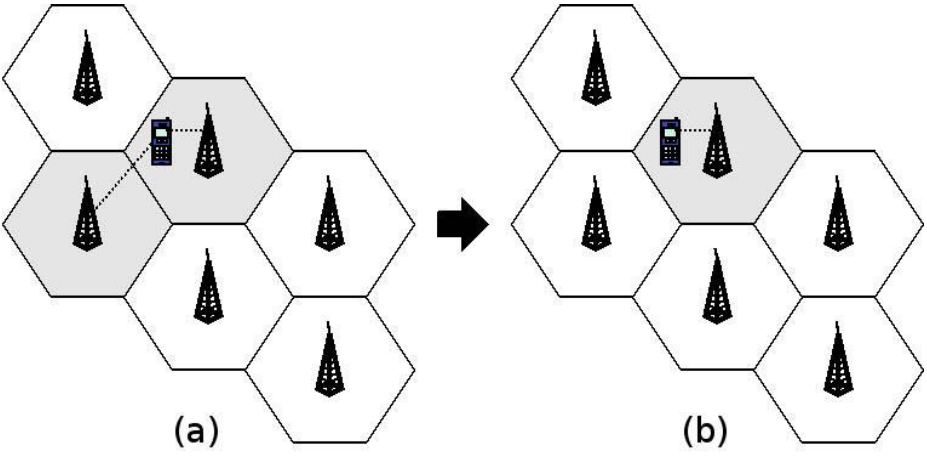


Fig. 2. Moving into another cell: (a) the phone moves physically into another region, preparing the connection with the new station without leaving the old connection; (b) the transition is complete and the phone is connected only to the new station

isDeterministic. *Boolean value. Does the current state of affairs determine the next action of the agent? The cell phone might be modeled either as a deterministic or nondeterministic agent. If we consider the phone as an agent that answers to its user's commands, then it is reasonable to consider it deterministic, for its behavior would not be random, it wouldn't, for instance, call a number unless it was requested to do so. On the other hand, if user interaction was hidden (i.e., implicit), then the phone could be seen as nondeterministic, since the user's behaviors (e.g., making calls) would provide a source of randomness.*
isCloneable. *Boolean value. Can the agent produce a copy of itself or of its internal parts? The cell phone is not cloneable, since it cannot duplicate itself.*
hasEnvironment. *One instance from the Environment class.*

Ontology class 2 (Percept). *A Percept is that which an Agent may sense through its sensors. The Percept class is abstract, and concrete subclasses must be defined. In the cell phone example, the percepts are the magnetic waves that reach the device through its antenna.*

Ontology class 3 (Message). *Message class is a concrete subclass of the Percept class. A Message is a piece of information that can be transmitted among agents. In the cell phone, for instance, if the user's voice is transmitted using discrete data packets, we could say that such packets belong to Message class. Notice, though, that if there are no discrete packets, if data flows continuously, then Message class could not be used to classify this transmission. Another Percept subclass would have to be created to account for that.*

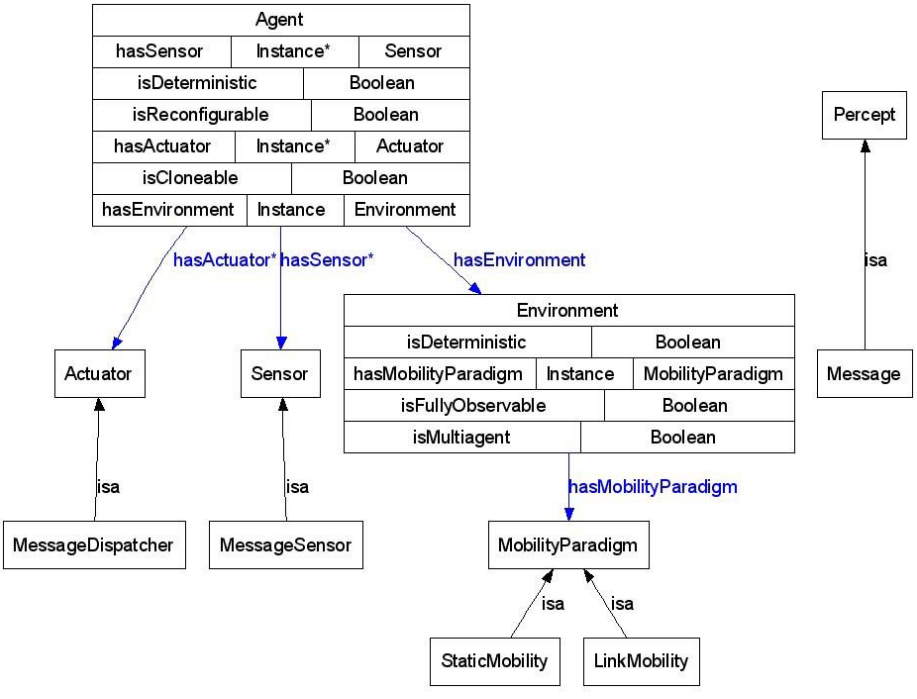


Fig. 3. Mobile agents domain ontology

Ontology class 4 (Sensor). *A Sensor is that which may receive Percepts. Sensor is an abstract class and concrete subclasses must be defined for each concrete subclass of Percept that are supposed to be perceived.*

Ontology class 5 (MessageSensor). *MessageSensor is a concrete subclass of the Sensor class. A MessageSensor is a Sensor that receives Messages. For instance, in the cell phone, the subsystem responsible for receiving and decoding the data packets could belong to MessageSensor class.*

Ontology class 6 (Actuator). *An Actuator is that which can perform actions over the Environment or other Agents. Actuator class is abstract and concrete subclasses must be defined for each action that the Agent may perform.*

Ontology class 7 (MessageDispatcher). *MessageDispatcher is a concrete subclass of the Actuator class. A MessageDispatcher is an Actuator that sends messages. In the cell phone example, the subsystem responsible for packing and sending data could belong to MessageDispatcher class.*

Ontology class 8 (Environment). *An Environment is where Agents exist. Cell phones, for example, exist in an environment of 'cells' (thus, the device's name). That is, physical space is divided into cells, and in each cell one or more*

antennas provide a connection point to the phones. From the cell phone stand point, the environment is nothing more than the antennas that it can connect to.

Environment class has the following properties.

isFullyObservable. *Boolean value. Can the Agent always sense the whole Environment? Cell phones cannot, since they are aware only of the antennas in the current cell.*

isDeterministic. *Boolean value. Given the Environment's current state and the Agents actions, is the next Environment's state determined? For cell phones, since hardware may fail, it is best to assume that the Environment is not deterministic.*

isMultiagent. *Boolean value. Can the Environment support more than one Agent? That's clearly true for cell phones.*

hasMobilityParadigm. *An instance from the MobilityParadigm class. How is mobility defined in this Environment?*

Ontology class 9 (MobilityParadigm). *A MobilithParadigm defines how mobility is handled by an Environment. MobilityParadigm class is abstract and concrete subclasses must be defined.*

Ontology class 10 (StaticMobility). *StaticMobility is a concrete subclass of MobilityParadigm. It defines the state of affairs in which no mobility at all takes place.*

Ontology class 11 (LinkMobility). *LinkMobility is a concrete subclass of MobilityParadigm. It defines the state of affairs in which the spatial position of an agent is given by its links to other agents, without any notion of distance. That's precisely the notion of mobility that a cell phone agent has, since links to antennas are all that such agents know about their position.*

2.2 Subontology 2: Formalisms

This subontology aims at describing the actual formalism structures that the verification tools employ. The description is given considering two different levels of abstractions. The first one defines abstract elements of any process calculus (e.g., operators, actions), while the second one specifies the elements of a particular agent calculus, the π-calculus. Graphical representations are provided.

Though we focus on the π-calculus, we believe that the ontologies can be extended to other mobility formalisms. In particular, because many of these formalisms, such as Join-Calculus [20, 19] and Ambient Calculus [17], can be proved equivalent to the π-calculus.

Ontology Classes. The main classes of this subontology are the following (see Fig. 4).

Ontology class 12 (Formalism). *Formalism is an abstract class. Actual formalisms must be represented by concrete subclasses.*

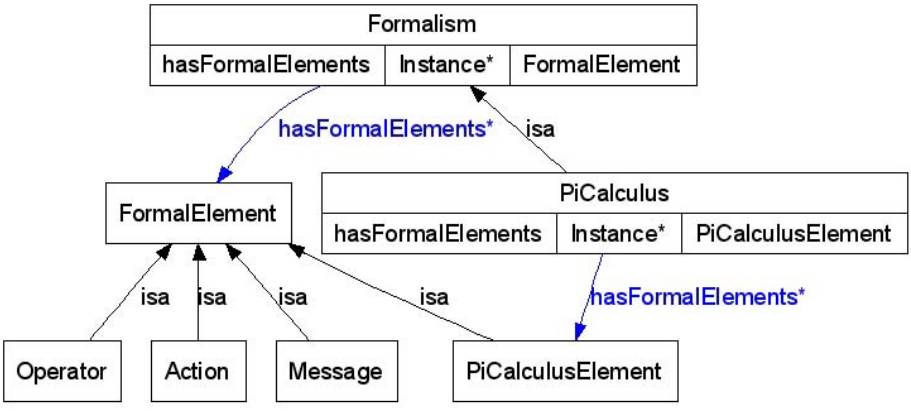


Fig. 4. Main formalism ontology classes

hasFormalElements. *Zero or more instances from the FormalElement class.*

Ontology class 13 (PiCalculus). *PiCalculus is a concrete subclass of Formalism class. It represents the π-calculus theory itself.*

hasFormalElements. *Zero or more instances from the PiCalculusElement class.*

Ontology class 14 (FormalElement). *FormalElement is an abstract class. It represents the syntactic building blocks of formalisms. This class must have concrete subclasses for each Formalism subclass.*

Ontology class 15 (Action). *Action is an abstract class. It represents the elements over which operations can be performed. That is, they are the atomic units that the process calculus works with.*

Ontology class 16 (Message). *Message is an abstract class. It defines what is sent through OutputActions and received by InputActions.*

Ontology class 17 (Operator). *Operator is an abstract class. Operators perform transformations over Actions.*

A class to group together all elements from π-calculus is also provided.

Ontology class 18 (PiCalculusElement). *PiCalculusElement is an abstract subclass of FormalElement. It represents the syntactic elements found in the π-calculus.*

Action, Message and Operator classes are the main abstract concepts that we use to describe a general process calculus. Each of these classes, in turn, have their own subclasses and, finally, these subclasses have actual concrete classes that represent an actual process calculus, the π-calculus.

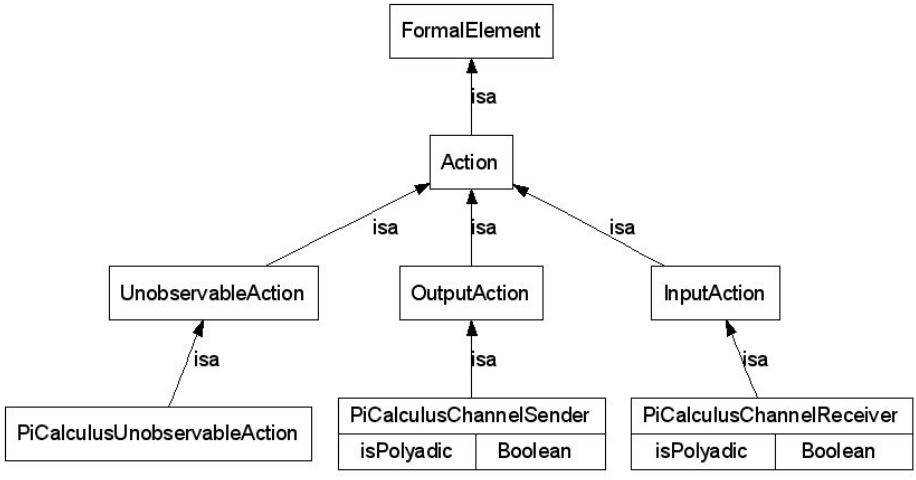


Fig. 5. Action subclasses

Ontology class 19 (InputAction). *InputAction is an abstract subclass of Action. It represents Actions that take some input.*

Ontology class 20 (OutputAction). *OutputAction is an abstract subclass of Action. It represents Actions that send some output.*

Ontology class 21 (UnobservableAction). *UnobservableAction is an abstract class. It represents Actions that do not affect other Actions.*

Ontology class 22 (ChoiceOperator). *ChoiceOperator is an abstract class. ChoiceOperators are Operators that given two expressions, output one of them in a nondeterministic manner.*

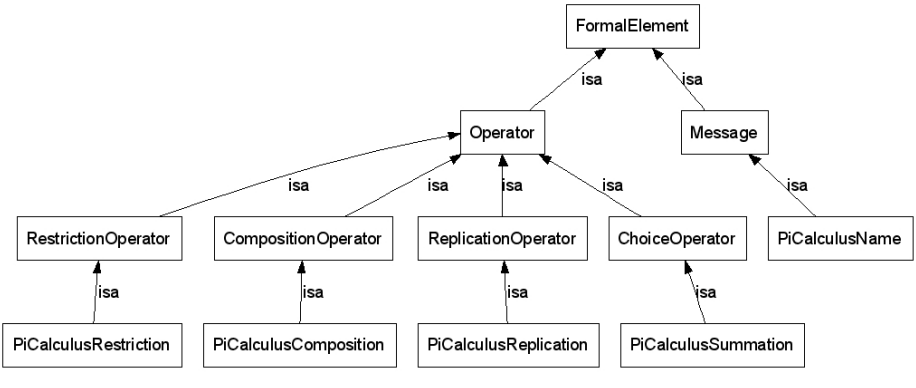


Fig. 6. Operator and Message subclasses

Ontology class 23 (CompositionOperator). *CompositionOperator is an abstract class. CompositionOperators are Operators that allow two expressions interact.*

Ontology class 24 (ReplicationOperator). *ReplicationOperator is an abstract class. ReplicationOperators are Operators that create copies of an expression.*

Ontology class 25 (RestrictionOperator). *RestrictionOperator is an abstract Class. RestrictionOperators are Operators that bind an identifier to a particular expression.*

The other classes are subclasses of PiCalculusElement and of some FormalElement. Each represents a particular construction from the π-calculus theory. Their names are self-explaining, but we shall list them here for the sake of completeness.

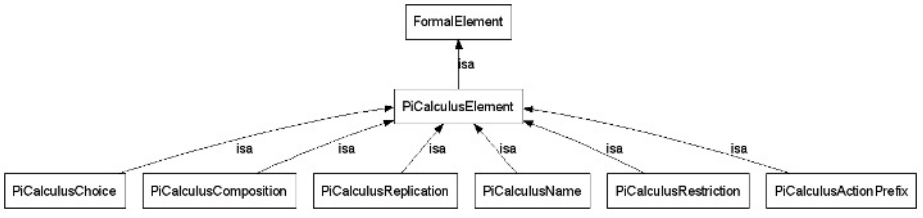


Fig. 7. π-calculus elements

Ontology class 26 (PiCalculusName). *Concrete subclass of Message. Defines π-calculus names.*

Ontology class 27 (PiCalculusActionPrefix). *PiCalculusActionPrefix is an abstract subclass of Action.*

Ontology class 28 (PiCalculusChannelReceiver). *PiCalculusChannelReceiver is a concrete subclass of PiCalculusActionPrefix and InputAction classes. It represents the π-calculus input channels. It has only one property.*

isPolyadic. *Boolean value. Can the channels receive messages consisting of more than one name?*

Ontology class 29 (PiCalculusChannelSender). *PiCalculusChannelSender is the concrete subclass of PiCalculusActionPrefix and OutputAction classes. It represents the π-calculus output channels. It has only one property.*

isPolyadic. *Boolean value. Can the channels send messages consisting of more than one name?*

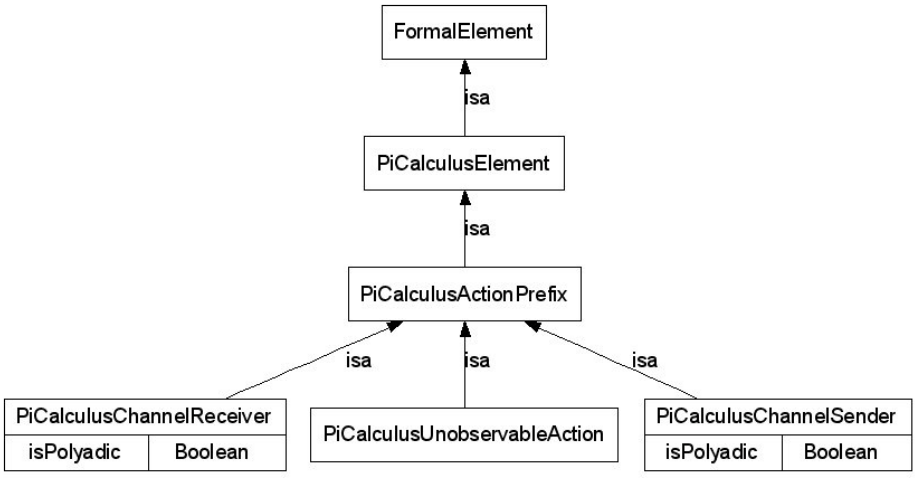


Fig. 8. π-calculus action prefixes

Ontology class 30 (PiCalculusUnobservableAction). *Concrete subclass of UnobservableAction. The τ action prefix.*

Ontology class 31 (PiCalculusComposition). *Concrete subclass of CompostionOperator. The π-calculus composition ($|$) operator.*

Ontology class 32 (PiCalculusChoice). *Concrete subclass of ChoiceOperator. The π-calculus choice ($+$) operator.*

Ontology class 33 (PiCalculusRestriction). *Concrete subclass of RestrictionOperator. The π-calculus restriction (new) operator.*

Ontology class 34 (PiCalculusReplication). *Concrete subclass of ReplicationOperator. The π-calculus replication ($!$) operator.*

2.3 Mappings of Agents onto Formalisms

At last, it is necessary to connect the first subontology to the second. We provide this connection using some rules which assert that the support for some structures in one subontology exists if, and only if, the support for some other structures from the second subontology also exists.

These rules, together with the profile[3] for a verification tool, make it possible to find out if the verification tool matches a search criteria. The search criteria is given using the mobile agent ontology, but the verification tool had been originally programmed and documented having the particular formalism (e.g., the π-calculus) in mind.

[3] The profile for a verification tool is the set of elements from both ontologies that have been registered to be supported.

Mapping Rules. We shall now state the rules, in a formalism-centric fashion.

Mapping rule 1 (PiCalculus). *PiCalculus theory is supported if, and only if, the Environment's MobilityParadigm is a LinkMobility.*

Mapping rule 2 (PiCalculusName).
PiCalculusName class is supported if, and only if, Message, MessageSensor or MessageDispatcher classes are supported.

Mapping rule 3 (PiCalculusChannelReceiver). *PiCalculusChannelReceiver is supported if, and only if, MessageSensor class is supported.*

Mapping rule 4 (PiCalculusChannelSender). *PiCalculusChannelSender is supported if, and only if, MessageDispatcher class is supported.*

Mapping rule 5 (PiCalculusUnobservableAction). *PiCalculusUnobservableAction is supported if, and only if, MessageDispatcher, MessageSensor and PiCalculusComposition classes are supported.*

Mapping rule 6 (PiCalculusChoice). *PiCalculusChoice is supported if, and only if, the supported Agents have the property Deterministic set to false.*

Mapping rule 7 (PiCalculusRestriction). *PiCalculusRestriction is supported if, and only if, the supported Environment is not fully observable.*

Mapping rule 8 (PiCalculusComposition). *PiCalculusComposition is supported if, and only if, the supported Environment has its properties set according to the following rules.*

isDeterministic. *Must be set to true.*
isMultiagent. *Must be set to true.*

Mapping rule 9 (PiCalculusReplication). *PiCalculusReplication is supported if, and only if, the supported Agent has its Cloneable property set to true.*

3 Usage Example

Let's now consider an example of how the ontology may be used. For simplicity, let's assume we have only two verification tools for the π-calculus, M_1 and M_2, which differ in notations employed as well as in the supported fragments of π-calculus.

We would like to integrate M_1 and M_2 in a system so that we could:

- query the system to find out if either M_1 or M_2 give support to a certain subset of π-calculus that we are interested in;
- write such a query using a language which does not depend on the verification tools' language syntax. For instance, M_1 might denote the composition operator as '—' and M_2 as 'comp', but we wish the query to be independent of these syntatic particularities.

To achieve these goals, the system could use our ontology. In order to integrate verification tools, the following steps could be carried out for each tool:

1. find out which π-calculus elements the verification tool supports. This could be done, for example, surveying the verification tool's documentation or source code. In the example, let's say that M_1 supports all the basic[4] π-calculus and that M_2 supports the same elements, except for the choice operator.
2. register the supported elements using the formalisms subontology. In the example, all we need to do is to tell the system that M_1 supports the PiCalculus formal theory with PiCalculusName, PiCalculusChannelReceiver, PiCalculusChannelSender, PiCalculusUnobservableAction, PiCalculusChoice, PiCalculusRestriction, PiCalculusComposition and PiCalculusReplication and tell that M_2 supports all of those too, except for PiCalculusChoice.

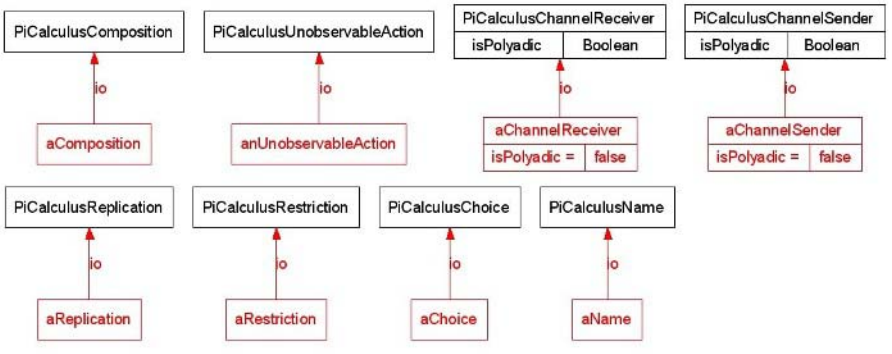


Fig. 9. Instances of all π-calculus elements we need

3. use the mapping rules to discover which elements from the mobile agent ontology are supported by the verification tool. In the example, we would find out that the only difference between M_1 and M_2 is that M_1 will support choice of agents, while M_2 won't.
4. make these elements available to a search engine. Notice that now the verification tool's particularities do not trouble the search, since they have been mapped into the ontology.

Now, suppose a user wants a verification tool that supports some elements from the π-calculus, in particular the choice operator. To locate the desired verification tools, the following steps could be performed:

1. using the ontology, the user tells the system which elements from the π-calculus theory are required.

[4] The subset of π-calculus which contains action prefixes, choice, composition, replication and restriction

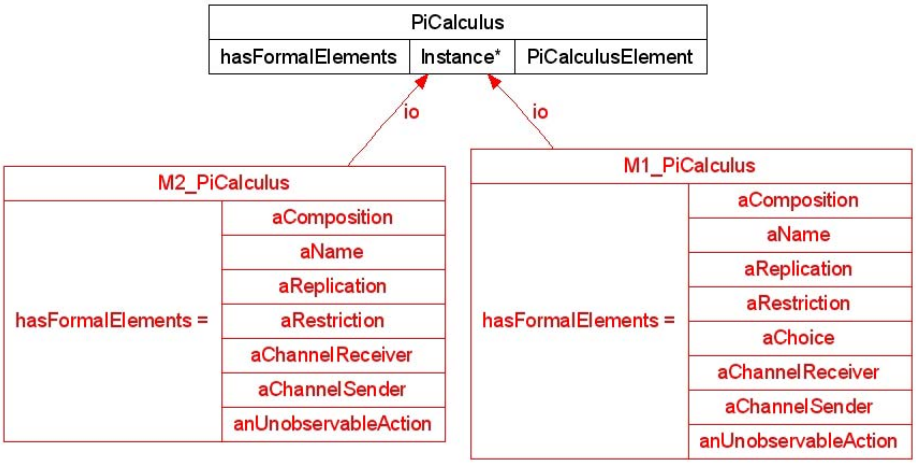


Fig. 10. Instances of the π-calculus theory that each verification tool support

2. using the mapping rules, the system transforms these elements into mobile agent's elements.
3. the system checks in its data base if there are any registered verification tools that gives support to the mobile agent elements. We have seen that only M_1 supports PiCalculusChoice. Thus, only M_1 will be returned to the user as a search result.

The point here is that the search is done using a common language, which does not depend on the underlying formal theory. Although so far we have been working only with the π-calculus, we believe that other formalisms, such as Ambient Calculus [7] or Join-Calculus [6], could be used as well.

The example π-calculus tools used in this section are fictitious and were proposed to illustrate the problem of having tools for a single formalism that are concerned with different fragments of it. In practice, we also have verification tools for π-calculus which actually support different subsets of the theory. The verification tool VTubaina [8], for example, does not support the choice operator, while another tool, HAL [9, 10, 11], does support choice but not replication. In fact, these tools can share services although they contemplate different fragments of π-calculus and their input languages differ.

4 Discussion

This paper presented two main ontologies: one for mobile agents and one on formalisms of mobile agents. Besides that, the relation between elements of both ontologies has been provided in order to show how agents can be modeled from abstract concepts to appropriate mobile agent formalisms. π-calculus has been

used as instance of the formalism ontology, and an example involving two proposed tools for π-calculus was presented to give insights on the usage of ontologies as a framework to register tools and use them in a collaborative way.

The ontologies defined here have a double purpose. From a computational stand point, it provides a common language for information exchange. This allows tools using different notation to integrate, albeit in a limited fashion. On the other hand, from the user's point of view, the ontology defines an *understandable* language, from which higher level concepts of the mobile agent domain may be easily handled.

We have given the cell phone as an example of mobile agent that can be modeled using our ontologies. Other examples can be easily found among modern wireless devices (e.g., wireless networks, where computers may connect as they move into a certain region), which share many of the cell phone's characteristics. Furthermore, mobile agency is also present in software systems. An example of software mobility is the Aglets project [16].

Regarding those features, verification tools can be registered as instances of the formalism ontology and we can further provide an environment for sharing services from various tools. For that, a software system must actually be implemented and make use of our ontologies to register verification tools in order to build a cooperative environment. We believe that Protégé-2000 can be used as a framework for such implementation, since it is designed as an *extensible*[5] tool for ontologies.

In this particular work, we have instanced the formalism ontology for π-calculus. This formalism ontology, however, is not limited to represent π-calculus. Instances of this ontology for other formalisms, such as Join-Calculus and Ambient Calculus, can be produced in the same way. With such new instances of formalisms, we could register tools based on these other calculus and, better than this, provide an environment that comprises services from different formalisms. The ontological relation among all instanced formalisms can be established via the main formalism ontology and, for certain classes of problems, tools services from different formalisms can be shared.

The ontology developed in the present work refers only to the representation of agents. To provide an environment for sharing services from verification tools for mobile agent systems, we also need, as stated in the introduction, an ontology to capture the *capabilities* of the verification tools. This study is in progress and is first devoted to capabilities of model-checkers and equivalence verification tools.

We don't have the expectation of solving the whole problem of services-collaboration among verification tools for mobile agent systems with such ontologies. We already known that ontologies are not enough for knowledge sharing [18] because a semantic mapping is necessary for certain pairs of formalisms, and this problem remains in services-sharing among verification tools for mobile agents. However, many of the formalisms for mobile agents are comparable and a great part of them can be mapped without a semantic conversion. The present work is a step-forward to provide services-sharing among tools for such classes of

[5] Through the use of plug-ins.

problems. For a complete services-sharing, a study on pairwise verification tools must be conducted and there is no guarantee that a semantic mapping can be found for each pair of formalisms.

Finally, we note that the ontologies described in this paper have not yet been implemented in a working software system.

Acknowledgments

This project has been granted by CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brazil*) and FAPESP (*Fundação de Amparo à Pesquisa do Estado de São Paulo -Brazil*), Project 03/00312-0.

References

1. Stuart Russel and Peter Norvig. *Artificial Intelligence:* A Modern Approach. Prentice Hall, 2003.
2. Robin Milner. *Communicating and Mobile Systems:* the π-Calculus. Cambridge University Press, 1999.
3. Nicola Guarino and Pierdaniele Giaretta. *Ontologies and Knowledge Bases:* Towards a Terminological Clarification. In *Towards Very Large Knowledge Bases:* Knowledge Building and Knowledge Sharing, N. Mars (ed.), pp 25-32. IOS Press, Amsterdam, 1995.
4. Thomas R. Gruber. A translation approach to portable ontologies. In *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.
5. Mark D'Inverno and Michael Luck. Understanding Agent Systems. 2nd ed. Springer, 2004.
6. *The Join-Calculus language*. Resources for the Join-calculus. Available at: <http://join.inria.fr/>. Accessed in: May 23rd, 2005.
7. Luca Cardelli. *Mobile Computational Ambients*. Introduction and other resources to the Ambient Calculus. Available at: <http://www.luca.demon.co.uk/Ambit/Ambit.html>. Accessed in: May 23rd, 2005.
8. Marcelo M. Amorim. VTUBAINA Tool. A Verification Tool for Up-to Bisimulation and Automata Integration Automatization. Available at: <http://www.lcpd.ime.usp.br/~mamorim/vtubaina/>. Accessed in: May, 23rd, 2005.
9. Ferrari, G., Gnesi, S., Montanari, U., Pistore, M. and Ristori, G., Verifying Mobile Processes in the HAL Environment, in: Alan J. Hu and Moshe Y. Vardi, Eds., *CAV'98*, *Springer LNCS* 1427, pp.511-515.
10. Montanari, U. and Pistore, M., An Introduction to History Dependent Automata, in: Andrew Gordon, Andrew Pitts and Carolyn Talcott, Eds, *Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II)*, ENTCS, Vol. 10, 1998
11. History Dependant Automata Laboratory (HAL). Available at: <http://rep1.iei.pi.cnr.it/projects/JACK/HAL/hal.html>. Accessed in: May, 23rd, 2005.

12. *Web Ontology Language (OWL)*. Specifications, articles, tools and other resources for OWL. Available at: <http://www.w3.org/2004/OWL/>. Accessed in: August 19th, 2004.
13. *Description logics*. Description logics resources. Available at: <http://dl.kr.org/>. Accessed in: February 21st, 2005.
14. The Protégé Ontology Editor and Knowledge Acquisition System. Available at: <http://protege.stanford.edu/>. Accessed in: August 19th, 2004.
15. OntoViz. Information and downloads regarding the OntoViz Protégé plugin. Available at: <http://protege.stanford.edu/plugins/ontoviz/ontoviz.html>. Accessed in: February 21st, 2005.
16. Aglets. Information and downloads regarding the Aglets framework, a system for mobile software agents. Available at: <http://aglets.sourceforge.net>. Accessed in: August 15th, 2005.
17. L. Cardelli and A. D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *LNCS*, Springer-Verlag, 1998.
18. Flávio S. C. da Silva, Ana C. V. de Melo, Jaume Agusti, Wamberto Vasconcelos, David Robertson, Marcelo Finger, and Vigínia Brilhante. On the insufficiency of ontologies: Problems on knowledge sharing and alternative solutions. *Knowledge-Based Systems*, 15(3):147–167, 2002.
19. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages*. ACM Press, 1996.
20. C. Fournet, G. Gonthier, JJ Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *CONCUR'96*, volume 1119 of *LNCS*. Springer–Verlag, 1996.
21. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, October 2003.

Evaluating Ontology Criteria for Requirements in a Geographic Travel Domain

Jonathan Yu, James A. Thom, and Audrey Tam

School of Computer Science and Information Technology,
RMIT University, 124 La Trobe Street, Melbourne, Victoria 3000, Australia
{jyu, jat, amt}@cs.rmit.edu.au

Abstract. An ontology is a model of a domain of knowledge. The knowledge that is captured in an ontology can be used for providing interoperability, sharing of information and reduction in semantic ambiguity in application areas such as Knowledge Management, the Semantic Web and E-Commerce. Given that each of these application areas may have differing requirements, how do we assess whether an ontology adequately addresses the requirements of a given application? In this paper, we consider how existing criteria and measures can be used to evaluate different aspects of the suitability of ontologies for application areas in the domain of travel and geography. Specifically, we consider some existing measures of coverage, as well as propose new ones, and determine whether they address the application requirement of appropriate granularity.

1 Introduction

Many organisations today possess increasingly large information repositories and they share common problems in managing their information and reusing it for other applications. Lonely Planet (LP)[1] is an example of such an organisation with regards to its information on travel. Ontologies can help to manage such information as they provide a machine processable and sharable model of information for various applications. It is especially useful if an ontology is already available and suitable, as building one is time consuming.

Researchers in the past have proposed design criteria for building ontologies. However, because of the increasing availability of ontologies, there is a need to evaluate existing ontologies using evaluation criteria and methodologies. These evaluation criteria can, in turn, be useful in determining the suitability of an ontology for a given application.

In this paper, we examine some requirements which arise in managing information such as a large corpus of travel information. We then analyse whether the existing criteria address these requirements when assessing the suitability of a set of ontologies within the travel and geographic domain. Section 2 will introduce some requirements in this domain using LP as a case study. In Section 3, we discuss some difficulties in ontology modelling and reuse, and we

[1] http://www.lonelyplanet.com

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1517–1534, 2005.

explore some criteria and measures for evaluating ontologies. In Section 4, we analyse whether the proposed evaluation criteria satisfies the requirements in this domain. Lastly, in Section 5, we explore one criteria, that is coverage, in greater depth and explore existing measures (such as the vector space similarity measure) and alternative measures (such as the F-Measure).

2 Information Requirements of Lonely Planet

LP has published around 650 books that describe travel information in various geographic locations around the world. Each guidebook belongs to one of the following categories: Travel guides, City guides, "On a shoestring" guides, "Best of..." guides, Classic Overland Route guides, Maps (road and city), Activity books, and General reference books. Descriptions of a particular geographic location may be found in many different books. For example, descriptions about the city Paris can be found in at least five different series of guidebooks:

- *France Travel Guide* (January 2003)
- *Western Europe Travel Guide* (January 2003)
- *Paris City Guide* (October 2002)
- *Europe "On a shoestring" Guide* (February 2002)
- *Paris Condensed* (February 2002)

Also, the content describing a given location in a particular book occasionally overlaps with content from other books. For example, in each of these books, descriptions of parts of Paris and even some entries for restaurants were the same in each book about Paris. In this section, we consider some requirements that arise out of managing and reusing such a collection of travel information.

2.1 Conflicting Requirements of Print and Digital Services

Guidebooks can contain valuable information. This information can be repackaged and customised to be used in other applications. For example, content may be collected and reorganised into a book on world food encompassing information on cuisines of the world and restaurant information.

Information can also be repackaged for digital services such as applications for mobile devices and the web. For such digital services, the expectations of users intrinsically impose a demand for up-to-date information. For example, if information about accommodation, restaurants and entertainment venues were to be delivered to a traveller's mobile device, it is the expectation that information such as the address and the opening hours of the place are current. Hence, if information from the guidebooks were to drive such digital services, it is important to ensure information is up-to-date.

Some information in the guidebooks is bound to be outdated as there may be an 18-month gap between the gathering of updated information to the publication dates of subsequent updated editions of a particular guidebook. For

example, restaurant locations. Expectations of users are more lenient as users may be aware of this.

Nevertheless, there is a conflict here in the requirements of users with regards to print and digital services. Therefore, some information from printed guidebooks may not be entirely suitable to drive some digital services.

2.2 Consistency Between Updates

Out of LP's collection of guidebooks, the more popular ones, such as the Paris City guide, are revised and updated every two years. Generally, each book, as well as its revised editions, has an asynchronous production schedule. That is, each book is researched, written (or updated) and may be published on different dates.

Upon an update in one book, there is bound to be conflicts. This is due to the overlapping content of some of the books and the independent nature of updates. However, where there are overlapping content between each book, it must be consistent.

2.3 Accommodating Different Views of the Same Information

Content may also differ between guidebooks. A book may have a different view of the topic or may be written with different audiences in mind. For example, the *Paris Condensed* and the *Paris City Guide* books have different descriptions of the Chinatown region in Paris. Also, these books may also categorise sections differently. For example, the *Paris Condensed* book categorises the places of interest in a topical manner (for example, museums and galleries) whereas the *Paris City Guide* book describes the different places of interest in order of city districts and regions. Hence, there is a need to accommodate different views of the same information.

2.4 Appropriate Granularity

Geographic tools such as Global Positioning Systems (GPS) and maps allow geographic locations to be mapped with an accuracy of a few metres making it possible to model an area in great detail. However, such as level of detail may not be required in the travel domain. Users, that is travellers, may only be interested in relevant information such as descriptions of airports, reputable accommodation, places of interest, shopping areas and places to eat.

The problem is then of determining the level of granularity required. If the information is too coarse grained, relevant information may be omitted. Conversely, if it is too fine grained , some information may end up unused and effort would have been wasted in modelling that information. Thus, there is a need here in being able to capture the *appropriate* level of granularity in the travel domain.

2.5 Flexibility in Classifying Items

Maps tend to split up a geographic location to distinguish one region from another. For example, the city of Paris in France may be separated from the sur-

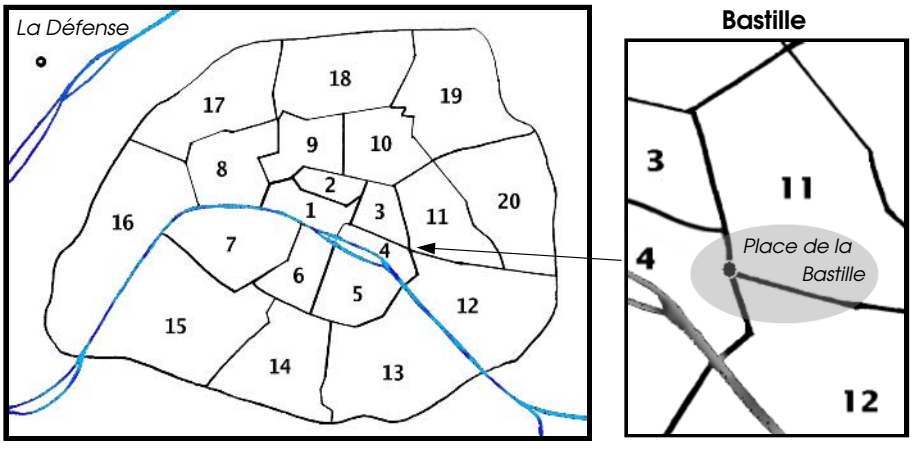


Fig. 1. a) Map of the districts in Paris; b) Close up of the Bastille Region

rounding regions according to the boundaries in the map. However, people may loosely associate places in the surrounding regions with Paris. For example, La Défense is often associated with Paris although it is not part of the city of Paris; it is in a nearby region (see Figure 1). Travellers may be interested in La Défense during their visits to Paris. Sometimes this does not correspond with the classification in maps and models of the city. As such, there is a requirement in this domain for flexibility in classifying places and accommodating these 'fuzzy boundaries'.

In summary, we have outlined five requirements describing issues to do with information management, reuse and modelling in the travel domain. Ontologies may be used to address some of these requirements and we will discuss this further in Section 4. In the next section we will briefly describe what ontologies are and discuss some of the issues related with them.

3 Ontologies

An *ontology* is an explicit model of a domain of knowledge consisting of a set of concepts, their definitions and inter-relationships [15]. It supports information reuse because the information captured can be reused for other applications.

Recently, an ontology specification language called the Web Ontology Language (OWL) has been proposed by the W3C as part of the Semantic Web initiative [12]. OWL is an ontology language for the web, which allows web documents to be formally described for machine processability and reasoning tasks [12]. It is built on existing web standards (XML and RDF) and is founded on Description Logics. Figure 2 shows how a simple and course grained conceptual model of Paris in France is specified in OWL. In the example in Figure 2 three classes are defined – Place, Country and City. Constraints are put on these classes such that City and Country are both subclasses of Place. Additionally,

```
<owl:Class rdf:ID="Place"/>

<owl:Class rdf:ID="Country">
 <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>

<owl:Class rdf:ID="City">
 <rdfs:subClassOf rdf:resource="#Place"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasCapitalCity">
 <rdf:type
    rdf:resource="&owl;FunctionalProperty"/>
 <rdfs:domain rdf:resource="#Country"/>
 <rdfs:range rdf:resource="#City"/>
</owl:ObjectProperty>
```

```
<Country rdf:ID="France">
 <hasCapitalCity rdf:ID="#Paris"/>
</Country>

<City rdf:ID="Paris"/>
```

Fig. 2. Model of Paris and France in OWL: a) Schema; and b) Instances

a functional property is specified for the class Country of a capital city called 'hasCapitalCity'. We can then specify instances of these classes. In our example, we define the country of France and the city of Paris and specify that France has Paris as its capital city.

3.1 Difficulties with Ontologies

Much work has been performed on ontology engineering methodologies and researchers have proposed design criteria for building ontologies [6, 7, 8]. Some criteria have also been proposed for evaluating ontologies for their design to be improved. We will consider these criteria in greater detail later in this section. However, we first consider some difficulties faced and draw examples from within the geographic domain.

The process of building an ontology manually is time consuming and sometimes difficult to specify. It involves acquiring the necessary knowledge and then specifying it using a language such as OWL. For example, building and implementing a travel ontology at LP involved modelling world information such as geographic locations, places of interest, cultural information and cuisines of the world. They found that some information that needed to be modelled often did not have a clearly defined boundary. For example, information about some regions that were required to be modelled were loosely associated with neighbouring places. This made it difficult to classify information in a strict hierarchy. This reflects the requirement for flexibility in classifying items as discussed earlier.

Existing ontologies may already be available. In such cases they could be reused, rather than building an ontology by hand. In the geographic domain, there are existing off-the-shelf ontologies available such as ESRI[2], and Getty[3] as well as GIS street level ontologies used by local authorities. However, it was found that these ontologies either had too much or too little detail.

[2] http://www.esri.com

[3] http://www.getty.edu/research/conducting_research/vocabularies/tgn

The difficulty in using ontologies which are too detailed is that they model low level details. For example, a particular ontology models street level details such as road names and the width of the footpath, which are items that local council authorities are interested in. On the other hand, travel applications may not require such details. Rather, what is required is a higher level of abstraction to model shopping districts with relevant details such as locations of boutique shops.

One solution is to model what is required over the low level details. However, there are instances where this is not possible. For example, a particular GIS street level ontology, has 42 different classifications of roads. LP already have their own model where there are 9 classifications of roads and some can not be mapped to the GIS street level ontology classifications. Hence there is still a difficulty in conflicting and missing definitions in different models.

In contrast, ontologies that lack the relevant details require a lot of annotation to extend and add in the missing parts. This reflects a requirement for appropriate levels of granularity, as discussed earlier.

The off-the-shelf ontologies that were considered by LP did not suit the application requirements. However how do we decide which ontology is suitable for the application at hand? We next discuss some evaluation criteria and analyse whether they can be used to evaluate the suitability of an ontology for a particular application.

3.2 Ontology Evaluation Criteria

With ontologies becoming increasingly available from various resources such as the Swoogle search engine[4] and the SchemaWeb ontology directory site[5], it is getting easier to pick an ontology *off-the-shelf* and customise it for an application. However, it then becomes a challenge to find a suitable ontology among a set of ontologies that describe the domain for a specific application, as we do not want to use an ontology for an application to find later that it is not suitable. Therefore, a mechanism is needed for evaluating ontologies for use in an application.

Various criteria have been proposed for the evaluation of ontologies. Upon analysis, some of the criteria proposed by the different researchers address similar aspects when evaluating ontologies and overlap. We collate and describe them below as nine distinct criteria.

1. Clarity
   A clear ontology should "effectively communicate the intended meaning of defined terms" and where possible the definition should be stated formally [6]. Definitions in an ontology should be clearly specified such that there is no doubt or ambiguity. For example, consider an ontology with a definition of 'Paris'. The definition should clearly indicate whether it is regarding a person's name or a place. If it is a place, which country is it

[4] http://swoogle.umbc.edu
[5] http://www.schemaweb.info

referring to – a town in North-East Texas, USA named Paris or the capital city in France?

2. Consistency
   Consistency, also called coherence [6], describes the logical consistency of an ontology [4]. For a given ontology's set of definitions and axioms (explicit or inferred), there should be no contradictions. Guarino and Welty went further in seeking to validate the consistency of the subsumption of entities and properties with their proposed OntoClean methodology [8]. This methodology seeks to correct entities that are modelled as subclasses when they should be consistently modelled as properties, as a subclass of another entity or even a separate entity on its own. The principles of essence, rigidity, identity and unity are used to determine this [8]. For example, the *Paris Condensed* guidebook describes riding bicycles in the city of Paris as dangerous, whereas Paris is described as bicycle-friendly in another guidebook – the *Paris City Guide*. Here there is a contradiction within the set of descriptions in information contained in the collection of books about riding bicycles in Paris.
3. Conciseness
   According to Gómez-Pérez an ontology is concise if:
   (a) It does not store any unnecessary or useless definition
   (b) Explicit redundancies do not exist between definitions
   (c) Redundancies cannot be inferred using axioms attached to other definitions
   (d) The definition itself is not redundant

   That is, an ontology is concise if redundancies do not exist and cannot be inferred from its definitions and axioms [4]. For example, it would not be concise if we modelled definitions about Bastille each time for the three districts that it may be part of. However to be concise, the definition for Bastille would describe it as being part of the 4th, 11th and 12th districts of Paris.
4. Expandability
   Expandability [4], also called extendibility [6], relates to the ability of the ontology to be extended further to describe specific application domains in a way that does not change the current definitions within the ontology. For example, consider an ontology about travel in Paris that only models information on places to eat. This ontology would be expandable if it were possible to include information about accommodation and places to stay without introducing inconsistent or redundant definitions.
5. Correctness
   Correctness refers to whether the representational choice made when entities and their properties are modelled correlate with entities in the world being modelled. Gómez-Pérez includes the metaphysical aspect as part of her description of consistency [4]. We will consider this in our definition of correctness.

   Correctness depends on the frame of reference that the ontology is based on. One approach to measure correctness is to verify correct modelling of

concepts from real world, such as interviews with domain experts and literature regarding the domain. For example, in the Methontology [11] design process, the ontology developers had to verify the correctness corresponding to a frame of reference from interviews with domain experts and other information sources. However this approach is time consuming and involves manual inspection.

Extending our previous example, a correct ontology would model La Défense as being part of a separate region of France rather than being part of the city of Paris.

6. Completeness
   Whether an ontology or its individual definitions are complete cannot be proven, however we can deduce an incomplete ontology by detecting individual definitions that may be incomplete or that at least one definition is missing from an ontology [4]. More specifically, we refer to a frame of reference to determining an ontology's incompleteness by using an ontology's set of competency questions [7] or with its reference to the real world itself. For example, if we modelled Bastille (an area located at the intersection of the 4th, 11th and 12th district) and we did not include it as being in one of the districts, say the 4th district, the definition of Bastille would not be complete. This is because it is considered to be part of the 4th district in addition to the 11th and 12th district in Paris (see Figure 1 for the map).
7. Coverage
   Hovy [9] refers to coverage in two distinct ways: *coverage of terms over the domain from concepts identified in the domain*; and *coverage or completeness of instances*. We will consider coverage as the coverage of terms over a given information domain rather than the latter.

   Brewster et al. [2] take a slightly different approach. They refer to coverage as the *congruence* or *fit* of an ontology with the domain represented by the corpus of information. The difference here is that the text corpus is taken to represent the domain itself.

   An example of coverage is if we had the situation as in Figure 3. In ontology 1, there are three terms that overlap with the defined set of concepts whereas ontology 2 has only one overlapping term. Hence in this case ontology 1 has better coverage of the domain than ontology 2.
8. Minimal Ontological Commitment
   Ontological commitment refers to the an ontology being able to be agreed upon by users or ontology adopters. Studer et al. [14] refer to adopters of an ontology as having agreed to *commit* to that ontology which describes some domain knowledge. Gruber defines ontological commitment as *"an agreement to use a vocabulary (i.e. ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology"* [5]. Minimal ontological commitment refers to minimising the ontological commitment of an ontology to allow more freedom in an ontology's usage. It is about not over-defining terms that may impede some potential users of the ontology. One way to minimise ontological commitment is to make "as few claims as possible about the world being modelled" [6].

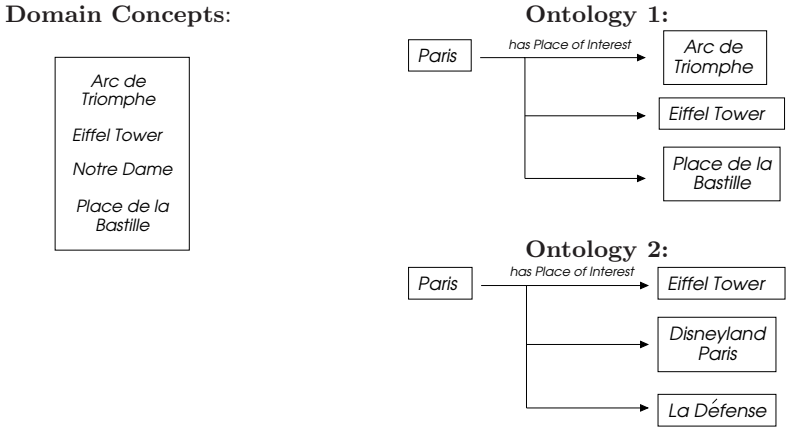


Fig. 3. Coverage Example Text

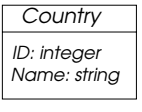


Fig. 4. Encoding Bias Example

Another way is by breaking an ontology into separate ontologies as Gruber suggests [6].

9. Minimal Encoding Bias
   An encoding bias occurs when "representational choices are made purely for the convenience of notation or implementation" [6]. This should be minimised to prevent an ontology to be specific only to a particular ontology notation, language specification or implementation. An example of encoding bias is found in Figure 4 where we have modelled country as having an integer value that is associated as an identifying number for our system. As ontologies describe aspects of the real world, countries, in general, does not have ID numbers, hence there is an encoding bias here.

Having described ontologies and the proposed ontology evaluation criteria, we will match them up to the information requirements in the next section, and then look at one of these criteria, that is coverage, in more depth in Section 6.

4 Satisfying Requirements in the Domain with Ontology Evaluation Criteria

Having described the proposed ontology evaluation criteria, we now consider whether they help in evaluating suitable ontologies meeting the requirements of

applications in the geographic domain. We will describe them by listing which requirements were addressed and which were not fully met.

4.1 Requirements Addressed

Ensure Information is up to Date. The nature of on-demand digital services requires information to be current. For example, if we needed the location of a particular restaurant, it isn't very useful to return an address of a restaurant that moved six months ago. The *correctness* criteria addresses this because by having an ontology that is correct means that its definitions agree with the world being modelled. More specifically, instances in an ontology may be required to be updated more frequently than classes, as they tend to be more volatile.

Appropriate Granularity. Often users are overloaded with information. In this domain, we may not require our geographic information to come down to the fine details (that is, to GPS co-ordinates). The only criterion that addresses this is *coverage*. The coverage that an ontology has over a given information domain may determine whether an ontology has a suitable level of granularity. We will explore this measure in detail in the next section.

4.2 Requirements not Fully Met

Consistency Between Books Upon Updates. Because of the range of different series of books that cover different some information that overlap, updating one book will mean that information in other books may become redundant. The *consistency* criterion refers to internal consistency of the ontology but does not capture this usage of consistency. However, an ontology can be utilised to aid maintaining consistency in overlapping information in books by providing access to these overlapping aspects of the book content.

Accommodating for Different Views. The requirement for accommodating for different views of the same information arises from having different books describing information from contrasting perspectives. This requirement is addressed by utilising an ontologies to represent each view. An approach to address this requirement could be to have a different ontology for each view and an ontology that map overlapping descriptions. For such a shared ontology, the criteria of *minimal ontological commitment* applies.

Flexibility in Classifying Items. The level of ontological commitment can bear some effects on the flexibility of an ontology in classifying items. The more that is described in an ontology, the more likely it is that there will be a higher ontological commitment, which may lead to less flexibility. Hence *minimal ontological commitment* affects this requirement. However, there is an issue in knowing what *minimal ontological commitment* looks like. Certainly it is possible to describe the weakest theory, but there is a danger here in that the ontology is stripped down so much that the information contained in an ontology becomes ineffectual with regards to our application.

Despite having a minimal ontological commitment and having definitions clearly specified, a given definition may not meet our requirement of flexibility. There are cases it can be difficult to define items due to the fuzzy nature of the definitions themselves. This is especially the case when describing places that possess *fuzzy boundaries*. Consider the extent that the region of Paris extends to. There are strict boundaries drawn in maps that define the tourism region. However, some places just outside of Paris are associated as being part of the city as well. For example, the area of La Défense (see Figure 1 for the map of Paris). Hence, the evaluation criteria, specifically *clarity*, does not help meet the requirement for flexibility in classifying items.

Also, being *clear* and *correct* about a domain in a strict and rigid manner does not necessarily address the requirement of flexibility in classifying items in this domain. For example, consider these definitions: *a place can be located in only one district*; and *Bastille is a place in the 4th district*. Although, the definitions may be clear and accurate to a certain extend (as Bastille is a place in the 4th district), it does not model the world correctly in its entirety as Bastille is also part of other districts in Paris, namely the 11th and 12th district.

However, an ontology would need to be clear and correct for applications to be able to process definitions unambiguously, as mentioned in the previous section. Hence there needs to be some reconciliation between the requirement of the domain to be modelled and for machines to be able to consume ontologies.

This issue of fuzzy boundaries and what defines part-whole relations has previously been explored in other areas. For example, Varzi [16, 17] explores the area of mereotopology and vagueness in the discipline of philosophy; and Klause & Clark [10] use fuzzy logic to try to deal with fuzzy sets in computer science.

In ontology engineering, Guarino and Welty [8] touch on this aspect of part-whole relations. In their OntoClean methodology, they aim to assess whether subsumption (that is is-a relations) are correctly modelled in an ontology by using some meta-properties. One aspect of this methodology is the assessment of an incorrectly subsumed relation where it should really be modelled as a part-whole relation.

We have seen in this section that while there are still some requirements that have not been fully addressed yet there are other requirements that have been addressed by the various proposed criteria. Table 1 shows a summary of the criteria that satisfy the requirements outlined in our application domain.

Some of these criteria can be successfully measured using ontology tools such as reasoners. Reasoners, such as FaCT and RACER, provide the means to check for errors in ontologies, such as redundant terms, inconsistencies between definitions and missing definitions. Additionally, Dong et al. have used existing software engineering tools and techniques to check for errors in ontologies in the military domain [3].

Other criteria can be more challenging to evaluate as they may be hard to quantify. While some criteria are available for assessing the suitability of an ontology, they would require manually inspecting the ontology. For correctness, it would require manual inspection to verify that the definitions are correct with

Table 1. Criteria which satisfy requirements in this domain

Requirement	Criteria								
	1	2	3	4	5	6	7	8	9
Consistency between books upon updates									
Ensure information is up-to-date					●				
Accommodating for different views								○	
Appropriate granularity							●		
Flexibility in classifying items								○	

● - *Addresses* ○ - *Partially Addresses*

1-Clarity
2-Consistency
3-Conciseness
4-Expandability
5-Correctness
6-Completeness
7-Coverage
8-Minimal O.C.
9-Minimal Enc. Bias

reference to the real world. This may not be a feasible task for a repository of ontologies.

5 Coverage Measure

A more quantitative approach is needed to evaluate ontologies based on the proposed evaluation criteria. Brewster et al. propose a more objective approach called data-driven ontology evaluation [2]. In this approach, an ontology is evaluated for its suitability based on its *congruence* or coverage with the domain represented by the corpus of information using various measures. The advantage of this approach is that it measures an ontology with respect to a corpus of information in an automated way. It does not rely on manual inspection. In this section, we will explore existing coverage measures in detail, as well as introduce alternative measures, to identify suitable ontologies and analyse whether coverage addresses the issue of an ontology having the appropriate granularity.

We now consider some of the measures of coverage proposed by Brewster et al. [2] and apply them in the travel and geographic domain. Given an application and a text corpus that represents the knowledge in that domain, Brewster et al. [2] aim to identify the most suitable ontology from a given set of ontologies. They propose four approaches in comparing an ontology with a text corpus: *counting the number of overlapping terms*, *vector space similarity measure*, *structural fit by clustering and mapping terms*, and using *conditional probability* to evaluate the 'best fit' of an ontology. We will only consider the first two measures as our text corpora are not annotated.

We also consider whether measures used in information retrieval – *precision*, *recall* and the *F-Measure* – can be adapted to measure coverage.

5.1 Dataset and Ontologies Used

To ascertain whether coverage helps to determine suitable granularity in our domain, the dataset used should have varying levels of granularity regarding travel. For this experiment, we considered a sample of travel text on Paris taken from three books with differing levels of detail:

- The Paris sections in *Western Europe Travel Guide* (WE) - least detail
- The Paris sections in *France Travel Guide* - more detail
- *Paris City Guide* - most detail

We used LP's ontology as our base ontology. The original purpose of LP's ontology was to maintain a consistent vocabulary for use within the organisation such as the names of places and activities used in their guidebooks. In particular it contains names of actual geographic places in the world such as countries, provinces and cities. This part of the ontology is coarse-grained: it models every continent and country in the world, but does not model fine details in many of the countries. For example, it models most of the provinces of France but does not have city- or town-level details

For measuring coverage, we used this ontology and varied its granularity by adding increasing levels of detail in France and Paris. We begin with an ontology that included instances of the regions in France (such as Brittany and Corsica) and the major cities or towns (such as Paris, Nantes and Marseille). This becomes O_{France}. We then increase the granularity in Paris by adding the city's major metro and train stations (such as Bastille and Place d'Italie), as more detailed instances of Paris locations and refer to it as $O_{ParisMajor}$. We increase the granularity again by adding all of the Paris metro and train stations (such as Rue des Boulets and Porte de Versailles), as more detailed instances of places of Paris. This becomes $O_{ParisDetailed}$, where $O_{France} \subset O_{ParisMajor} \subset O_{ParisDetailed}$.

5.2 Method

For each measure, we extract the unique set of terms from our dataset after first applying stopping and stemming using the Porter stemmer [13]. The same techniques are applied to the text labels of concepts, instances and properties of the ontologies.

Number of Overlapping Terms. This involves finding matching terms between the terms in an ontology and the text corpus. Brewster et al. [2] only propose to measure the overlap between the extracted terms in the ontology and the text corpus it is being compared to.

Vector Space Similarity (VSS). Brewster et al. propose a vector space measure to compare various ontologies to a given corpus [2]. However, they do not describe the implementation details. We implemented the vector space model using the cosine angle to obtain a similarity score [1]. We omit the inverse document frequency (idf) factor as we are comparing extracted terms from the given ontology to a single text corpus at a time. Hence the ontologies can be seen as the queries. We also omit query term weighting, that is the weighting of terms in the ontology, and just give a binary weight, that is 1 if the term is present or 0 if not.

The measure of counting of overlapping terms presented above does not take into consideration the size of the corpus or ontology. Brewster et al. [2] compare a diverse set of ontologies with a single text corpus. However, we compare a set of

three ontologies of increasing granularity to three different text corpora. To allow a relative comparison among the different corpora, we need to consider the sizes of both the corpus and ontology. To help address this, we now present additional measures, adapted from information retrieval, that have not been used before in ontology evaluation. Specifically, these are precision, recall, and the F-Measure.

Precision. measures the percentage of ontology terms that overlap with the corpus. This is given by:

$$precision = \frac{\mid O \cap C \mid}{\mid O \mid}$$

Recall. measures the percentage of the text corpus terms that overlap with the ontology. This is given by:

$$recall = \frac{\mid O \cap C \mid}{\mid C \mid}$$

F-Measure. is a combined metric of the precision and recall and gives a harmonic mean that combines both the values of precision and recall [1]. This is given by:

$$F(j) = \frac{2}{\frac{1}{recall_j} + \frac{1}{precision_j}}$$

5.3 Results and Discussion

Table 2 shows a count of the set of terms extracted from both the ontology and the text corpus, the set of overlapping terms and the associated measures. From this, we observed that the size of the three text corpora differ greatly – the Paris chapter in the *Western Europe Travel Guide* being the smallest and the *Paris City Guide* having the most terms. When we add the names of the major metro and train stations to the base ontology (O_{France}) to form $O_{ParisMajor}$, there is a 15% increase in the number of terms (113 terms) compared with O_{France}. When we include the names of all of the metro and train stations in the ontology in $O_{ParisDetailed}$, there is an increase of nearly 50% (416 terms) in the number of terms compared with $O_{ParisMajor}$. However, we observe that this does not necessarily produce an equivalent increase in the number of overlapping terms extracted from the ontology and the text corpus. Also, in Table 2, the results highlighted in bold show which is the most suitable ontology for a given corpus using each coverage measure.

Figure 5a shows the trends for the overlapping measure. When more detail is added to the ontology, the number of overlapping terms increases. However, there is no penalty in this measure for including unnecessary terms in the ontology.

Figure 5b shows that as our ontologies increase in granularity, the vector space similarity score generally increases in each text corpus of our dataset. However,

Table 2. Results for ontology and corpus size, number of overlapping terms, vector space similarity, precision, recall and the F-Measure

Corpus	$\|C\|$	Ontology	$\|O\|$	$\|O \cap C\|$	VSS	Precision	Recall	F-Measure
WE-Paris	2792	France	739	265	0.1083	0.3586	0.0949	0.1501
		Paris Major	852	360	0.1367	**0.4225**	0.1289	0.1976
		Paris Detailed	1268	**533**	**0.1595**	0.4204	**0.1909**	**0.2626**
France-Paris	7142	France	739	448	0.1220	0.6062	0.0627	0.1137
		Paris Major	852	559	0.1420	**0.6561**	0.0783	0.1399
		Paris Detailed	1268	**831**	**0.1571**	0.6554	**0.1164**	**0.1976**
Paris City	11953	France	739	535	0.1778	0.7240	0.0448	0.0843
		Paris Major	852	648	**0.1974**	**0.7606**	0.0542	0.1012
		Paris Detailed	1268	**957**	0.1958	0.7547	**0.0801**	**0.1448**

for the *Paris City Guide* corpus, adding fine granularity details of places in the city of Paris to the ontology (that is $O_{ParisDetailed}$), had a slightly negative effect on its similarity score. This suggests that the level of detail was too much for the *Paris City Guide* corpus, but surprisingly not for the other less detailed corpora.

Figures 5c and 5d show trends for recall and precision respectively. We observe that recall generally improves when more detail is added to the ontology, and that precision improves when adding the names of the major Paris metro and train stations. However, when all of the names of the stations were added, that is in $O_{ParisDetailed}$, precision starts to decrease slightly. This may indicate that the level of detail was too much for all three text corpora.

Figure 5e show trends for the F-Measure, which combines aspects of both recall and precision. We observed that the F-Measure score increases as the granularity of the ontology increases, and seems to be dominated by the recall component.

An appropriate ontology should contain sufficient but not unnecessary detail to model all instances down to the appropriate level of granularity for the application. A good coverage measure would capture this. The behaviour of the vector space similarity measure does not correctly capture the notion of granularity. None of the other measures are able to show that a particular ontology is more suited for corpora of different levels of granularity.

When determining an appropriate granularity for a given ontology, terms that appear infrequently may be just as important as terms that appear frequently. The vector space similarity measure may not be applicable for determining granularity as it places greater weighting for words that appear more often. A high similarity score tells us that the ontology covers terms that appear more frequently. This might correlate to an adequate coverage of the upper (coarse-grained) levels of the ontology (assuming that the higher level concepts and its instances occur more often in the corpus). For example, the term *City* or *Paris* would occur more often than *Chinatown* or *Buttes aux Cailles*. This may explain why the similarity score for $O_{ParisDetailed}$ decreases slightly for the *Paris City Guide* corpus. It may be that terms in the lower (fine-grained) levels of the ontology appear infrequently in the corpus.

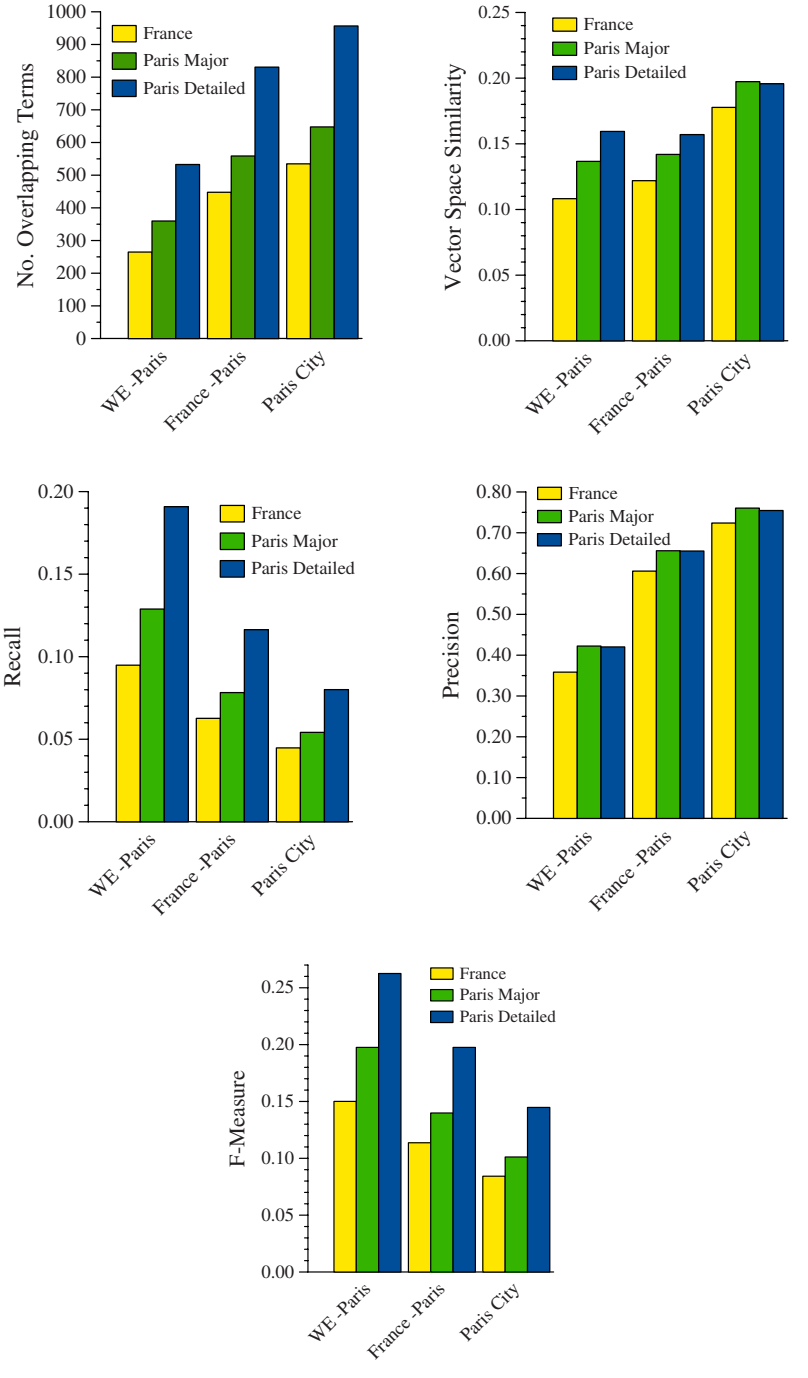


Fig. 5. (a) No. Overlapping Terms, (b) Vector Space Similarity, (c) Recall, (d) Precision and (e) F-Measure

6 Conclusion and Future Work

Given the requirements arising from the LP application domain, there are criteria that can help determine suitable ontologies from a given pool of ontologies – correctness, coverage and to a certain extent, minimal ontological commitment. Also, it was found that the clarity criterion that ensures strictness in an ontology may impede the requirement for flexibility in an ontology's classification in this domain. Some of these criteria may be difficult to evaluate because they can be hard to quantify or there are no measures for their evaluation such as correctness and completeness. Hence for future work, measures should be explored for these criteria so that qualitative and quantitative values can determine the extent to which an ontology can be suited to an application domain.

We have considered two existing measures for coverage (number of overlapping terms and vector space similarity) and proposed new measures for coverage that we adapted from information retrieval (precision, recall and the F-Measure). We found that the vector space similarity measure was not suitable for determining appropriate granularity. Our results were inconclusive regarding the suitability of the other measures. To determine the most appropriate measure for coverage, further work is required with more examples of both corpora and ontologies, as well as other applications. Alternative measures for coverage should also be considered.

Acknowledgements. Ron Gallagher, Bruce Melendy and colleagues at Lonely Planet.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[2] C. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data driven ontology evaluation. In *Proceedings of Int. Conf. on Language Resources and Evaluation*, Lisbon, Portugal, 2004. European Language Resources Association.

[3] J. S. Dong, C. H. Lee, H. B. Lee, Y. F. Li, and H. Wang. A combined approach to checking web ontologies. In *Proceedings of the 13th Int. Conf. on World Wide Web*, pages 714–722. ACM Press, 2004.

[4] A. Gómez-Pérez. Towards a framework to verify knowledge sharing technology. *Expert Systems With Applications*, 11(4):519–529, 1996.

[5] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[6] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, 43(5-6):907–928, 1995.

[7] M. Grüninger and M. Fox. Methodology for the design and evaluation of ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, April 1995.

[8] N. Guarino and C. Welty. Evaluating ontological decisions with ontoclean. *Communications of ACM*, 45(2):61–65, 2002.

[9] E. Hovy. Comparing sets of semantic relations in ontologies. In R. Green, C. A. Bean, and Sung Hyon Myaeng, editors, *The Semantics of Relationships*, pages 91–110. Kluwer, 2002.

[10] P. Krause and D. Clark. *Representing Uncertain Knowledge: An Artificial Intelligence Approach*. Intellect, Oxford UK, 1993.

[11] M. F. López, A. Gómez-Pérez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(5):37–45, January 1999.

[12] D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language Overview*. World Wide Web Consortium (W3C), February 2004. http://www.w3.org/TR/owl-features/.

[13] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[14] R. Studer, V. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *IEEE Trans. on Data and Knowledge Engineering*, 25(1–2):161–197, 1998.

[15] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[16] A. Varzi. Basic problems of mereotopology. In N. Guarino, editor, *Formal Ontology in Information Systems*, pages 29–38. Amsterdam and Oxford: IOS Press, 1998.

[17] A. Varzi. Vagueness in geography. *Philosophy and Geography*, 4(1):49–65, 2001.

A Self-monitoring System to Satisfy Data Quality Requirements

Cinzia Cappiello, Chiara Francalanci, and Barbara Pernici

Politecnico di Milano, Department of Electronics and Information
{cappiell, francala, pernici}@elet.polimi.it

Abstract. Quality of information benefits both on line transactional processing and on line analytical processing. However, quality assurance processes are mostly human intensive and the literature provides limited support to their automation. This paper proposes a rule-based data monitoring and improvement approach as a first step towards self-management of quality of data. These rules specify when to trigger both assessment procedures and improvement actions (e.g. data cleaning), on the basis of the actions performed on the databases and specific quality requirements associated with queries performed by users. They also capture all the events occurring as a consequence of data quality problems and alert the Quality Administrator if human involvement is required. Rules are classified and formalized in the paper. The overall data quality monitoring and improvement process is explained with examples.

1 Introduction

Information quality plays a key role both inside organizations and in inter-organizational relationships. In the former case, information quality reduces errors, increases process efficiency, and improves decision making efficacy. In the latter case, it represents a critical component of each company's trustworthiness. The primary requirement for data quality assurance is the continuous control of data values stored in the operational databases and, possibly, their improvement. This impacts both on line transactional processing and analytical processing. Quality monitoring procedures imply the adoption of algorithms for measuring data quality and automatic techniques for the improvement of data when their quality decreases below acceptable values. Note that acceptable values are dynamically defined as users' requirements.

It is important to have knowledge about the quality of the data stored and used in the organizational processes. In our perspective, a company that mainly provides informational services and receives a large amount of requests, as, for example, in a multichannel environment, has to pay great attention to the quality assessment procedure. In the literature, it has been often stated that the effects of poor data quality can be major if they have an impact on customer dissatisfaction or obstacle the definition of suitable management decisions [13][6]. Organizations can guarantee the high quality of their data by certifying them and proving to the users that their quality requirements have been satisfied.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1535–1552, 2005.

The importance of the certification of the exchanged data is not only valuable for the communication between organization and users but also between organizations, especially, in cooperative environments. With the development of networked economies, organizations are often involved in cooperative processes and in a communication process the exchange of data and of data quality information, helps the creation of trustworthy relationships. These considerations also suggest the need for tools assessing and certifying data quality.

Clearly, in case of poor quality, improvement initiatives have to be undertaken. The literature provides a variety of techniques for data improvement. The most straightforward solution suggests the adoption of data-oriented inspection and rework techniques, such as data bashing or data cleaning [6]. These techniques focus on data values and can solve problems related to data accuracy and data consistency quality dimensions [6]. A fundamental limitation of these techniques is that they do not prevent future errors. They are considered appropriate only when data are not modified frequently [6]. On the other hand, a more frequent use of data bashing and data cleaning algorithms involves high costs that can be difficult to justify. To overcome these issues, several experts recommend the use of process-oriented methods [6][13][14][16][18]. These methods allow the identification of the causes of data errors and their permanent elimination through a change in data access and update activities. These methods are more appropriate when data are frequently created and modified. Organizations can also adopt mixed strategies in which they can decide to adopt a data-oriented technique or a process-oriented technique depending on data and process types.

The improvement phase always needs an accurate and precise data analysis in order to select the best improvement techniques, and to manage the adoption of a process-oriented technique. This suggests the importance of the implementation of new roles in the organization responsible for data quality management. A data quality professional figure, such as a *Quality Administrator*, is required to manage all the knowledge about data structure, operations, and related processes.

Starting from these observations, this paper proposes a framework to perform data quality assessment and improvement in order to satisfy specified requirements. Indeed, the methodology takes as input the levels of data quality both required by the users and specified for each data set. The proposed solution aims at providing a first step towards a self-monitoring system, since the evaluation of quality and consequent improvement activities are triggered based on a set of predefined monitoring rules. Rules can trigger both process-oriented and data-oriented improvement actions as described in Section 4. The overall data quality assessment and improvement methodology is supported by a software architecture called *PoliQual* responsible for data quality management inside the organization.

The paper is organized as follows. Section 2 presents the related work. Section 3 describes the functional model of the proposed architecture. Finally, Section 4 explains the rule-based methodology on which the architecture is based and Section 5 shows some preliminary results obtained in the implementation and test phases.

2 Related Work

Data quality has to be assessed and monitored continuously in order to guarantee high quality levels. To improve quality, organizations can adopt either data-oriented or process-oriented techniques. The former focus on data values and can solve problems related to data accuracy and consistency [6]. In particular, these techniques discover anomalies and inaccuracies by comparing values with benchmarks or by performing local analysis to detect inconsistencies and duplications. Process-oriented methods allow the identification of the causes of data errors and their permanent elimination through an observation of the whole process in which data are involved. Correction activities change data access and update procedures. These methods are more suitable when data are frequently created and modified. They require a considerable effort for process analysis and redesign, but guarantee long-term benefits. The literature provides several frameworks that allow the representation of the processes that manipulate data and the identification of the sub-processes in which data quality decreases [16]. It also describes multiple data quality programs that are based on process-oriented methodologies, such as TDQM (Total Data Quality Management) and TIQM (Total Information Quality Management) [6][18]. These data quality programs are useful, but require a cumbersome personalization effort before application. For example, the primary goal of the TDQM methodology is to provide users with a high level of data quality by considering data as a particular type of manufacturing products. The TDQM methodology cycle consists of four phases in which data quality dimensions are chosen and measured, quality problems are analyzed and improvement techniques are defined. The authors provide a series of guidelines that each organization can customize and apply by developing its own techniques and algorithms. However, this task requires a noticeable effort from enterprises and the literature does not provide any specific methodological support to reduce this effort. In this respect, a fundamental open question is the association of dependable measures with data quality dimensions. The literature does not provide an exhaustive set of metrics that organizations can apply. Only a few algorithms have been developed for a subset of dimensions, such as accuracy, completeness, consistency, and timeliness [1][12][13]. Quality assurance instead needs objective measures of quality, since most users cannot judge the quality of data and simply trust data sources.

Note that data quality assessment is always performed on data values contained in the operational databases. It directly impacts on the efficiency of the organizational processes and it also indirectly increases the efficacy of the processes at the decisional level. For example, in the literature, studies on data warehouses state the importance of the high quality of the data values used in the decision making process [9].

Data quality is also important in inter-organizational relationships. The definition of criteria, metrics, and methodologies to master and manage the quality of the data exported by each autonomous system was also addressed in [15]. This contribution aims at investigating data quality issues in Cooperative Information Systems. Specifically, all cooperating organizations export data quality dimen-

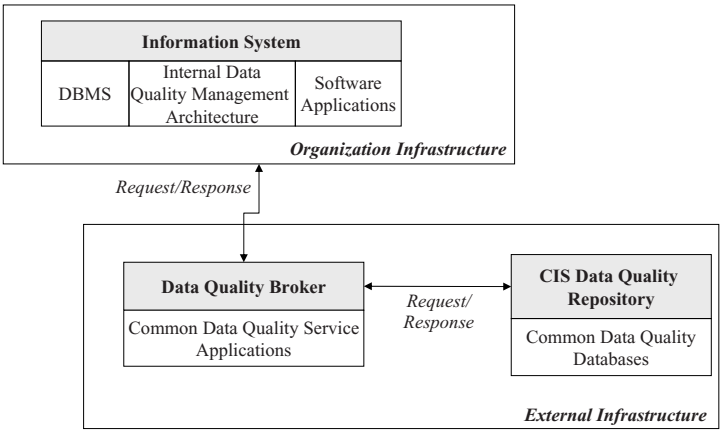


Fig. 1. Data Quality Management Architecture in Cooperative Information Systems

sion values evaluated for the application data according to a specific data model. In the defined data model, each data class is associated with a quality class which specifies the quality dimensions evaluated for the specific data class. Each field belonging to each tuple in the data class is associated with its quality value.

In Cooperative Information Systems (CIS), organizations need to share information, which may have a different level of quality in different organizations. In order to guarantee a consistent high level of data quality, organizations have to implement a common architecture to control and improve the quality of their data. In [4] an architecture composed by an internal infrastructure and an external infrastructure has been proposed (Figure 1).

The external infrastructure is composed of the Data Quality Broker and the CIS Data Quality Repository [15]. It is accessible from all the organizations involved in the Cooperative Information System and it has two important functions: it offers a set of data quality services and controls the data exchanged among organizations. In Figure 1 the information flow in the data quality management architecture is represented; an organization included in the Cooperative Information System sends its information request to the Data Quality Broker that with the collaboration of the CIS Data Quality Repository defines the series of operations involved to retrieve data. The Data Quality Broker offers also a set of services for data improvement and recovery. The internal infrastructure is represented by an internal Data Quality management architecture that has to be implemented in each organization involved in the Cooperative Information system to ensure a good internal data quality management. In this paper a quality management architecture is presented and is called PoliQual. The design of the methodologies and tools underlying the PoliQual work is one of the objectives of this paper and it is discussed in Section 4.

A total data quality management program can be achieved only with continuous data assessment, monitoring, and improvement. Organizations need self-

monitoring tools that can help these activities by tracking and identifying anomalies in data management limiting human intervention. Benefits from the implementation of such tools can be also considered in terms of reputation and trustworthiness increase. Indeed, since the assessment procedures should be able to automatically publish quality metadata, they can be used to produce a quality certificate that associated with data allows attesting to the quality of an organization's data.

The main contribution of this paper is the design of a methodology and tools that aim at supporting a total data quality management program.

3 Data Quality Evaluation

The PoliQual architecture illustrated in this paper allows managing data associating quality information to stored and retrieved data. The basic data and quality model we adopt in this paper assumes a flat structure for data (as in the relational model) and that each data object is associated with its quality dimensions as illustrated in the example in Figure 2 which will be used as main example in the following. The dimensions that are considered in the present work and so far implemented in PoliQual are accuracy, completeness, and timeliness.

Customer Data			***Quality Metadata***		
Name	**Address**	**Trading_op #**	**Completeness**	**Accuracy**	**Timeliness**
....	....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....

Fig. 2. Example - data schema and associated quality metadata

The model we adopt for exporting data and quality data is based on the Data and Data Quality (D^2Q) model [15]. This model refers to the model underlying XML-QL [5], that is a Query Language for XML. In XML-QL data model an XML document is modeled by an XML graph. An XML Graph consists of a graph in which each vertex is represented by a unique string called an object identifier, attributes are associated with nodes, elements are represented by edge labels, and leaves are labeled with values. In the D^2Q model, this model has been extended by adding quality information, stored in metadata and associating it with each data value, thus with each leaf. Therefore, data classes are defined as sets characterized by a name and a series of tuples $\pi_i = <\text{name}_i; \text{type}_i>$ and quality metadata are associated with each tuple.

In our approach, the data quality model is modified and the quality information is built through a hierarchy of metadata. Indeed, it is possible to distinguish first-level metadata MD1 that represent a measure of data quality dimensions and second-level metadata MD2 (meta-metadata) that allow the evaluation of data quality dimensions, that is they represent a metric.

Furthermore, from a database perspective, the D^2Q data model implies the storage of quality metadata related to each field defined in a table. In large databases, the management and especially the analysis of such an amount of metadata is difficult. Since one of the goal of this research is the definition of an architecture that allows continuous quality assessment and monitoring, a high volume of information can be an obstacle to system implementation and efficiency. For this reason, it has been chosen to simplify the model and to perform analysis and monitoring operations on aggregate quality metadata associated with a tuple $\pi_i = <\text{name}_{ij}; \text{name}_{i(j+1)} \ldots \text{name}_{iN}>$. The risk of this approach is based on the compensation effects that an aggregation function, e.g. the average, could introduce. However, in case of very poor quality of a field, the overall quality of the tuple will decrease and it will be probably lower than admissible values. Consequently, the intervention of the Quality Administrator will help to find out where the causes of poor quality are.

As stated before, so far the implemented dimensions are accuracy, completeness, and timeliness that are objective dimensions and, therefore, are suitable for a quantitative evaluation. These three dimensions constitute a minimal set that provides sufficient information about the suitability of data along the process in which they are involved. We assume that quality values can be evaluated on each property, that is a field contained in a database tuple, on the basis of algorithms available in the literature as follows:

- *Accuracy* is defined in [17] as "the extent to which data are correct, reliable and certified". The correctness of data can be defined as a measure of the proximity of a data value v to some other value v' that is considered correct [13]. In our approach, each property is associated with a boolean value that reveals its correctness. The aggregate evaluation function that we consider for the accuracy dimension is the ratio between the number of correct values and the total number of values in a given data set. The value correctness is verified applying data bashing techniques with respect to available dictionaries, after applying data normalization (i.e. *str.* is substituted with *street*).
- *Completeness* specifies the degree to which specific values are included in a data collection[13]. The completeness of each property is evaluated according to the presence of its value. The presence or absence of the value is represented by a boolean value. An aggregate evaluation function of this dimension is obtained calculating the ratio between the number of complete fields and the total number of fields in a given data set.
- *Timeliness* is defined as the property of information to arrive early or at the right time [7]. Timeliness is usually measured as a function of two elementary variables, currency and volatility [1][2]. The measure of timeliness, used in the PoliQual implementation, is defined in [1] as:

$$Timeliness = \max\left[\left(1 - \frac{Currency}{Volatility}\right); 0\right]^s$$

where the exponent s is a parameter necessary to control the sensitivity of timeliness to the currency-volatility ratio. With this definition, the value of

timeliness ranges between 0 and 1. Note that along the hierarchy of metadata presented in the data model, currency and volatility dimensions can be defined as second-level metadata.

The approach can be easily extended to other data quality dimensions and to more complex data structures. In fact, in Section 4 we will discuss the approach to quality assessment and improvement in general, assuming to consider a set Q $= \{qd_j\}$ of data quality dimensions.

In a traditional information system, a user retrieves data by specifying a simple query. In a data quality oriented approach, and above all with certification purposes, users have to specify their own quality requirements. In the methodology proposed in this paper, the model expects that a generic user (internal or external or, in some cases, an application) sends a request to retrieve a given data set together with the specification of quality requirements.

Specifically, a data-quality oriented request is based on the submission of a regular query to organizational databases along with the specification of user quality requirements. Since the metadata are organized in a hierarchical structure, and thus stored in XML files, queries can be expressed in a quality oriented version of XQuery language [10]. For example:

```
for $i in input() where customer[avg(completeness(Name),
completeness(Address), completeness(trading_op))> 0.9]
return($i/Name, $i/Address)
```

The expected result is produced by associating to each tuple the aggregate values of its quality metadata. In Figure 3 is represented the general structure of the result while in Figure 4 an example is shown. Note that the generic user will only receive the aggregate values associated with first-level metadata.

The associated values are those derived from the original tuple. In order to associate an aggregate value for each data quality dimension, an average of quality values on all properties computed to evaluate the aggregate quality value, i.e.:

$$qd_j(tuple_{ik}) = \frac{\sum_{n=1}^{N} qd_j(tuple_{ik}.property_n)}{N}$$

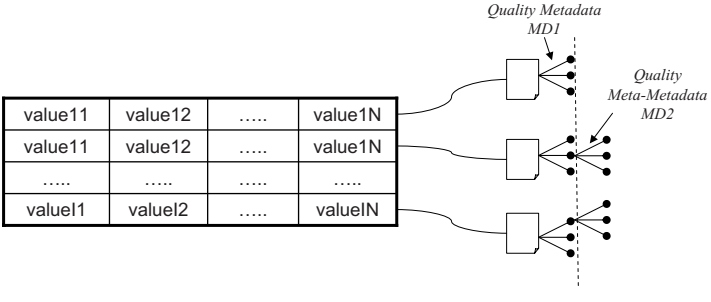


Fig. 3. Structure of the result: data objects and related metadata

Name	Address	Completeness	Accuracy	Timeliness
....	....	.....	.....	.....
.....	.....	.....	.....	.....

Fig. 4. Structure of the result: example

The function $qd_j(tuple_{ik}.property_n)$ will calculate the first level metadata associated the n-th property involved in the k-th tuple included in a database db_i. The evaluation function will consider second-level metadata along the algorithms defined for the quality dimension qd_j.

A more sophisticated model can associate a weight vector to each schema to indicate the relevance of each property in the computing the aggregate value, so that:

$$qd_j(tuple_{ik}) = \sum_{n=1}^{N} w_{jn} \cdot qd_j(tuple_{ik}.property_n)$$

where

$$\sum_{n=1}^{N} w_{jn} = 1$$

In this case, it is supposed that the weights associated with properties are defined by the users along their degree of usefulness.

Furthermore, in order to calculate aggregate values, for some dimensions such as timeliness, the minimum or maximum assessment function can be also used. Indeed, along a data set, the minimum value of timeliness can be more significant than an average value in order to understand the updateness of the whole data set.

Note that we do not consider complex queries that imply, for example, the join of two or more tables. In the literature, information quality assessment algorithms to manage these more complex cases have been proposed only for the completeness dimension [11].

Evaluation of quality of returned tuples can be performed by PoliQual in two ways:

- **On line evaluation:** the property quality values and aggregate quality values are computed along a specific request using the suitable algorithms.
- **Off line evaluation:** aggregate quality values are precomputed and stored in a quality repository on the basis of the model shown in Figure 2.

3.1 On Line Evaluation

The on line evaluation approach involves an additional cost to compute the result of the query, since, differently from generic data, first-level metadata are not simply retrieved from a local database, but are calculated with an evaluation algorithm and thus using, where required, second-level metadata. Further, if the

request involves a large amount of data, response time can be too high. For requests of large data sets, it could be more appropriate to consider the results provided by the off line evaluation process, as discussed in the following.

If an organization cannot satisfy a request with its own data, especially in a cooperative information system, it can request to another organization if it owns suitable data. Indeed, in some cases, there are public or private organizations that are certified for the treatment of a specific type of data and their databases can be used as benchmark values in a quality assessment process. This action is called *on line improvement* and it allows an organization to increase the quality of its own data through the cooperation with other organizations.

3.2 Off Line Evaluation

The off line evaluation differs from the on line evaluation since it is independent of the execution of a particular query. The PoliQual architecture calculates the first-level quality metadata associated with the data belonging to an organization's databases. This calculation is triggered by the rules discussed in Section 4. Rules may trigger the evaluation periodically or as a consequence of a given internal event. An off line evaluation can be requested also by the Quality Administrator to analyze a specific situation that has not been detected automatically. The periodic assessment and storage of quality metadata allow the enterprise to answer users' queries by accessing both the database containing the requested data and the database containing corresponding quality metadata. This reduces the response time to the queries of users, but it may provide out-of-date information about the quality of data. In fact, the stored quality metadata do not take into account all the changes performed in the time interval between two periodic assessments. A critical issue is the definition of the time interval between two periodic assessments in order to maximize the currency of quality metadata. Another critical issue is the unavailability of data during periodic assessments. Indeed, the evaluation operations lock data which, consequently, are not available to the user during the execution of evaluation algorithms. Mechanisms to manage concurrent data access are required.

3.3 Example

In order to clarify the functional model of PoliQual, let us consider the following example. PoliQual receives a query requesting the data describing the customers that did at least a trading operation for which timeliness is greater than 0.8:

```
for $i in input() where customer[trading_op >=1 and
avg(Timeliness(Name), Timeliness(Address),
Timeliness(trading_op))> 0.8] return($i/Name,$i/Address)
```

The response to this query is obtained through the following steps:

1. For each customer that satisfies the query and belongs to the requesting organization's database, the method associated with the on line evaluation is invoked. The method returns the evaluation of the degree of timeliness corresponding to the customer data.
2. If the returned degree of timeliness is greater than 0.8, the corresponding customer will be included in the result set of the query. On the contrary, if the customer record does not satisfy quality requirements, the system will alert the user and store failure details.

4 Quality Monitoring

In the previous section we have discussed how the evaluation of data quality metadata can be performed. In the present section we discuss the decision rules for applying the above mentioned evaluation techniques, focusing in particular on evaluation of quality metadata and improvement actions.

A software architecture, namely PoliQual, has been designed in order to support the improvement, assessment, and maintenance of data quality in each organization and, consequently, to ensure a good internal data quality management. PoliQual is represented in Figure 5. It is composed of four modules: *Quality Analyzer, Quality Assessment, Monitoring*, and *Quality Certification*. The Quality Analyzer module parsers user requests in order to select information taking into consideration users' requests as formulated in Section 3. In case on line evaluation is needed, it invokes the Quality Assessment module. The Quality Certification module associates first-level quality metadata with results. The Monitoring module supports the Quality Administrator, collecting relevant events that may require major improvement actions, and activates off line quality assessment operations. PoliQual supports both automatic evaluation and improvement operations, but also needs the intervention of a Quality Administrator, in particular to decide when and how to perform major data cleaning and improvement operations, involving external resources. In this sense we describe our approach for the self-management of quality of data as semi-automatic, since it is supported by rules that trigger both assessment and improvement phases, but in some cases it requires human intervention.

The rules trigger the interaction among internal modules (*Internal rules*) and between PoliQual and the external components, in particular the Quality Administrator (*External rules*). Internal rules mostly trigger the assessment operations and establish the communication among the modules belonging to PoliQual. The internal rules are completely automatic and human support is limited to the initialization of a few parameters. External rules are strictly related to the improvement phase that often requires a human intervention. The functions performed by PoliQual are discussed in the following.

4.1 Assessment Phase

The assessment phase is composed of on line evaluation and off line evaluation procedures. As regards the **on line evaluation**, a query Q is submitted by a user

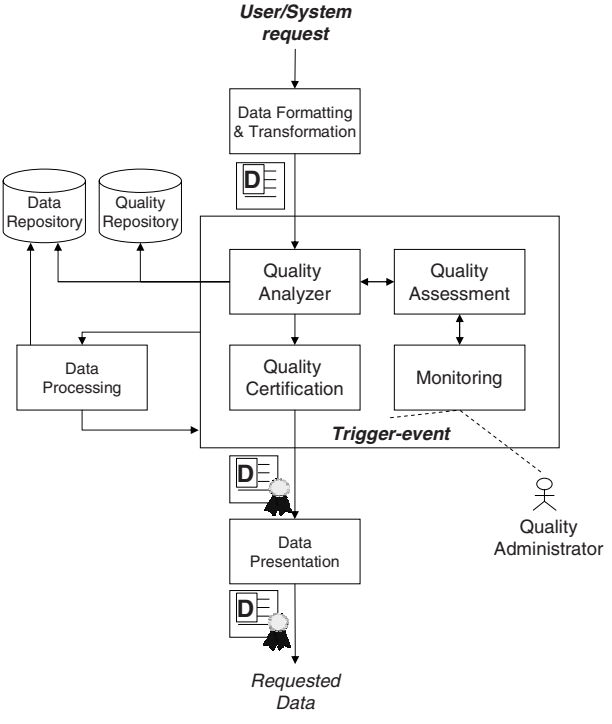


Fig. 5. The PoliQual architecture

and processed by the *Data Formatting & Transformation* module that translates it into a format that can be understood by the Quality Analyzer. The Quality Analyzer analyzes the request and retrieves the query result R from the *Data Repository*. Then, the Quality Analyzer considers user quality requirements. It extracts the information required for quality evaluation from the *Quality Repository*. The Quality Repository contains both the second-level metadata needed for the evaluation and the first-level metadata resulting from the latest off line quality evaluation. Indeed, the retrieval of data quality values can be performed in two ways, as described in the previous section. Let us consider the data answering that are the result R of a query Q submitted by the user. Corresponding quality values can be calculated by activating assessment algorithms that are executed by the Quality Assessment module or can be gathered from the result of the last off line evaluation. An on line evaluation guarantees that the quality metadata are up-to-date, also in case of updates to the tuple and time-varying timeliness. On the other hand such evaluations are expensive in computational terms. Consequently, the discriminating factor between the two approaches is the Computation Time necessary to perform assessment operations. The sum of Computation Time and Delivery Time (td), that is the time necessary to send data to the user, must be lower than the acceptable Service Time (ts) representing the maximum time interval that the user is willing to wait for. Let us

consider a user performing a query Q extracting K tuples (tuple_{i1},... tuple_{iK}) from a database db_i specifying m requirements on corresponding data quality dimensions qd_j. If tc_j is the estimated computation time of quality dimension qd_j for a single record, the assessment phase must satisfy the following rule:

$$\forall j \in [1,m] \;\; ((\sum_{j=1}^{m} tc_j) * K + td < ts \Rightarrow \forall k \in [1,K] \;\; (Assessment(qd_j(tuple_{ik}))))$$

where

qd_j ($tuple_{ik}$)= value of the quality dimension qd_j associated with the tuple_{ik},

Assessment(qd_j ($tuple_{ik}$))= activation of the assessment procedure that evaluates the value $\text{qd}_j(\text{tuple}_{ik})$.

If the rule is not satisfied, then the quality values have to be retrieved from the Quality Repository that contains the results of the last off line assessment:

$$\forall j \in [1,m] \;\; ((\sum_{j=1}^{m} tc_j) * K + td > ts \Rightarrow \forall k \in [1,K](Retrieve(qd_j(tuple_{ik}))))$$

where

Retrieve(qd_j ($tuple_{ik}$))= activation of the procedure that retrieves the value $\text{qd}_j(\text{tuple}_{ik})$.

Using the function Retrieve($\text{qd}_j(\text{tuple}_{ik})$), a critical issue can occur in case a new tuple is created in the database db_i after the latest off line evaluation. In this case the Quality Repository does not contain the quality value associated with the tuple and $\text{qd}_j(\text{tuple}_{ik})$ is equal to null for all data quality dimensions. For this reason, after the retrieval operation another rule is applied in order to control if these anomalies occur:

$$\forall j \in [1,m] \;\; \forall k \in [1,K] \;\; (qd_j(tuple_{ik}) = null \Rightarrow Assessment(qd_j(tuple_{ik})))$$

If user requirements are satisfied and thus the values of all quality dimensions are acceptable, the Quality Assessment notifies the quality values to the Quality Analyzer and the quality certificate is built and associated with the requested data by the Quality Certification module. Before data are sent to the user, they are sent to the Data Processing module, which cooperates with other software applications that are in charge of preparing the final response to the user. The Data Presentation module sends the response to the user according to a specific format.

The results of the Quality Assessment module are used as an input for the Monitoring module: if the values of one or multiple quality dimensions do not satisfy user requirements, the Quality Assessment module sends an alert message to the Monitoring module, which evaluates whether quality improvement actions are needed. Alert messages are regulated by a set of rules, classified as *Process rules*, that support the verification of user quality requirements by the Assessment module. As an example, we can consider a user performing a query Q extracting K tuples (tuple_{i1},...tuple_{iK}) from a database db_i. Let us suppose

that the user requires a value associated with the dimension qd_j be equal or greater than 0.8. The Assessment module considers the following rule:

$$\forall k \in [1, K](\ qd_j(tuple_{ik}) < 0.8$$

$$\Rightarrow Send\_alert(dimension = qd_j.name, tuple_{ik}, qd_j(tuple_{ik}), 0.8))$$

For each tuple that does not satisfy user requirements, an alert message is sent to the Monitoring module specifying the name of the quality dimension that has been assessed, the tuple that does not satisfy requirements, the actual and target values of the quality dimension. The Monitoring module stores all alert messages in a file that can be used for subsequent aggregate analysis supporting the improvement process.

With the **off line evaluation**, as soon as the calculation is triggered, the Monitoring module must communicate to the Quality Assessment module the data that have to be controlled and the set of data quality dimensions that have to be evaluated. The metadata that measure the results of the evaluation are stored in the Quality Repository. The calculation is triggered by a set of internal rules that can be classified as Temporal and Functional:

- *Temporal Rules* are designed to assess data periodically. To define temporal rules, it is necessary to establish the evaluation period, that is the time interval between two off line evaluations. The evaluation period should be defined by considering both the update frequencies of source databases and the source availability requirements. Indeed, frequent data changes should be associated with a shorter evaluation period but it is also necessary to consider that an off line evaluation procedure disables the accesses to the system for the all the execution period. Let t and t_i be the current time and the time instant in which the last off line evaluation has been performed on database db_i. The database contains K tuples (tuple_{i1},...tuple_{iK}), qd_j a quality dimension and $\Delta \mathrm{t}_{ij}$ the evaluation period associated with db_i and qd_j. An example of temporal rule is:

$$\forall t \forall i \forall j\ \ (t = (t_i + \Delta t_{ij}) \Rightarrow \forall k(Assessment(qd_j(tuple_{ik}))))$$

  Note that the off line evaluation can be triggered on a subset of a database by specifying a query in the corresponding temporal rule. It can be also triggered on a specific set of quality dimensions with different time periods by defining multiple temporal rules.
- *Functional rules* are defined to capture events occurring as a consequence of a modification in a database. These events are captured by monitoring update, create, and delete operations. Usually, the off line evaluation is not performed at each modification of a database, due to the high costs of assessment operations and system unavailability. It is more advisable to perform the off line evaluation after a predefined number of modifications N. This number can refer to the number of data updates, creations, and deletions and it is determined by the Quality Administrator along a database analysis. Alternatively, functional rules can include different thresholds for

different types of modifications. In PoliQual, modifications are associated with records, as opposed to fields. Therefore, functional rules count a single modification even if multiple fields are changed in the same record. Let us consider Updated(tuple_{ik}), as a function that is equal to the number of times that the tuple_{ik} has been updated, and a database db_i an example of functional rule is:

$$\sum_{k=1}^{K}(Updated(tuple_{ik})) > N_{update} \Rightarrow \forall j \forall k(Assessment(qd_j(tuple_{ik})))$$

where N_{update} represents the number of update operations triggering the off line evaluation. After the assessment of db_i, $\forall$k Updated(tuple_{ik}) is set to 0.

Temporal and functional rules can be combined as follows:

$$\forall t \forall i \forall j \ (\sum_{k=1}^{K}(Updated(tuple_{ik})) > N_{update} \vee (t \geq t_i + \Delta t_{ij}) \Rightarrow \forall k(Assessment(qd_j(tuple_{ik}))))$$

In this case, the count of update operations should start from t_i.

4.2 Improvement Phase

The rules that support the improvement phase are particularly relevant. They trigger the communication between the Monitoring module and the Quality Administrator to support the improvement phase. Note that the improvement phase is not completely automatic. PoliQual can suggest the type of improvement actions to undertake and the time period to perform them. The Quality Administrator should analyze critical situations and perform the most appropriate improvement actions. In the previous section, it has been described how the Monitoring module stores alert messages generated by low quality values. When the number of user requests that are not satisfied is high, data should be improved. An overall low quality of data for a high number of users can affect the quality of business services [13] and the cost of improvement actions can be justified. When an improvement process is needed, the Monitoring module sends a request to the Quality Administrator suggesting a thorough analysis of data. When the user request that extracts K tuples (tuple_{i1},...tuple_{iK}) from a database db_i is not satisfied and the alert message is sent to the Monitoring module, the Monitoring module counts the number of alerts on all the detected tuples and activates the analysis request message considering the following rule:

$$\forall k \in [1, K], \forall j \ (\sum(Alert(qd_j.name, tuple_{ik})) > N_{alert} \Rightarrow Analysis(dimension = qd\_name, tuple_{1k}))$$

where N_{alert} represents the number of alert messages indicating a critical quality situation that requires a detailed analysis.

The analysis performed by the Quality Administrator should evaluate whether the divergence between user requirements and actual quality values is critical. If it is considered critical, an improvement action is required. As discussed in the previous section, improvement actions are based on either data-oriented or process-oriented techniques. The former are appropriate when data are not modified frequently, as they are expensive and have short-term effects. If data are not changed frequently, they need the application of inspection and rework techniques, such as data bashing or data cleaning. These techniques focus on data values and can solve problems related to data accuracy, consistency, and completeness [6]. It is possible to define rules to support the analysis conducted by the Quality Administrator in order to identify the most suitable improvement technique and to activate it. For example, when the Monitoring module sends to the Quality Administrator an analysis request for a tuple, the Quality Administrator can activate rules to trigger the data-oriented tools implemented by the organization verifying whether data-oriented tools are suitable for those data. Since the improvement technique are in general applied to the whole data set db_i in which the tuple is included, these rules consider all the tuples in db_i. Let us consider two functions:

- Created($tuple_{ik}$): function that is equal to 1 if the $tuple_{ik}$ has been created between the current time instant t and a fixed time t_i ($t_i < t$) and 0 otherwise.
- Updated($tuple_{ik}$): function that is equal to the number of times that the $tuple_{ik}$ has been updated between the current time instant t and a fixed time t_i, ($t_i < t$).

A rule activating improvement tools can be:

$$\frac{\sum_{k=1}^{K} Updated(tuple_{ik}) + \sum_{k=1}^{K} Created(tuple_{ik})}{t - t_i} < F \Rightarrow Activate(data\_oriented\_tool(db_i))$$

where F measures the threshold number of modifications to db_i in a specific time interval. If a database contains critical data and is frequently accessed by users, process-oriented approaches to data improvement are more appropriate, since they prevent future errors with a long-term effect. If the rule above is not satisfied, that is the frequency of changes is greater than F, then, process-oriented improvement initiatives should be activated by the Quality Administrator to identify the causes of data errors and eliminate them permanently. This requires the observation of the processes in which data are involved. Improvement actions change data access and update activities through process analysis and redesign. In our architecture, this type of improvement methods is supported by the History dimension. This dimension tracks the time evolution of the quality of a data set in order to identify which operations have improved or worsened quality values and, thus, build a historical database that can be used for statistical evaluations and process improvement. The History dimension stores all the events that have caused a data modification, from creation to deletion. For each data modification the information that has to be stored in the History is the name of the user performing the modification, date, hour, and type of

operation (creation, update or deletion) and the percentage of data quality variation. The analysis of the History file helps the Quality Administrator to identify the processes that have to be analyzed and redesigned to obtain a permanent improvement of data quality.

5 Implementation and Test Results

The functionalities of the architecture proposed in this paper have been implemented as Web services. This implementation has a simple Web interface that allows users to access services through a common Web browser. The architecture has been implemented with JAVA Server Pages (JSP) and JAVA Servlet. Microsoft Access and SQL have been used to define the structure of databases and queries, respectively. In order to provide efficient and scalable Web services, the system has been designed with independent data, application, and presentation layers. Not all the quality dimensions listed in the previous sections have been implemented. So far, the implementation is complete for timeliness, completeness, accuracy, and history.

The algorithms used to assess timeliness, completeness, accuracy, and history have been tested on sample databases. As discussed in the previous sections, quality dimensions can be evaluated in on line or off line mode. Testing has pointed out that the on line evaluation can provide real-time quality values, but if the assessment algorithms are performed on line on large amounts of data, time response can be excessively high. In this respect, the off line evaluation is preferable, although it provides quality values valid at the time of the last assessment.

Our analysis have pointed out that for certain quality dimensions, such as timeliness and completeness, the average response time per record is about 1 s, the average computation time per record is about 400 ms and the delivery time that is the average time for the generation of the JSP page is estimated as 600 ms. As the number of records increases, the average computation time grows linearly, since the algorithms evaluating quality dimensions are characterized by linear complexity [3]. Delivery time is almost constant. By applying the rule defined in Section 4, it is possible to calculate a rough threshold value for the number of records. If query results involve a number of tuples below the threshold value, it is suggested to perform the on line assessment, otherwise it is advisable to retrieve the values stored by the last off line evaluation. On the contrary, for the accuracy dimension, it is better to perform the off line evaluation, as calculations consider all the records in the databases and algorithms are exponentially complex [8]. Although tests have been performed on a database of small dimensions (4000 records), in order to obtain the most correct and reliable results, and thus to find the maximum number of errors, the execution time is 15 minutes. Note that tests have been performed on a Pentium III 650 Mhz and 192MB RAM.

6 Concluding Remarks

The paper has presented a framework for data quality self-management oriented to satisfy users' quality requirements. A number of quality dimensions are sup-

ported, including assessment and data quality tools, on flat data structures, such as for instance relational tables. Evaluation of quality has been performed on tuples, and quality requirements on queries are expressed on single tables.

A number of possible future research directions are possible.

A first direction is towards including in the framework evaluation of quality of complex data structures and evaluating quality of results of queries which are derived from more than one base table. Algorithms have been proposed in the literature for some dimensions, such as completeness [12] while measurement of other quality dimensions in complex queries needs still further investigations.

Further investigation is needed also for the rule based approach for assessment. While some experimental data provide some evidence on choosing between on line and off line assessment, further research is needed to relate these evaluations to the evolving nature of data: some data are static, and in general do not need further evaluation in time, such as names and birth dates, while others are heavily dependent on underlying processes (e.g. Stock quotes). More elaborated criteria for deciding whether to perform an on line evaluation should be studied. In addition, further investigation is needed on the relationships between rules, to guarantee that a correct answer, i.e. satisfying quality requirements, is given to users most of the times, and that on line and off line evaluations do not require a high percentage of rework performing already available assessments.

The proposed framework is a first attempt at providing an integrated set of tools for a systematic data quality management.

Acknowledgements

This work has been partially supported by the Italian FIRB Project MAIS.

References

1. D.P. Ballou, R.Y. Wang, H.L. Pazer, and G.K. Tayi. Modelling information manufacturing systems to determine information product quality. *Management Science*, 44(4), 1998.
2. M. Bovee, R.P. Srivastava, and B. Mak. A conceptual framework and belief- function approach to assessing overall information quality. In *Proceedings of the Sixth International Conference on Information Quality*. MIT Press, November 2001.
3. C. Cappiello, C. Francalanci, and B. Pernici. Time-related factors of data quality in multichannel information systems. *Journal of Management Information Systems*, 20(3):71–91, 2004.
4. C. Cappiello, C. Francalanci, B. Pernici, P. Plebani, and M. Scannapieco. Data quality assurance in cooperative information systems: a multi-dimension quality certificate. In *Proceedings of the International Workshop on Data Quality in Cooperative Information Systems (DQCIS 2003)*, January 2003.
5. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and Suciu D. Xml-ql: A query language for xml. In *Proceedings of the 8th International World Wide Web Conference*, 1999.
6. L.P. English. *Improving Data Warehouse and Business Information Quality*. John Wiley & Sons, 1999.

7. M.J. Eppler. *Managing Information Quality*. Springer-Verlag, 2003.
8. M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings ACM SIGMOD International Conference Management of Data*, 1995.
9. M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of Data Warehouse*. Springer-Verlag, 2000.
10. D. Milano, M. Scannapieco, and T. Catarci. Quality-driven query processing of xquery queries. In *Proceedings of the International Workshop on Data and Information Quality (DIQ'04) in conjunction with the CAiSE*, pages 78–89, 2004.
11. F. Naumann. *Quality-Driven Query Answering for Integrated Information Systems*. LNCS 2261, 2002.
12. F. Naumann, J. C. Freytag, and U. Leser. Completeness of integrated information sources. *Information Systems*, 29(7):583–615, 2004.
13. T.C. Redman. *Data Quality for the Information Age*. Artech House, 1996.
14. M. Scannapieco, E. Pierce, and B. Pernici. Ip-uml: Towards a methodology for quality improvement based on the ip-map framework. *AMIS (Advances in Management Information Systems) Monograph on Information Quality*, 2005.
15. M. Scannapieco, A. Virgillito, M. Marchetti, M. Mecella, and R. Baldoni. The daquincis architecture: a platform for exchanging and improving data quality in cooperative information systems. *Information Systems*, 29(7):551–582, 2004.
16. G. Shankaranarayan, R. Y. Wang, and M. Ziad. Modeling the manufacture of an information product with ip-map. In *Proceedings of the 6th International Conference on Information Quality*, 2000.
17. Y. Wand and R.Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communication of the ACM*, 39(11), 1996.
18. R.Y. Wang. A product perspective on total data quality management. *Communications of the ACM*, 41(2), 1998.

An Ontology- and Resources-Based Approach to Evolution and Reactivity in the Semantic Web

Wolfgang May[1], José Júlio Alferes[2], and Ricardo Amador[2]

[1] Institut für Informatik, Universität Göttingen
[2] Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa

Abstract. The Web of today can be seen as an active and heterogeneous infrastructure of autonomous systems, where reactivity, evolution and propagation of information and changes play a central role. In the same way as the main driving force for XML and the Semantic Web idea was the heterogeneity of the underlying data, the heterogeneity of concepts for expressing behavior calls for an appropriate handling on the semantic level. We present an ontology-based approach for specifying behavior in the Semantic Web by Event-Condition-Action (ECA) rules that models rules as well as their event, condition, and action components, and languages as resources. The necessary information about semantics and suitable processors is then associated with the language resources. The approach makes use of the data integration facilities by URIs that allow for a seamless integration of information and services physically located at different places. Additionally, that point of view allows for sharing and reuse of these resources throughout the Semantic Web.

1 Introduction

The current Web does not only consist of HTML pages, but of *nodes*, some of which are still browsing-oriented, but in general also providing behavior (often summarized as *Web services*). With this, the perspective shifts more to the idea of the Web as a network of (autonomous) *information systems*. Current portals usually integrate a fixed set of known sources, often using "hard-coded" integration. A problem when overcoming this restriction is its *heterogeneity*, both in the actual data formats, and also *semantic heterogeneity*. The goal of the *Semantic Web* is to bridge this heterogeneity and provide unified view(s) on the Web. In this scenario, XML (as a format for storing and exchanging data), RDF (as an abstract data model for states), OWL (as an additional framework for state theories), and XML-based communication (Web Services, SOAP, WSDL) provide the natural underlying concepts.

In contrast to the current Web, the *Semantic Web* should be able not only to support querying, but also to propagate knowledge and changes in a semantic way. This *evolution* and *behavior* depends on the cooperation of nodes. In the same way as the main driving force for XML and the Semantic Web idea was the heterogeneity of the underlying data, the heterogeneity of concepts for expressing *behavior* requires an appropriate handling on the semantic level. Since the contributing nodes are prospectively based on different concepts such as data

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1553–1570, 2005.

models and languages, it is important that *frameworks* for the Semantic Web are modular, and that the *concepts* and the actual *languages* are independent.

Here, *reactivity* and its formalization as *Event-Condition-Action (ECA) rules* provide a suitable model because they provide a modularization into clean concepts with a well-defined information flow. An important advantage of them is that the *content* (event, condition, and action specifications) is separated from the *generic semantics* of the rules themselves. They are easy to understand, and provide a well-understood formal semantics: when an event (atomic event or composite event) occurs, evaluate a condition, and if the condition is satisfied then execute an action (or a sequence of actions, a program, a transaction, or even start a process). ECA rules provide a generic uniform framework for specifying and implementing communication, local evolution, policies and strategies, and –altogether– global evolution in the Semantic Web.

In the present paper, we develop an ontology-based approach for describing (reactive) behavior in the Web and evolution of the Web that follows the ECA paradigm. We propose a modular framework for *composing* languages for events, queries, conditions, and actions, as well as application-specific languages and ontologies for atomic events and actions. Modularity allows for high flexibility wrt. the heterogeneity of the potential sublanguages, while exploiting and supporting their meta-level *homogeneity* on the way to the Semantic Web.

Structure of the Paper. The remainder of the paper is structured as follows: In Section 2, we analyze the notion of *state* in the Semantic Web and the consequences for the design of the ECA framework for describing *behavior* in the Semantic Web. The rule level of the ontology is then presented in Section 3, including the coarse level of an XML Markup. Section 4 deals with the integration of trigger-like ECA rules "below" the semantical level in the homogeneous local environments of nodes. The Semantic Web level is then refined with a more detailed analysis of the event, query, condition, and action concepts in Section 5; leading to a refined XML Markup. The architecture of the realization of the framework in the Semantic Web based on the "actual" *resources* (e.g., language processors that are associated with the language resources) of the RDF ontology is described in Section 6, followed by a short conclusion.

2 Rules in the Semantic Web: Requirements Analysis

2.1 States and Nodes in the Semantic Web

As described above, the *Semantic Web* can be seen as a network of autonomous (and autonomously evolving) nodes. Each node holds a *local* state consisting of extensional data (facts), metadata (schema, ontology information), optionally a knowledge base (intensional data), and, again optional, a behavior base. In our case, the latter is given by the ECA rules under discussion that specify which actions are to be taken upon which events and conditions.

The state of a node in the Semantic Web is represented in XML, RDF(S), and/or OWL. Usually, XML serves for the physical level, mapped to an RDF/

OWL ontology for integration throughout the Web. Here, a framework where the behavior base (i.e., the ECA rules) is also part of the Semantic Web and represented in a declarative way (and can be queried, reasoned about, and updated), will in our view prove useful.

2.2 Behavior of Nodes

Cooperative and reactive behavior is then based on events (e.g., an update at a data source where possibly others depend on). The depending resources detect events (either they are delivered explicitly to them, or they poll them via the communication means of the Semantic Web). Then, conditions are checked (either simple data conditions, or e.g. tests if the event is relevant, trustable etc.), which can include queries to one or several nodes. Finally, an action is taken (e.g., updating own information accordingly).
The behavior has to take into account the distributed state of knowledge:

1. behavior can be local to a node, i.e., all events are explicitly detectible at the node, the condition is a query against local data, and the actions are also local ones;
2. conditions can include queries that may involve other nodes (either explicitly addressed, or by evaluating a Semantic Web query);
3. actions can also effect other nodes; again either by sending an explicit message to a certain node, or as an *intensional* update against "the Web". Such updates are expressed and sent as messages, and appropriate nodes will react upon their receipt by updating their local database;
4. there are also relevant events that are only detectible at other nodes, or intensional events "on the Web"; also, event combinations (from possibly different sources) have to be taken into account.

With these extensions, together with the "Semantic" property of the rules, the ECA concept needs to be more flexible and adapted to the *global* environment. Since the Semantic Web is a world-wide living organism, nodes "speaking different languages" should be able to interoperate. So, different "local" languages, be it the query languages, the action languages or the event languages/event algebras have to be integrated in a common framework. In contrast to "classical" ECA rules, our approach makes a more succinct separation between event, condition, and action component, which are possibly (i) given in separate languages, and (ii) possibly evaluated/executed in different places. Since not every node will provide ECA capabilities, there will also be nodes that provide "ECA services" (extending the concepts of *publish-subscribe* and *continuous query* services), being able to execute ECA rules (submitted by any Semantic Web participants) that use arbitrary sublanguages (see Section 6).

The requirement (4) also calls for application-dependent handling and detection of atomic events. In general, each application ontology must also specify how derived events can be detected based on actual events (e.g., "account X goes below zero" based on "a debit to account X", or "flight LH123 on 10.8.2005 is fully booked" from "person P is booked for seat 5C of flight LH123 on 10.8.2005").

3 Ontology of Rules and Languages

In usual Active Databases in the 1990s, an ECA language consisted of an event language, a condition language, and an action language. In the Semantic Web, there is a heterogeneous world of such languages which has to be covered. The target of the development and definition of languages for (ECA) rules, events, conditions and actions in the Semantic Web should be a semantic approach, i.e., based on an (extendible) ontology for these notions that allows for *interoperability* and also turns the instances of these concepts into objects of the Semantic Web itself. Thus, in the Semantic Web, we do not have a unique ECA language that consists of three such languages, but there is the semantic concept of an *ECA rule* as shown as an UML diagram in Figure 2. The event, query/test, and action components are described in appropriate languages, and ECA rules can use and combine such languages flexibly. The model is accompanied by an XML ECA rule (markup) language, called ECA-ML.

Components of Active Rules in the Semantic Web. A basic form of active rules are the well-known *database triggers*, e.g., in SQL, of the form

ON *database-update* WHEN *condition* BEGIN *pl/sql-fragment* END.

In SQL, *condition* can only use very restricted information about the immediate database update. In case that an action should only be executed under certain conditions which involve a (local) database query, this is done in a procedural way in the *pl/sql-fragment*. This has the drawback of not being declarative; reasoning about the actual effects would require to analyze the program code of the *pl/sql-fragment*. Additionally, in the distributed environment of the Web, the query is probabyl (i) not local, and (ii) heterogeneous in the language – queries against different nodes may be expressed in different languages. For our framework, we prefer a *declarative* approach with a *clean, declarative* design as a "Normal Form": Detecting just the dynamic part of a situation (event), then check *if* something has to be done by probably obtaining additional information by a query and then evaluating a *boolean* test, and, if "yes", then actually *do* something – as shown in Figure 1.

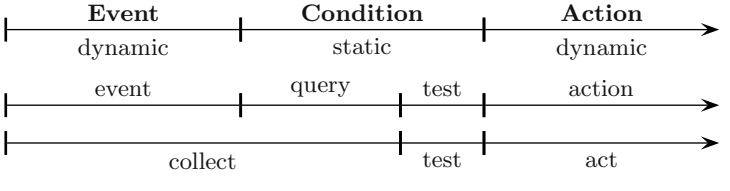


Fig. 1. Components and Phases of Evaluating an ECA Rule

With this further separation of tasks, we obtain the following structure:

- every rule uses an event language, one or more query languages, a test language, and an action language for the respective components,
- each of these languages and their constructs are described by metadata and an ontology, e.g., associating them with a processor,

– there is a well-defined *interface* for communication between the E, Q&T, and A component by variables (e.g., bound to XML or RDF fragments).

Sublanguages and Interoperability. For applying such rules in the Semantic Web, a uniform handling of the event, query, test, and action sublanguages is required. For this, rules and their components must be objects of the Semantic Web, i.e., described in XML or RDF/OWL in a generic *rule ontology* describing the UML model shown in Figure 2.

The modular structure requires a communication of parameters between the rule components. We propose to use rule-wide logical variables for a communication flow according to Figure 1: Variables can be bound in the event component and in the subsequent query component; previously bound variables can also be used here. Variables occurring several times are interpreted as join variables, requiring all occurrences to be bound to the same value. The test and action components use these variables.

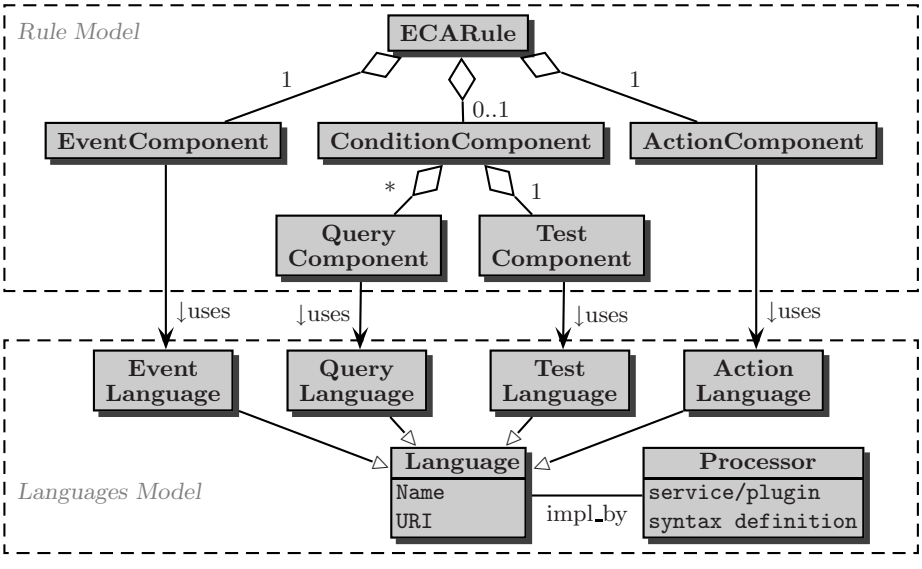


Fig. 2. ECA Rule Components and Corresponding Languages

Based on this ontology, we propose the following markup (ECA-ML):

```
<eca:rule rule-specific attributes>
  rule-specific contents, e.g., declaration of logical variables
  <eca:event identification of the language >
    event specification, probably binding variables; see Section 5.2
  </eca:event>
  <eca:query identification of the language >     <!-- there may be several queries -->
    query specification;   using variables, binding others; see Section 5.3
  </eca:query>
```

```
<eca:test identification of the language >
  condition specification, using variables; see Section 5.3
</eca:test>
<eca:action identification of the language >
  action specification, using variables, probably binding local ones; see Section 5.3
</eca:action>
</eca:rule>
```

The actual languages (and appropriate services etc.) are identified by namespaces and their declarations in the <eca:...> elements (see Example 2 later).

A similar markup for ECA rules (without separating the query and test components) has been used in [BCP01] with *fixed* languages (a basic language for atomic events on XML data, XQuery as query+test language and SOAP in the action component). This fixed approach falls short wrt. the language heterogeneity, and especially the use and integration of languages for composite events. The XChange approach [BP05] also uses fixed languages for specifying the event, condition, and action component. In contrast, the approach proposed here allows for using *arbitrary* languages. Thus, these other proposals are just *two* possible configurations. Our approach even allows to mix components of both these proposals.

Languages, Rules, and Rule Components as Resources. For the Semantic Web, both the languages, and the instances (i.e., rules, rule components, and also e.g. subevents) are *resources*. This allows to *reuse* rules and to recombine subparts (e.g., events) in several rules.

For the architecture, we propose to use a completely modular concept: the framework for the ECA rules must allow to *plug in* or *connect to* detection engines for events (atomic events such as simple data updates or incoming messages, application-level events and *composite events* such as "if first A happens and then B"), processors for queries, and processors for actions (including updates, intensional updates, and transactions): For each rule, the event, query, test, and action components contain *references* to the corresponding languages as resources, given as a URI, similar to XML's namespaces. These resources in turn are associated with further resources related to the language, e.g., a DTD or XML Schema, an ontology description (e.g., in OWL), and a language processor (e.g., as a Web Service). We come back to this issue in Section 6 where we show that in the end, the processing of each contributing sub-ontology can be associated with separate engines or nodes in the Web, i.e. (anticipating the analysis of the subsequent sections):

- ECA rule processors
- underlying database engines with local, trigger-like rules,
- detection mechanisms for (application-independent) event algebras,
- for each application ontology, detection mechanisms of application-level events,
- query processors,

– services for Web transactions, and
– application-level actions (explicit or intensional actions).

ECA Rules in the Semantic Web are required on several abstraction levels (programming language/data structure level, logical level, and semantic level), and with different scope (local or global). In most of the rules on the higher levels (global, referring to the application ontology), the event, query, test, and action components are subject to heterogeneity. But, there are also mostly simple local rules e.g., for maintaining local consistency, mappings to the underlying data model, and reactions, that work in a homogeneous, local environment, where the "classical" ECA paradigm is sufficient. We first integrate these *trigger-like* rules into the framework and come back to the heterogeneous Semantic Web case in Section 5.

4 Trigger-Like Rules

The base level is provided by rules on the *programming language and data structure level* that react directly on changes of the underlying data. Usually they are implemented inside the database as *triggers*, e.g., in SQL, of the form

ON database-update WHEN condition BEGIN pl/sql-fragment END.

In the Semantic Web, the data model level is assumed to be in XML or RDF format. While the SQL triggers in relational databases are only able to react on changes of a tuple or an attribute of a tuple, the XML and RDF models call for more expressive event specifications according to the (tree or graph) structure.

Events on XML Data. Work on triggers for XQuery has e.g. been described in [BBCC02] with *Active XQuery* (using the same syntax and switches as SQL, with XQuery in the action component) and in [BPW02, PPW03], emulating the trigger definition and execution model of the SQL3 standard that specifies a syntax and execution model for ECA rules in relational databases. In [ABB+05], we developed the following proposal for atomic events on XML data:

– `ON {DELETE|INSERT|UPDATE} OF` *xsl-pattern*: if a node matching the *xsl-pattern* is deleted/inserted/updated,
– `ON MODIFICATION OF` *xsl-pattern*: if anything in the subtree rooted in a node matching the *xsl-pattern* is modified,
– `ON INSERT INTO` *xsl-pattern*: if a node is inserted (directly) into a node matching the *xsl-pattern*,
– `ON {DELETE|INSERT|UPDATE} [SIBLING] [IMMEDIATELY] {BEFORE|AFTER}` *xsl-pattern*: if a node (optionally: only sibling nodes) is modified (immediately) before or after a node matching the *xsl-pattern*.

All triggers should make relevant values accessible, e.g., `OLD AS ...` and `NEW AS ...` (like in SQL), both referencing the node to which the event happened, additionally `INSERTED AS`, `DELETED AS` referencing the inserted or deleted node.

Similar to the SQL `STATEMENT` and `ROW` triggers, the granularity has to be specified for each trigger:

- `FOR EACH STATEMENT` (as in SQL),
- `FOR EACH NODE`: for each node in the *xsl-pattern*, the rule is triggered only at most once (cumulative, if the node is actually concerned by several matching events) per transaction,
- `FOR EACH MODIFICATION`: each individual modification (possibly for some nodes in the *xsl-pattern* more than one) triggers the rule.

The implementation of such triggers in XML repositories can e.g. be based on the *DOM Level 2/3 Events* or on the triggers of relational storage of XML data.

Events on RDF Data. RDF triples describe properties of a resource. In contrast to XML, there is no subtree structure (which makes it impossible to express "deep" modifications in a simple event), but there is metadata. A proposal for RDF events can be found in RDFTL [PPW03, PPW04]. The following proposal has been developed in [ABB+05]:

- `ON {INSERT|UPDATE|DELETE} OF` *property* `[OF` *class*`]`.

If a property is added to/updated/deleted from a resource (optionally: of the specified class), then the event is raised. Additionally,

- `ON {CREATE|UPDATE|DELETE} OF` *class* is raised if a resource of a given class is created, updated or deleted.

On the RDF/RDFS level, also metadata changes are events:

- `ON NEW CLASS` is raised if a new class is introduced,
- `ON NEW PROPERTY [OF CLASS` *class*`]` is raised, if a new property (optionally: to a specified class) is introduced.

Besides the `OLD` and `NEW` values mentioned for XML, these events should consider as arguments (to bind variables) Subject, Property, Object, Class, Resource, referring to the modified items (as URIs), respectively.
Trigger granularity is `FOR EACH STATEMENT` or `FOR EACH TRIPLE`.

Integrating Triggers into the ECA Ontology: Opaque Rules. In the above trigger-like cases, the languages for specifying the *event*, *condition* and *action* components are the database-level events, and the local query and update languages. For that, from the implementation point of view, the trigger rule as a whole does not require an explicit markup but can be expressed in its native syntax. In our ontology, we embed this as *opaque* rules, see Figure 3 and the following XML markup:

```
<eca:rule>
  <eca:opaque xmlns:foo="uri of the trigger language">
    <foo:trigger>
      ON database-update WHEN condition BEGIN action END
    </foo:trigger>
  </eca:opaque>
</eca:rule>
```

Since such opaque rules are ontologically "atomic" objects, their event, condition, and action components cannot be accessed by Semantic Web concepts. Thus,

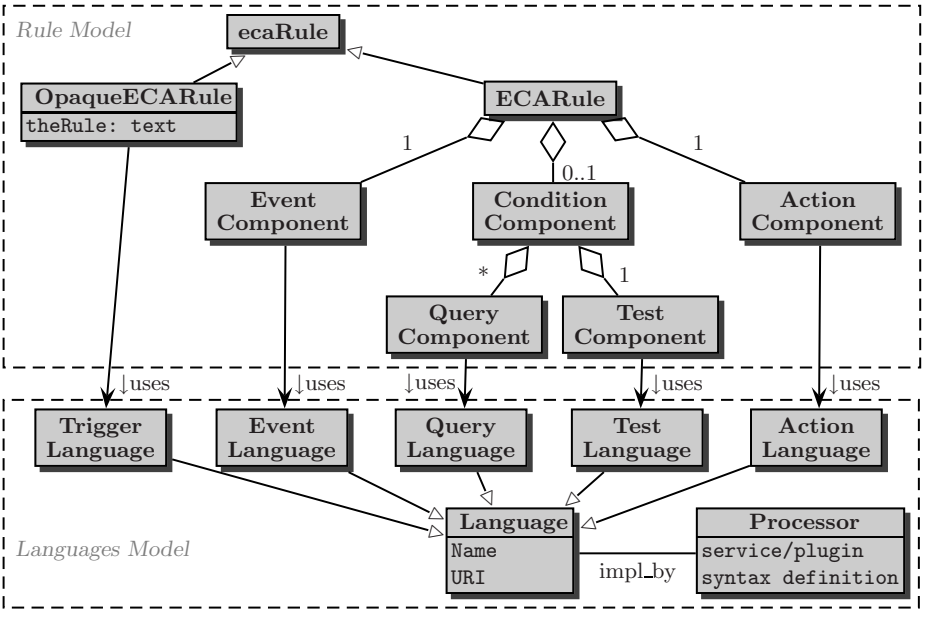


Fig. 3. ECA Rule Components and corresponding Languages II

there is no reuse, and no support for rule analysis. The database-level events on XML or RDF data can also be seen as atomic events in the sense of non-opaque ECA rules; thus, it is also possible to *lift* the triggers to the ontology level and represent them as full-fledged ECA rules.

5 Ontologies for the Rule Components

In the subsequent sections, we develop more detailed ontologies for the event, query, test, and action components, with special emphasis on the event component (the others are similar).

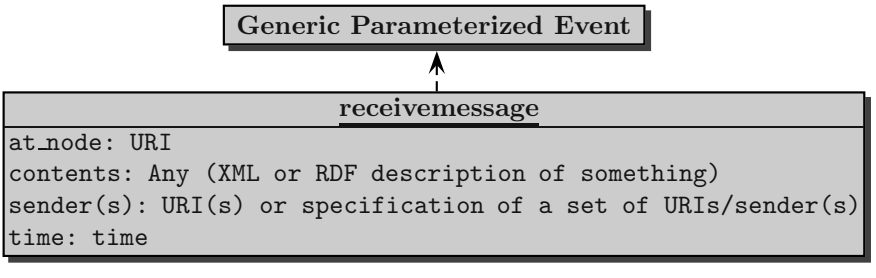
5.1 Ontology of Events

The ontological structure of the event component consists of three subclasses (see explanations below and Figure 4):

- atomic events, that again split into several subtypes:
  - data level events as those discussed in Section 4,
  - events of a given application domain, e.g., in banking, travel organizing, administration; such atomic events are described in terms of the ontologies of the application domain,
  - generic parameterized events that instantiate generic event patterns, e.g., "receive a message about ...".
- *composite* events, e.g.: "A or B", "A and B", or "A and then B".

Atomic Application Level Events. Atomic application-level events are the visible happenings in the application domain. Note that in contrast to the above data-level trigger events on XML or RDF data, there is an important difference between *actions* and *events*: an event is a visible, possibly indirect or derived, consequence of an action. E.g., the action is to "debit 200E from Alice's bank account", and visible events are "a debit of 200E from Alice's bank account", "a change of Alice's bank account" (that are immediately detectable from the update operation), or "the balance of Alice's bank account becomes below zero", which has to be derived from an update. Note that application ontologies have to describe the relationship between actions and resulting events. Orthogonal to being derived or not, application-level atomic events can be associated with a certain node (e.g., "if Springer publishes a textbook on the Semantic Web") or can describe happenings on the Web-wide level (e.g., "if a textbook on the Semantic Web is published"). In the latter case, event detection is even more complicated since it must also be searched and derived *where* and *how* the event can be detected. This is not the task of the rule execution, but of application-level reasoning, based on the application ontology. With this, *application-level* rules (i.e. reacting on application-level *global* events) like *business rules* can be described.

Generic Parameterized Events. Generic Parameterized Events are patterns of atomic events that are ontologically independent from the actual application. The most prominent ones are concerned with communication, i.e., receiving and sending messages, or transactional events. Note that also sending of a message can be a relevant event to trigger other rules, e.g., for policies (waiting for an answer for 10 minutes, then sending it again), or "listening" and deriving other events. In general, such events are associated with a certain node.



The specification of such an event can e.g. be used in a rule like "in case that I receive a message from my bank with my account statement that contains a debit of more than 1000E then ...", where the occurrence of a generic event is restricted further wrt. its content. The receipt of the message is an event that can be detected by the communication service of a node, whereas the additional test must be checked on the application level.

Composite Events: Event Algebras. Event algebras, well-known from the Active Database area, serve for specifying *composite* events by defining *terms* formed by nested application of operators over *atomic* events. Each operator

has a semantics that specifies what the composite event means. Detection of a composite event means that its "final" atomic subevent is detected, e.g., as in [CKAK94]:

$$(1)\ (E_1 \nabla E_2)(t) \ :\Leftrightarrow\ E_1(t) \vee E_2(t)\ ,$$
$$(2)\ (E_1 \triangle E_2)(t) \ :\Leftrightarrow\ E_1(t) \wedge E_2(t)\ ,$$
$$(3)\ (E_1; E_2)(t) \ :\Leftrightarrow\ \exists t_1 \leq t : E_1(t_1) \wedge E_2(t).$$

Event algebras contain not only the aforementioned straightforward basic connectives, but also additional operators. A bunch of event algebras have been defined that provide also e.g. "negated events" in the style that "when E_1 happened, and then E_3 but not E_2 in between, then do something", "periodic" and "cumulative" events, e.g., in the SNOOP event algebra [CKAK94] of the "Sentinel" active database system. Some preliminary work on composite events in the Web is presented in [BKK04], but that only considers composition of events of modification of XML data.

Example 1 (Cumulative Event, [CKAK94]). A "cumulative aperiodic event"

$$A^*(E_1, E_2, E_3)(t) \ :\Leftrightarrow\ \exists t_1 \leq t : E_1(t_1) \wedge E_3(t)$$

occurs with E_3 and then requires the execution of a given set of actions corresponding to the occurrences of E_2 in the meantime. Thus, it is not a simple event, but more an active rule, stating a temporal implication of the form "if E_1 occurs, then for each occurrence of an instance of E_2, collect its parameters, and when E_3 occurs, report all collected parameters (in order to do something)".

A cumulative periodic event can be used for "after the end of a month, send an account statement with all entries of this month":

$E(Acct) :=$
$A^*(first\_of\_month(m), (debit(Acct,Am) \nabla deposit(Acct,Am)), first\_of\_month(m{+}1))$

where the event occurs with *first_of_next_month* and carries a *list* of the debit and deposit actions.

5.2 Ontology of the Event Component

With this ontology, the event component may consist of a combination of one ore more event algebras, using atomic events of one or more applications, and possibly atomic data-level events from several data models, and some generic parameterized events (see Figure 4).

An important matter here is that all components of an event specification can be associated with the appropriate components of the language using *identifiers*. This identification is provided for the XML Markup level by *namespaces* and their URIs, and for the RDF level directly by URIs (see also Section 6).

XML Markup for the Event Component. The eca:event elements contain elements according to an event algebra language (identified by its namespace), and embedded into this, eca:atomic-event elements are the "leaves" of the event language level. Inside of eca:atomic-event elements, the namespaces of the applications are used for the actual atomic event patterns. Whenever an atomic event

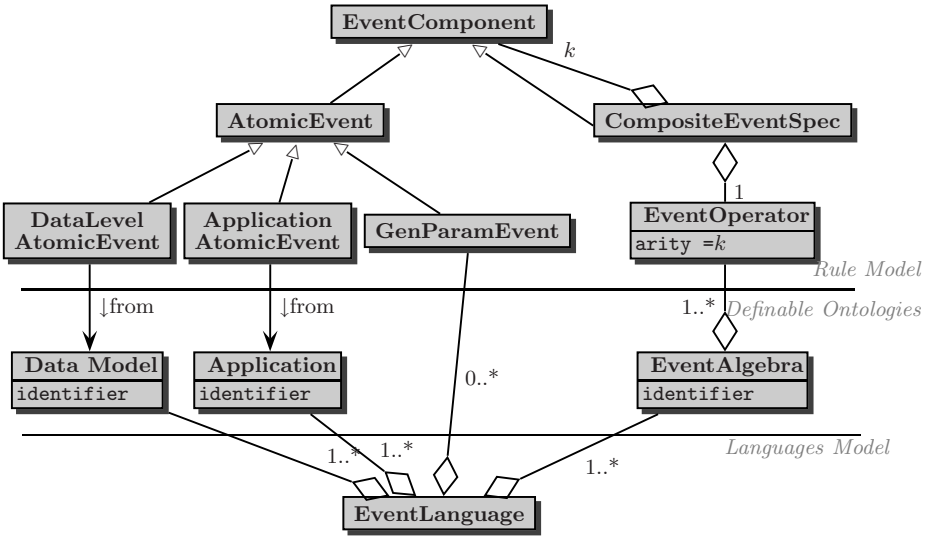


Fig. 4. Event Component Ontology

matches such a pattern, it (i.e., its XML or RDF representation) is bound to the temporary variable $event. eca:bind-variable elements inside eca:atomic-event elements allow for binding rule variables by using $event.

Example 2. Consider the event of Example 1. Atomic events are (i) temporal events that are assumed to be provided/signalled by some service, e.g. as <temporal:first-of-month month="5" year="2005" />, and (ii) events of the banking application, provided as e.g.,

<banking:deposit account="1234"><amount>200</amount></banking:deposit>.

```
<eca:rule>
  <eca:variable name="account" select="arguments[1]" />
  <eca:variable name="list" />
  <eca:variable name="month" />
  <eca:event xmlns:snoop="uri of the snoop event algebra"
             xmlns:banking="uri of the banking ontology"
             xmlns:temporal="uri of some web service">
    <snoop:cumulative-event cumulative-result="list" />
      <snoop:cumulative-start> <eca:atomic-event>
        <temporal:first-of-month>
          <eca:bind-variable name="month" select="$event/@month" />
        </temporal:first-of-month>
      </eca:atomic-event> </snoop:cumulative-start>
      <snoop:cumulative-collect> <snoop:disjunctive>
```

```
        <eca:atomic-event> <banking:debit account="$account" />
          <eca:bind-variable name="list" select="$event" />
        </eca:atomic-event>
        <eca:atomic-event> <banking:deposit account="$account" />
          <eca:bind-variable name="list" select="$event" />
        </eca:atomic-event>
      </snoop:disjunctive> </snoop:cumulative-collect>
      <snoop:cumulative-end> <eca:atomic-event>
        <temporal:first-of-month month="$month+1" />
      </eca:atomic-event> </snoop:cumulative-end>
    </snoop:cumulative-event>
  </eca:event>
    :
</eca:rule>
```

Note that the cumulative event defines the variable list to be cumulative, i.e., for each <eca:bind-variable name="list" select="$event" />, the event (as XML element) is *appended*.

5.3 Query, Test, and Action Ontologies

The ontologies for the query, test, and action components follow a similar design.

Queries. Queries can be queries against individual nodes, or against "the Web". Here, existing languages like XPath, XQuery, or RDQL that are commonly supported can be used. Such languages that are based on a kind of basic expressions and algebraic operators use a classical tree markup. Since query languages are in general supported in the Web nodes themselves, there is in general no need for specific services; often, the query component is *opaque*.

Tests. Tests in general use boolean operators and quantifiers which are already covered in Markup Languages like, e.g., FOL-RuleML [BDG+] for formulas in first-order logic. Instead of first-order atoms, also "atoms" of other data models can be used, employing identifiers in the same way as for the event component. Since all relevant information is gathered in the event and query components, the test is evaluated locally.

Actions. The action component is similar to the event component: we distinguish atomic actions on the database level (updates expressed in DOM, XUpdate, XQuery+Updates, or in an RDF update language), generic actions (sending messages with some content), and execution of composite actions, even as *transactions on the Web*. Actions here also include intensional updates on the semantic level (that must be translated into actual updates at certain nodes). The actual processing of transactions and intensional updates is independent from this framework. Similar to the definition of composite events, composite actions and transactions can be defined e.g. in the style of CCS [Mil83], augmented with transactional commands.

6 Rules, Components, Languages and Processors as Resources

Rules on the semantic level, i.e., RDF or OWL, lift ECA functionality wrt. two (independent) aspects: first, the events, queries, and actions refer to the RDF/ontology level. An even higher level regards rules themselves as objects of the Semantic Web: rules are specified in RDF/OWL using the above rule ontology and event, query, test, and action subontologies.

Reuse: Rules and Rule Components as Resources. The above ontology directly leads to a resource-based approach: every rule, rule component, event, subevent etc. becomes a resource. Every rule is then interpreted as a network of RDF resources of the contributing ontologies (ECA, event algebras, applications etc.). By this, e.g. collections of (sub)events as well as complete (application-specific) rule bases can be designed, published by associating them with a URI, and reused. Figure 5 shows the rule and the event component given in Example 2, combining two application-independent language ontologies:

- the ECA ontology (gray, doublelined), and the SNOOP ontology of the event algebra (incorporating the semantics of the SNOOP operators; gray),

and two application-dependent ontologies:

- the banking application-level ontology: there, the semantics of the atomic events defined in this ontology must be available (diagonally crosshatched).
- the temporal ontology: there, information about temporal events is available (crosshatched).

6.1 Information Behind the External Resources

The external resources (SNOOP, banking, and temporal domain) must be associated with further resources of the respective ontologies that are used when actually processing rules:

- Sublanguage ontologies (here, SNOOP): URI or a service for processing the language, e.g., a Web Service where the composite event specification can be registered, and that, when informed about relevant events, runs the detection and informs the client about success (transferring the result parameters). (Similar for languages for composite actions.)
- Temporal domain: URI of a service that provides the relevant atomic events.
- Application ontologies (here, banking domain): in most cases, the "client" knows which "server" (e.g., the bank where the account is located) provides the relevant atomic events. For other ontologies such as stocks or travel, the resource that is "responsible" for the ontology could also provide notification services for atomic events. In either case, derived events (that can locally use another event algebra) have to be defined there (since their definition conceptually also belongs to the ontology, this is not surprising). In the same way, a service for execution of atomic actions and the definition of composite actions (using any action language) can be provided.

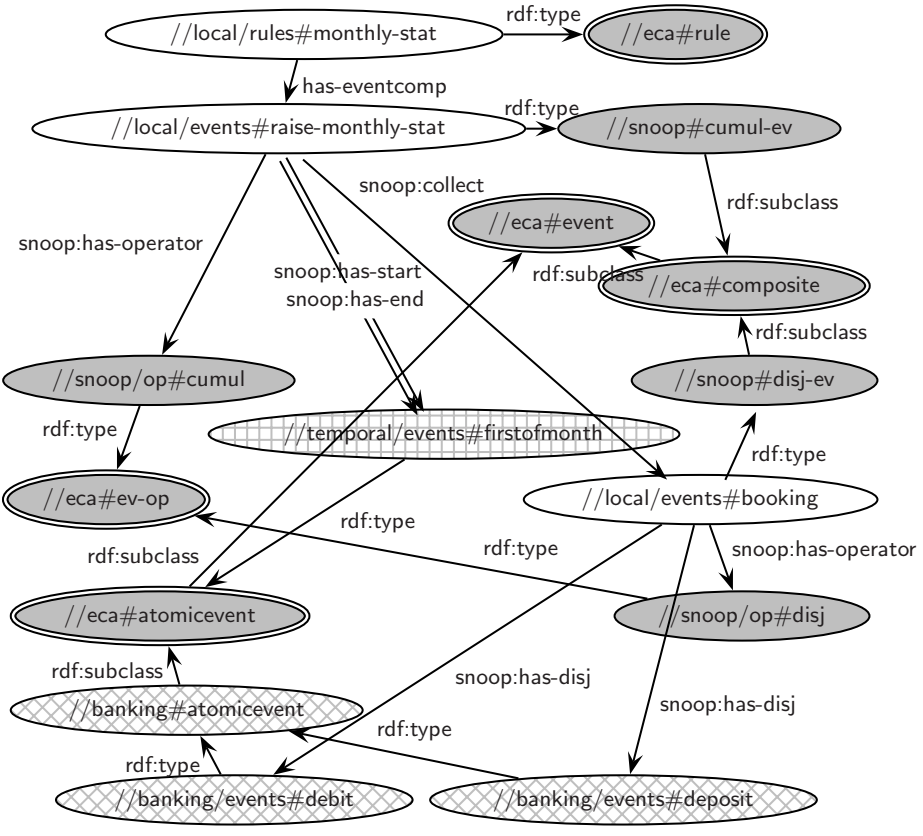


Fig. 5. Example Rule and Event Component as Resources

A Modified Rule Using a Derived Event. The banking ontology could define a derived booking event as the disjunction of debit and deposit. The rule could then directly use this derived event. In the ontology diagram, the only difference would be that the booking node is //banking/events#booking and appears diagonally crosshatched (and its semantical information must be kept at the banking resource – but in this case it can also be used in specifications that do not know the SNOOP language).

6.2 Architecture and Processing: Cooperation Between Resources

Rules can be evaluated locally at the nodes where they are stored, or they can be registered at a *rule evaluation service*. The rule evaluation engine manages the actual handling of rules based on the language URI references. As described above, every subconcept (i.e., events, queries, tests, and actions) carries the information of the actual language it uses in its xmlns:*namespace* URI attribute (note that this even allows for nested use of operations of *different* event algebras). Assume the case where the language processors are available at these URIs

as a Web Services. For event detection (and analogously, execution of composite actions), at least two resources (or services) must cooperate: Event detection splits into the *event algebra* part (that is detected algorithmically by a resource representing a *language* ontology, e.g., SNOOP) and the atomic events of the application ontology. Thus, the algebra processor must be notified about the atomic events. This can be done in several ways:

Straightforward: The "straightforward" way is that the client C organizes the communication between the event generator(s) and the event algebra processor (see Figure 6): C registers rules to be "supervised" at a rule execution service R. For handling the event component, R reads the language URI of the event component, and registers the event component at the appropriate event detection service S (note that a rule service that evaluates rules with events in different languages can employ several event detection mechanisms).

During runtime, the client C forwards all received events to R, that in turn forwards them to all event detection engines where it has registered event specifications for C, amongst them, S. S is "application-unaware" and just implements the semantics of the event combinators for the incoming, non-interpreted events. In case that a (composite) event is eventually detected by S, it is signalled to-

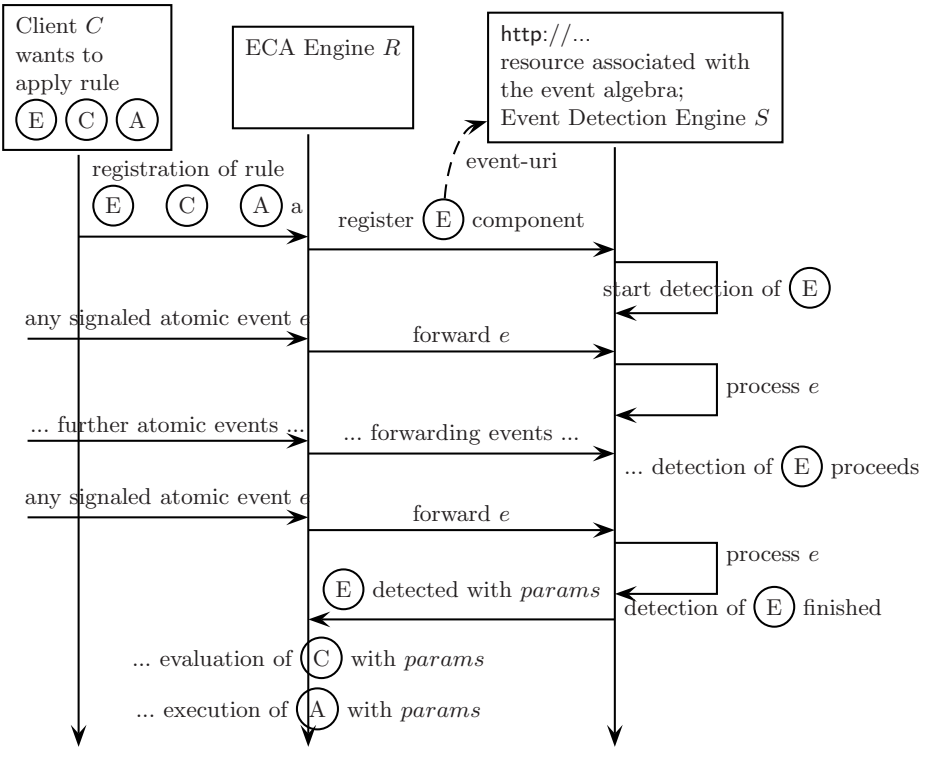


Fig. 6. Straightforward Communication (UML-style sequence diagram, temporal axis downwards)

gether with its result parameters to R. R takes the variables, and evaluates the query&test (analogously, based on the respective languages), and finally executes the action (or submits the execution order to a suitable service).

(Note that this strategy can be extended towards "selecting" and brokering events according to their namespace in a similar way to the architectures described below.)

Application-centered: The client submits its composite event specification to a service that is aware of all relevant events in the application domain. This service then employs an appropriate event detection service by registering the event specification, and informing it about the atomic events (e.g., "@bank: please trace the following composite event in language L on my account" (and employ a suitable event detection service for L)).

Language-centered: When a rule or an event specification is submitted for registration, this has to be accompanied by information on which resource(s) provide the atomic events (e.g., "@snoop: my bank is at *uri*, please supervise my account and tell me if a composite event ev occurs), or the detection service even has to find appropriate event sources (by the namespaces of the atomic events). The detection service then contacts them directly. This proceeding is e.g. appropriate for booking travels where the client is in general not aware of all relevant events (e.g., "@snoop: you know better than me who is well-informed about events relevant for traveling, please detect the event ev_{travel} for me"), as illustrated in Figure 7: A client registers a rule (in the travel domain) at R (Step 1.1). R again submits the event component to the appropriate event detection service S ((1.2), here: snoop). Snoop looks at the namespaces of the atomic events and sees that the travel ontology is relevant. The snoop service contacts a travel event broker (1.3) who keeps it informed (2.2) about atomic events (e.g.,

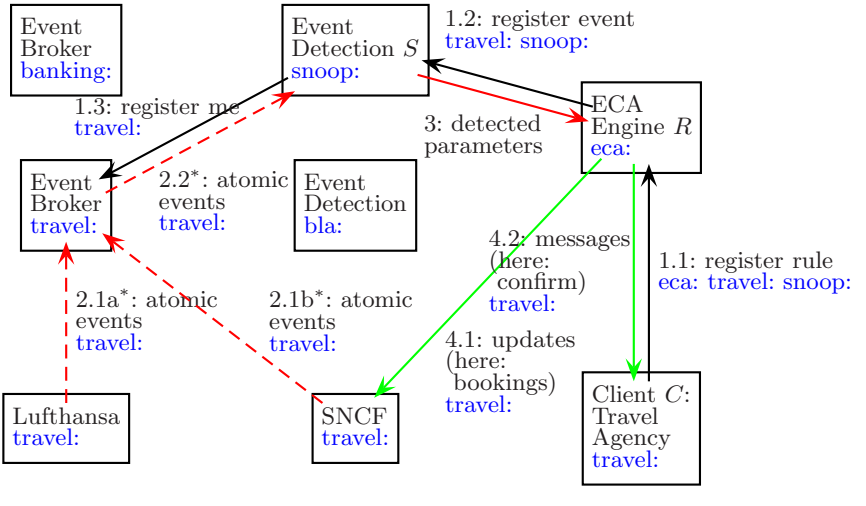


Fig. 7. Language-Centered Communication

happening at Lufthansa (2.1a) and SNCF (2.1b)). Only after detection of the registered composite event, S submits the result to R (3) that then evaluates the Q&C component, and probably executes some actions (4.1, 4.2).

7 Conclusion

We have presented a modular ontology-based framework for ECA rules. The ontology does not only *describe* the domain, but by including the processing resources on the Web also provides the infrastructure for actual implementation of the framework. Different alternatives allow for service-oriented strategies. More detailed aspects are currently investigated, and an implementation has been started, stepwise extending from XML-level ECA rules and services to the above framework. The modularization of the approach allows for dealing with many research issues separately.

Acknowledgement. This research has been funded by the European Commission within the 6th Framework Programme project REWERSE, number 506779 (cf. `http://rewerse.net`).

References

[ABB+05] J. J. Alferes, M. Berndtsson, F. Bry, M. Eckert, W. May, P. L. Pătrânjan, and M. Schröder. Use Cases in Evolution and Reactivity. Technical Report I5-D2, REWERSE EU FP6 NoE, 2005. Available at `http://www.rewerse.net`.

[BBCC02] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Intl. Conference on Data Engineering (ICDE)*, pp. 403–418, San Jose, California, 2002.

[BCP01] A. Bonifati, S. Ceri, and S. Paraboschi. Pushing Reactive Services to XML Repositories Using Active Rules. In *Int. WWW Conference*, 2001.

[BDG+] H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S. Tabet, and G. Wagner. FOL RuleML: The First-Order Logic Web Language. `http://www.ruleml.org/fol/`.

[BKK04] M. Bernauer, G. Kappel, and G. Kramler. Composite Events for XML. In *Int. WWW Conference*, 2004.

[BP05] F. Bry and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. *20th ACM Symp. Applied Computing*. 2005.

[BPW02] J. Bailey, A. Poulovassilis, and P. T. Wood. An Event-Condition-Action Language for XML. In *Int. WWW Conference*, 2002.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proceedings of the 20th VLDB*, pp. 606–617, 1994.

[Mil83] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, pp. 267–310, 1983.

[PPW03] G. Papamarkos, A. Poulovassilis, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In *Workshop on Semantic Web and Databases (SWDB'03)*, 2003.

[PPW04] G. Papamarkos, A. Poulovassilis, and P. T. Wood. RDFTL: An Event-Condition-Action Rule Languages for RDF. In *Hellenic Data Management Symposium (HDMS'04)*, 2004.

Automatic Web Service Composition Based on Graph Network Analysis Metrics

John Gekas and Maria Fasli

University of Essex, Department of Computer Science,
Wivenhoe Park, Colchester CO4 3SQ, UK
{jgekas, mfasli}@essex.ac.uk

Abstract. The web services paradigm has enabled an increasing number of providers to deploy and host autonomic and remotely accessible services. However, the true potential of such a distributed infrastructure can only be reached when such autonomic services can be combined together as parts of a *workflow*, in order to collectively achieve combined functionality. In this paper we present our work in the area of automatic workflow composition among web services with semantically described functionality capabilities. For that purpose, we are using a set of heuristics derived from the *connectivity structure* of the service repository in order to effectively guide the composition process. The methodologies described in this paper have been inspired by research in areas such as citation analysis and bibliometrics. In addition, we present comparative experimentation results in order to evaluate the presented techniques.

1 Introduction

The web services programming and deployment paradigm has received a lot of attention recently. Many authors have argued that this will be the dominating paradigm on the web in the years to come [CASATI *et al*., 2001], thus changing its structure from a web of information and manually-accessible applications, to that of a web of automatic services. Although web services can be considered as "*…reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols*" [RICHARDS *et al*, 2003], the true potential of a Service-Oriented infrastructure can only be realised by integrating individual web services as part of a workflow [BERNERS-LEE, 2001]. Many scenarios and architectures based on integrated web service workflows have been proposed already, such as Virtual Organisations [KHOSHAFIAN, 2002], B2B (Business to Business) systems [HOGG *et al*., 2004] and SCM (Supply Chain Management) scenarios [MIN and BJORNSSON, 2004].

However, the day when effective and ready-to-use systems performing automatic web service integration can be widely used is yet to come. One main problem is the lack of a standardised, uniformly accepted format of describing a service's functionality semantically, so that this information can be used towards a composition request. To this direction, the *semantic web initiative* [SEMANTIC WEB, 2001] effort addresses the problem of providing web-based resources (e.g. static information web sites, web-based applications) with semantic meaning, so that their content and functionality can be extracted and used by computer systems. A number of formats have

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1571 – 1587, 2005.

been suggested, such as RDF and RDF-S, DAML-S and its successor OWL-S, some of which gaining growing popularity [ANOKOLEKAR *et al*., 2002].

In this paper, we present our approach towards automatic service composition. We assume that workflow generation takes place within a repository-style environment (the *registry* or *network*) upon request. The work presented here is twofold: first, we present the framework we have developed in order to experiment with service composition methodologies. The second part refers to the particular methodologies we have tested in order to address the problem of dynamic service integration, at a typical service composition scenario. The two aspects of the work presented in this paper should not be confused –the service composition problem domain can be divided in a number of different use cases, each of them with their own requirements and potential solution approaches. The approach proposed here refers only to one such use case, whereas different use cases can be examined and tested within our framework.

Our service composition approach focuses on viewing the service composition registry as a *hyperlinked graph network* of scalable size, and dynamically analyzing its structure in order to derive useful heuristics to guide the composition process. We believe that the structure of the service network (or, in other words, the *ecology* of the search space) can provide powerful and useful information, which can then be used in order to guide the composition process effectively. For this purpose, we present and assess a set of methodologies/heuristics that attempt to utilize this notion. The heuristics described in this paper fall into the following 3 categories: *importance, relative importance* and *relatedness* among the members of the service repository. The choice of the particular methodologies has been deliberate: global importance, relative importance and node relatedness are the main methodologies used in graph-based citation analysis and bibliometrics [ITO *et al*., 2004]. The rest of this paper is organised as follows: Section 2 outlines current work in the area. Section 3 presents the architectural framework, while in Section 4 we discuss the methodologies used in our research. In Section 5 we present some experimental results showing the comparative performance of the methodologies tested, and we conclude this paper by pointing out some interesting conclusions and further work directions.

2 Literature Review

A number of approaches addressing the problem of automatic web service integration have been presented recently, both academic and industrial. [PEER, 2003] suggests the use of existing AI planning systems, even though he does not present a solution prototype. [WU *et al*., 2003] are also geared toward that direction, using SHOP2, an existing HTN (Hierarchical Task Networks) AI planning system. However, classical planning systems are not fully applicable in the service composition problem domain. The workflow of a business plan may require control structures (e.g. branch and loop structures), that classical AI planning may not be able to generate. Additionally, data flow control and performance under very big and scalable search spaces is also an issue.

[TUT and EDMOND, 2002] on the other hand, propose the use of patterns in service composition. According to this approach, each composition request is matched against an appropriate abstract *composition template*, which is mapped to existing services at the implementation level. Similar is also the approach followed by [LIMTHANMAPHON and ZHANG, 2003], where successful past composition requests are stored, so that similar composition structures can be followed in future re-

quests as well. Even though these approaches can be effective in many cases, they suffer from the restrictions posed by the patterns followed at composition time – in other words, effective solutions that do not follow the applied patterns are not likely to be found. [MONTEBELLO and ABELA, 2002] follow a different approach: composition is performed with the use of a rule-based expert system, JTP (Java Theorem Prover). This is not the only approach where the use of expert systems is proposed for use with semantic web services: [KOPENA and REGLI, 2003] use Jess (Java Expert System Shell) in order to reason about web services capabilities, and the composition approach followed by [PONNEKANTI and FOX, 2002] is also rule-based.

Finally, we should also note that there are some approaches that view service composition as a semi-automatic and interactive process, where the system makes corrections and suggestions based on partially composed workflows. The approaches proposed by [SIRIN *et al*., 2004], [AGARWAL *et al*., 2004] and [JIHIE *et* al. 2004] follow this logic.

3 Experimental Framework

Our framework consists of a repository-style registry, where web service semantic descriptions are stored. We have to note here, that no actual web services are hosted at the registry; however, semantic descriptions of already existing services are stored; this allows web services from various providers to be integrated within our system. We also have to add that our research has been focused on two particular kinds of web services: informational services (such as a financial news service) and E-Commerce services (e.g. Amazon transaction service). This choice was deliberate: Web-based statistical observations led us to believe that these are the dominating types of services populating today's web. The overall architecture of our system is illustrated in Fig.1:

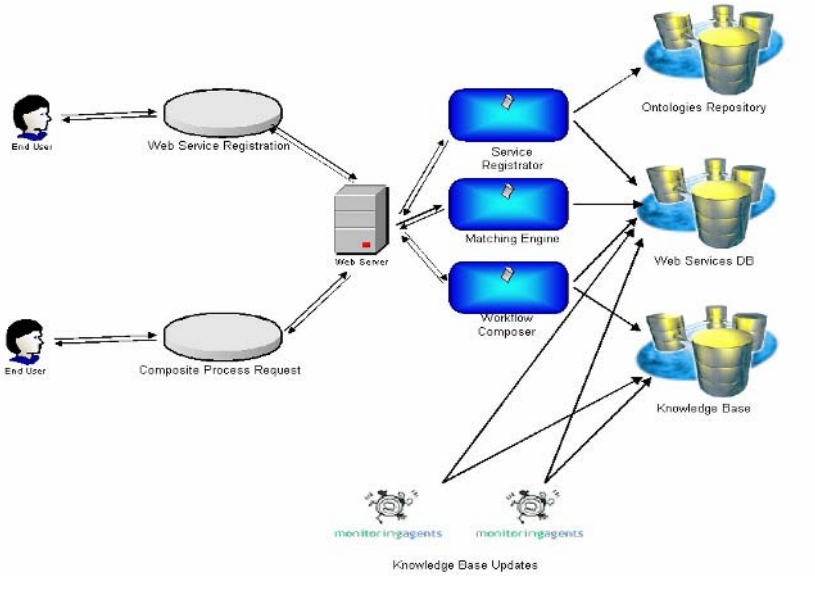


Fig. 1. Service Composition Framework Architecture

The separate parts of our framework are explained in the following sections:

3.1 Web Service Registration System and Semantic Description Storage

Web services are "registered" with the system through a web-based interface, where the service's host provides some basic information about the service's operations he/she wishes to register. At the current version of our system, the service's semantic description is semi-automatically generated using the DAML-S format, based on the provided information. However, since the more recent OWL-S format provides similar functionality to DAML-S with higher expressiveness, a migration to OWL-S is planned to take place in the immediate future. Some important information that needs to be provided by the service's provider is:

- Web Service Category: which general category the service belongs to, i.e. Information Service, E-Commerce Service, Travel Booking etc.
- Operational Domain: Functionality domain for a particular operation – Information Query, Transaction Execution, Authentication etc.
- Semantic Data Types (SDT's) domain and range: Each semantic data type used by a service is described in terms of *domain* and *range*. Domain refers to the nature of the data type, such as e.g. ProductName, whereas the range refers to its intended use, e.g. UserName, SearchQuery, TransactionConfirmation etc. A similar semantic categorisation between web services and data types is also followed by [HESS and KUSMERICK, 2005].

A service category (e.g. E-Commerce Service) may consist of operations belonging to different operation domains: for instance, an operation whose purpose is to carry out a commercial transaction belongs to the TransactionExecution domain, whereas an operation whose purpose is to query the price for a given product belongs to the InformationQuery domain. The service categories, operation domains and semantic data type descriptions are defined in the ontologies used by our system, described below.

3.2 Ontologies

The semantic descriptions generated for each registered service are based on the common ontologies used by the system, which serve as a common vocabulary used by the descriptions. Notions such as web service categories, operational domains, semantic data types etc are defined within the ontologies, with the appropriate mappings among them. For instance, the data type BookAuthor can be mapped to the data type PersonName; similarly, ProductProvider can be mapped to CompanyName. Exact matches is not the only option, of course – i.e. the data type BookISBN is a subtype of ProductCode. The ontologies used for our research are developed in-house using the DAML+OIL format. Problems like inter-operability between different ontologies or automatic semantic description generation without common ontological background are beyond the scope of this paper. [HESS and KUSMERICK, 2005] have done some work towards this direction.

3.3 Request Matching Engine

Before any attempt for workflow generation takes place, a given request is matched against the web services registered with our system, in order to check whether a service exists that can satisfy the request. Given a request, matching is performed at the Input/Output parameter level, and the Service Category/Operational Domain level, if provided by the request. However, *subtype matching* (e.g. some or all of a service's parameters are semantic subtypes of another's) is attempted as well, as described in [PAOLUCCI *et al*., 2002] and [KAWAMURA *et al*., 2003].

3.4 Knowledge Base

The Knowledge Base (KB) stores information about the registry's structure and connectivity figures among the registered web services. In particular, for every registered web service operation, the following information is stored in the KB:

- The number of service operations that point to it, as a percentage of the overall service category/domain.
- As above, the percentage of web service operations it points to.
- The amount of semantic parameter types the operation is linked to (i.e. its output parameters), again as a percentage of all the semantic parameter types utilized by the category/domain.

The above figures are calculated and stored upon registration, as a triggered event. Alternatively, the above information could be generated and updated by crawling the search space offline – such an approach however could be computationally demanding, depending on the size of the search space. We also have to note that PR figures can be associated with semantic data types themselves, showing the percentage of data types a particular data type is linked to.

Borrowing the terminology presented in [PAGE and BRIN, 1998], the above figures consist the *local PageRank* value for a given operation, which gives us an idea of how "important" an operation is within its Category or Domain – the higher the local PageRank value, the more popular a particular operation is considered to be. This way, we know which service categories are better inter-connected than others and which operations linked together; this information can prove valuable while guiding the search process at composition time.

We also have to note that PageRank estimation takes place within a given *scope*, which can either be a Service Category, an Operational Domain or the overall search space. If the PageRank figure is calculated as a percentage out of the whole of web services belonging to a particular Category (such as InformationService), then its scope is this Service Category. Alternatively, PageRank values could be calculated as a percentage of an Operational Domain (such as InformationQuery or TransactionExecution), or out of the overall search space.

The PageRank algorithm has received some criticism as well, the most important argument being that PageRank values are not related to specific queries/requests, thus they are not always an accurate measure of "importance" [BHARAT and MIHAILA, 2000].

3.5 Workflow Composer

The workflow composer is a recursive depth-first algorithm, which starts from the initial state and tries to reach the goal state following the shortest route possible. At the beginning of the composition process, the algorithm attempts to determine the most "promising" service categories / operation domains in which the search should be focused. This is done in the following way: since the provided input parameters are specified in the request, the algorithm determines which service category or domain is most promising by examining the local PR values for the given input parameters at every category / domain. For instance, it is reasonable to expect that the SDT WeatherTemperature will be used at a considerably higher rate within the WeatherService category, than say, E-CommerceService. At every stage of the search process, the algorithm decides which node to follow next based on which one is more "important", or "relative" to the goal state (depending on the composition heuristic in use) – the other possible nodes are placed at a *"Priority Queue"* based on their own importance (or relativity) figures. In this sense, our algorithm works as a Markov Decision Process, where the following step depends on which node looks more "promising" to lead to the goal state faster. Using an iterative PageRank for higher depths would probably give us a better idea about how important a web service is, but cannot be calculated at composition time since it is computationally demanding, and we have been trying to keep offline processing to a minimum.

Following is the workflow composer algorithm, shown as pseudocode:

```
Method ExpandPartialWorkflow (workflow WF)
{
    Get WF PriorityQueue PQ

    For each p[PR] in PQ
    {
        Get operations (O1,…,On) that match p[PR]

        For each Oi in (O1,…,On)
        {
            Add Oi to WF
            PQ = new PQ
            Get all output parameters (p1,…,pn) for Oi


            For each p in (p1,…,pn)
            {
                Get PR
                Add p[PR] to PQ
            }

            Sort PQ in descending order
            ExpandPartialWorkflow (WF)
        }
    }
}
```

Fig. 2. Workflow Composer Algorithm

WF stands for workflow and PQ for Priority Queue. p [PR] refers to a particular semantic data type p, with local PageRank value equal to PR. The above process continues until a solution is found, and as long as the maximum depth has not been reached. We assume that at least one web service operation whose input semantic data

types match the request has been found already, and has been added as the first node to a previously empty workflow.

4 Approach Description

Our approach considers web service integration as a graph search problem, where the search space consists of all the potential web service operations that can be part of a workflow. The search space (the service registry) is viewed as a hyperlinked graph network with no size restrictions. Thus, we are focusing on performing effective search through a search space of scalable size, by attempting to minimize the path followed from the initial state to the goal state. In order to do this, we are using heuristics derived from the repository of semantic descriptions, regarding the *connectivity structure* of the repository and how "tightly" various types of web services are linked together (the *ecology* of the search space). These heuristics are used in order to generate the "Priority Queues" used by the composition algorithm. Connectivity structure figures are estimated automatically by the system and updated as new services are added. In this sense, we have used a similar methodology to the one used by major search engines in order to rank individual web resources and estimate the connectivity graph of sub-parts of the web. In this section, we present a set of heuristic methodologies we have tested in order to guide the service composition process:

Global Importance: Various approaches [PAGE and BRIN, 1998], [KLEINBERG, 1997] describe ways to rank individual web resources based on their popularity. This way, a major-scale search engine can reason on how "important" a web resource is, in terms of how many other resources point to it and how many resources it points to itself. Even though most approaches follow a similar pattern, we could say there are two broad categories: the first estimates the overall "importance" of a web resource, whereas the second one estimates the importance of a web resource relatively to a particular area/request (*relative importance*).

We believe that some aspects of search engines' ranking methodology can be applied to our problem domain: [PAGE and BRIN, 1998] describe their algorithm, titled PageRank (henceforth PR) as follows: "*…given a web page A and a web page B, the PageRank value of A is the probability that a 'random web surfer' will visit page B just by following links initiated from A*". Following a similar train of thought, we can also define the probability that a certain web service B can be reached at a certain depth, given that another web service A has been accessed. Web services can also be considered as web resources that point to other web resources. Assume a web service A that provides a semantic data type (SDT) X as an output parameter, and a web service B that accepts data type X as its input parameter. In this case, we assume that there is a *forward link* between A and B. Likewise, we can define backward links as well.

Furthermore, we could define similar connectivity measures for semantic data types themselves: a semantic data type that is being used by X% of the search space as an input parameter, is considered to have a forward PR equal to X%, whereas a data type that is being used by Y% of the operations belonging to our search space as output parameter, is thought to have a backward PR equal to Y%.

Relative Importance: Relative Importance is similar to the metrics of importance mentioned above, but normalized towards a particular area of request. Approaches

such as [KLEINBERG, 1997] and [HAVELIWALA, 2002] follow the notion of relative importance in order to effectively rank web resources. This normalization can either take place offline (sometimes referred to as *PageRank with Priors* – PR figures estimated towards some predefined areas of interest), or at runtime. In our experiments, the overall importance PR figures are adjusted towards each particular request at runtime: the request's "area of interest" is dynamically determined by the goal SDT's of the request (e.g. a request that contains the goal SDT's *BookName* and *BookAuthor* is directly related to the notion *Book* described in our ontologies), and the overall PR values are adjusted so that they show the percentage of web service operations (or SDT's) that point towards the particular request area.

SDT Relatedness: Relatedness refers to the process of identifying to what degree two web resources are related to each other. Since the majority of web-based resources have textual form, relatedness analysis is closely related to text analysis and matching methodologies (such as *q-grams*, *Neumann kernels* and so on). However, web services cannot be considered as textual resources, but rather as functional ones, i.e. exposing functionality features semantically. Web service descriptions, although being textually understandable, are mainly destined for machine-based processing. In this context, when we talk about relatedness between two web services, we refer to the similarity of the data flows generated by their services, estimated as the *semantic distance* between the corresponding data types. The idea behind this technique is that the number of informational web services connecting two different pieces of information (sdt's) will be greater, when the pieces of information are more closely (semantically) related with each other. For instance, it is logical to expect that there will be more services connecting the SDT's BookName and BookAuthor, rather than e.g. BookName and AirportLocation. For the estimation of semantic distance we have chosen a rather simple metric based on graph based distance between attribute values, as described in [FOO *et al*., 1992]. However, alternative (and more complicated) models, as the ones presented in [RODDICK *et al*., 2003] can be considered. In more detail, the semantic distance between two given data attributes within a common ontology is estimated as the minimum number of edges that need to be traversed when the ontology is viewed as a graph. This can be described as following:

Let our ontological data model be represented by graph G, with root node R. Also, let's assume that we want to define the semantic distance between semantic data attributes sdt_i and sdt_j. Then, the sets $A = \{e_{i1}, e_{i2}, ..., e_{im}\}$ and $B = \{e_{j1}, e_{j2}, ..., e_{jn}\}$ (with set sizes denoted by $|A|$ and $|B|$) contain the edges that need to be traversed in order to get from R to sdt_i and sdt_j, respectively. The semantic distance between sdt_i and sdt_j can be given by the size of the set $(A/B)\bigcup B$, thus being $|(A/B)\bigcup B|$.

The problem of semantically matching attribute values between different ontological backgrounds is not being addressed by our research at this point.

Workflow Request Evaluation: Given a specific composition request, we can deduce whether it can lead to solutions, and if yes, how many. Such information can be deduced before any attempt for workflow composition takes place and can be ex-

tremely useful, since it allows the composition process to be activated only for those requests that can lead to solutions and only up to a certain number of solutions, thus saving computational power and processing time. This process is briefly explained in the following paragraph:

Let's assume matrix A, whose rows represent the web services present in our registry, and its columns represent all semantic data types described in our ontologies. Then, each element in the matrix would show the number of times the corresponding web service (denoted by the row index) utilizes each particular SDT. By definition, each element in the matrix would either be 0 or 1, since it is logical to assume that a particular web service would not provide the same SDT more than once.

$$A = \begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix}$$

Matrix A is represented as orthogonal, but this is not necessary. In fact, it is highly unlikely that the number of web services supported by a registry will be the same as the semantic data attributes described by the registry's ontologies. This is a *web service – to – semantic data type* (ws-sdt) matrix, whose elements can be either 0 or 1.

The transpose of A, A^T, calculated by substituting A's columns with its rows and vice versa, shows exactly the opposite relationship: how many times (0 or 1, as mentioned above) a particular datatype is utilized by each web service in the registry. In this sense, it can be considered as semantic data type – to –web service matrix (sdt-ws).

By multiplying A with A^T we get matrix AA^T, which can be considered as the ws-ws matrix. Similarly, by multiplying A^T with A we get $A^T A$, which is the sdt-sdt matrix. AA^T shows how many "links" exist between two particular web services, whereas $A^T A$ shows how many links exist between two given semantic data types. We have to note here, that both AA^T and $A^T A$ are diagonal (i.e. the number of rows equals the number of columns and $a_{ii} = 0$). Thus, AA^T and $A^T A$ have the following form:

$$\{AA^T, A^T A\} = \begin{matrix} 0 & a_{12} & a_{13} & a_{14} \\ a_{21} & 0 & a_{23} & a_{24} \\ a_{31} & a_{32} & 0 & a_{34} \\ a_{41} & a_{42} & a_{43} & 0 \end{matrix}$$

Matrix $(AA^T)^2$, (AA^T multiplied by itself) shows as how many links exist between two given web services, at depth = 2. Similarly, $(AA^T)^n$ shows as how many links exist between two given web services at depth = n. Following a similar train of thought, if we substitute matrix AA^T (ws-ws) with $A^T A$ (sdt-sdt), by raising at a power of n we can see how many links (*paths*) exist between two given SDT's at the

given depth = n. Since a workflow composition request is primarily represented as initial and goal states of the data provided by the workflow, the above process can show us how many solutions a given request can lead to (0, if it has no solutions).

We also have to note that our work concentrates on a typical composition scenario, where both provided input parameter types and the required output parameter types are explicitly declared in the request. We also assume that a solution can be given by an execution combination of a number of web service operations. More business-oriented scenarios (e.g. Supply Chain Management), along with issues like workflow control constructs and constraints are addressed by our research, but description of the corresponding mechanisms is beyond the scope of this paper.

5 Experimental Work

We have performed some initial experiments in order to evaluate our approach. The experiments operate on the framework we described above, and their main purpose is to examine the accuracy of the composition algorithm, using the search heuristics described earlier. The experiments presented here have been performed within the InformationQuery operation domain, with a total of 702 requests and 254 solutions.

5.1 Experiment Setting

The experimental settings are given in the following table:

Table 1. Experiment Setting

Experiment Setting	Value
Search Space Size	110 Operations
Algorithm	Depth-First, Recursive
Heuristics Used	a. Local PR per Operation (importance) b. Request-Related PR (relative importance) c. SDT Semantic Distance (relatedness)
Heuristics Direction	Forward

The experiment settings can be explained as follows: the first 2 lines (“Search Space Size” and “Algorithm”) are self-explanatory. Line 3 (“Heuristics Used”) shows the heuristics that have been used in order to guide the composition process for each experiment. The first option is the local PageRank values for each web service, which corresponds with the “importance” of each web service within our search space. The second option refers to the local PR values, normalized to the topic of each particular composition request (“relative importance”), and the third option refers to the semantic distance of the data flows generated during the composition process (“relatedness”). Finally, the last line (labeled “Heuristics Direction”) refers to the direction of the metrics described above, i.e. forward or backward. The same experiments with the Heuristic Direction set to backward have not been performed yet.

In addition, in the following table we show the semantic data types used in our experiments, along with the alias's used in the diagrams shown in the next section:

Table 2. Semantic Data Types used in our experiments

Alias	Semantic Data Type Name
sdt01	NewsType
sdt02	AirportCode
sdt03	AirportLocation
sdt04	BookISBN
sdt05	BookName
sdt06	CompanyName
sdt07	CompanyStockQuoteTicker
sdt08	LocationTown
sdt09	LocationZipCode
sdt10	ProductCode
sdt11	ProductName

5.2 Experiment Results

In order to evaluate the performance of the tested methodologies, we have followed the following process: for all the SDT's that are present in the InformationQuery domain, we have generated every single combination among them, on a 1-to-1 basis. For each of these combinations, we have generated an appropriate request using the corresponding SDT's as the request's input and output data types (in other words, using the SDT's as the request's initial and goal state, in terms of data attributes). Consequently, we have run our algorithm against all the requests. This process has been followed for all the heuristic methodologies described earlier, so that their comparative performance can be presented. The performance metric used is to compare the number of "nodes" (web service operations) that were used as part of the solutions, with the total number of nodes expanded by our search algorithm – this gives us an idea of how well-guided the search algorithm was during the composition process. A high ratio of solution nodes to total expanded nodes means that the composer was successfully guided, so the higher ratios an approach generates, the more successful is considered to be. In addition, we have calculated PageRank figures for all the semantic data types that take part in the experiment, at depth = 3, which gives us an idea of how "well-linked" the experimental search space is at increasing depths. The above figures (ratio between solution and expanded nodes) are also compared against the distribution of the PR values of the semantic data types, so as to spot any potential trends about whether the performance of a particular methodology is related to the link structure of the search space at increasing depths. We have to note that the mentioned PR distribution at depth = 3 is not in any way used during the composition process, but only serves for evaluation purposes. As mentioned above, three series of experiments were performed – one for each composition methodology. We first present the performance of each approach separately (in comparison to the distribution of the PR growth rate of our search space) and then comparatively.

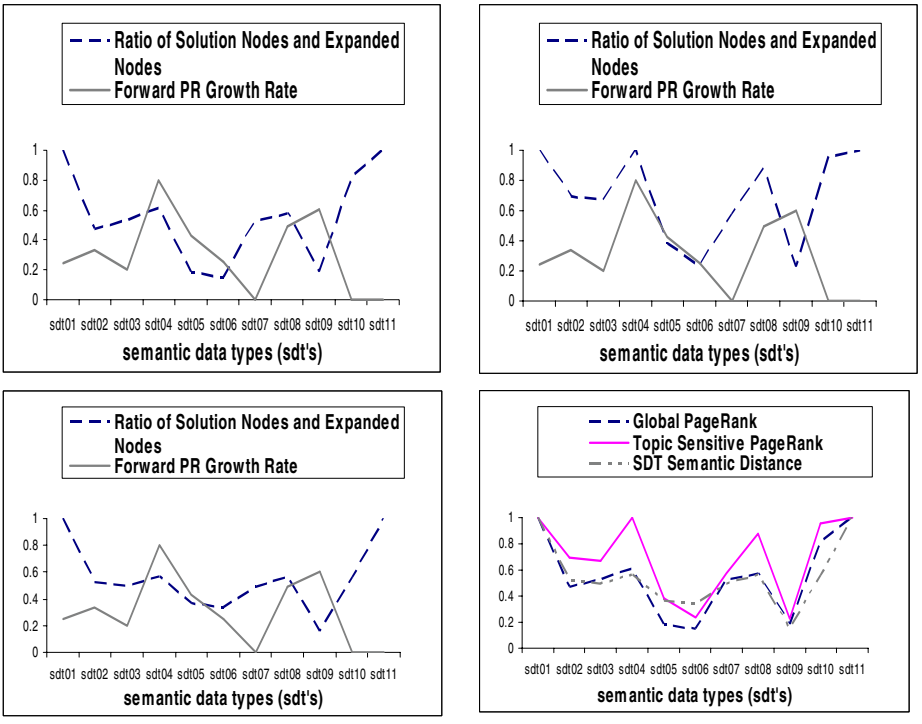


Fig. 3. Comparison of Average ration of Expanded Nodes and Solution Nodes per Semantic Data Type with Forward PR Growth Rate, using: (a) Global PageRank, (b) Topic-Sensitive PageRank, (c) SDT Semantic Distance, and (d) comparative figures

Fig.3 shows the comparison of Solution Nodes and total Expanded Nodes for a successful request, in relation to the Forward PR distribution of the search space, at depth = 3. The first three diagrams show the performance of the three heuristic methodologies described earlier: (a) corresponds to importance – PageRank, (b) to relative importance – Topic-Sensitive PageRank, and (c) to relatedness – SDT Semantic Distance. On the X axis, we can see the semantic data types (SDT's) present at the experiment. These SDT's can be considered as the starting state of a successful composition request (in other words, the provided Input Parameters for a given request). In all diagrams, the solid line shows the growth rate of the forward PR for the given SDT's. The dashed line shows the ratio between the solution nodes and the overall expanded nodes for the requests that had the given SDT's as starting state. Both lines occupy values between 0 and 1 inclusive, which are shown on the Y axis.

Even though the diagrams represent different behaviours, we can see the general relation between the above figures: both lines (ratio of solution and expanded nodes and forward PR growth rate) follow similar directions, which leads us to believe that the heuristic methodologies presented in this paper can work more effectively when the PR growth rate of the SDT's present at our search space becomes higher at in-

creasing depths – in other words, when the search space is better-linked at increasing depths.

Even though the lines in all three diagrams follow similar trends, we can see that the performance of the approaches differ significantly. Keeping in mind that higher ratios between solution and expanded nodes mean a better-guided and more effective composition process, we can see that diagram (b) is the one showing the best performance. This means that the Topic-Sensitive PageRank metric as the composer's search heuristic led to better results.

5.3 Discussion

The diagrams above show that the relative importance metric (topic-sensitive PageRank) generates more effective results in our context, during the workflow composition process. We can maybe justify this behaviour by considering the nature of the heuristic metrics that were used for our experiments: PageRank is a metric of global importance within a search domain, showing how big part of the domain each node is linked to. Keeping this in mind, we can expect that the global nature of PageRank (independent to each particular request) can sometimes "attract" the composition algorithm towards "powerful" web services within our repository, without them being directly related to the particular request in question. An example of such behaviour can be noted in Fig.3 (a) by looking at the corresponding figures for the LocationZipCode SDT (sdt09); it has a relatively high PR growth rate (=0.58 at depth = 3) but a rather low ratio of solution and expanded nodes (=0.19). Topic-sensitive PageRank on the other hand, seems to be doing quite well overcoming this side-effect.

The (rather low) performance of the SDT semantic distance heuristic technique can also be justified: as it has already been mentioned, the semantic distance heuristic is based on the hypothesis that the number of informational web services connecting two different SDT's will be higher, when the SDT's are closer-related to each other. This notion implies that in order for this heuristic to perform effectively, the distribution of the semantic distance lengths between all SDT pairs within our ontologies has to be similar with the degree distribution of the same SDT pairs within our search space. Clearly, this does not have to always be the case.

We have also seen that the PR growth rate of a given data type is related to the performance of the search algorithm used. The ratio of solution and expanded nodes gives us an idea of how well-guided the search algorithm is during the composition process, and we saw that the ratio is higher when the PR growth rate of the starting state data type is higher. In other words, this shows that the better-linked the operation domain is at increasing depths, the more effective our composition heuristics perform. Thus, we can assume that our composition approach can lead to more effective results when they are applied to rather "condense" search spaces. Beyond this elementary intuition, we believe that the degree distribution of the service repository (search domain) is a major issue, as it can greatly affect the performance of the composition mechanisms in use. The SDT degree distribution of our service repository can be shown in the following diagram:

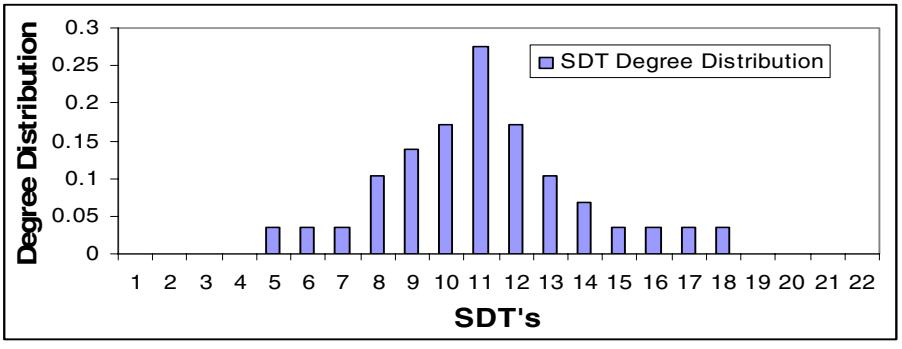


Fig. 4. Degree Distribution for the SDT's present within the web service repository used for our experiments (InformationQuery domain)

In the above diagram, each bar represents a semantic data type whose degree distribution can be shown on the Y axis. We can see that our service repository follows a rather normal (Poisson) distribution. However, many distribution patterns can be applied to hyperlinked network environments, such as *small world networks*, *random graph networks*, *scale-free* and *power-law* distributions [SITGES, 2002]. As a major future work direction, we intend to apply the methodologies described in this paper, to simulated service repositories of different topologies and examine how different topologies affect composition performance.

6 Conclusions – Further Work

In this paper we presented our work in the area of semantic web service composition. The work that has been performed in this area has two aspects: first, we presented the architectural framework we have built in order to experiment with service composition methodologies; second, we presented a set of approaches/techniques in order to tackle the problem at a typical service composition scenario. Our solution approach regards the service composition problem as a search problem within a scalable, dynamic and potentially large search space. In addition, the service repository is regarded as a hyperlinked environment consisting of web resources linking to other web resources. Our work mainly focuses on the heuristics guiding the composition process, rather than the composition algorithm itself; our experiments have been performed using a depth-first search algorithm, which works in a best-first fashion. However, different algorithms can be supported by our platform (see below).

There have been a number of approaches in the service composition problem domain, and some of them were discussed earlier in this paper. We believe that our work is different than the approaches presented and contributes to this particular research area in a number of ways: we understand that service composition is a problem that can be formed in a number of different use cases, each of them having its own requirements and characteristics. Furthermore, we are using the dynamic structure of the search space itself, in order to generate effective heuristic mechanisms to guide the composition process – we follow this methodology having been inspired by research done in search

engine technologies and hyperlinked environments. This way, we consider our approach to be highly dynamic and scalable. Finally, to our knowledge, no experimental approach following these principles has been presented at the time of writing.

The experiments presented in this paper describe three different scenarios on how we can use the service repository's link structure in order to guide the workflow composition process in an effective way: the first one refers to metrics of *global importance* of the web services present in our search domain, denoted by global PageRank figures; the second approach normalizes the global PageRank figures towards particular composition requests (topic-sensitive PageRank), whereas the third scenario examines the use of the *ontology-based semantic distance* metric between the semantic data flows generated during the composition process. The results presented in the previous section give an idea of which approach performs better in our search domain – furthermore, a justification of our results is also attempted. In more detail, we saw that the topic-sensitive PageRank metric achieves the best performance in our experiments; this fact can be justified by taking into account the nature of the heuristic metrics themselves: global PageRank figures are request-independent, and SDT semantic distance to guide the composition process can achieve good performance when the distribution of the semantic distance lengths between all SDT pairs within our ontologies is similar to the degree distribution of the same SDT pairs within our search space. This relation has not been taken into account within the context of our experiments. No attempt has been yet made to compare our experimental results with those coming from alternative service discovery and composition approaches: within our work, we view the service discovery and composition problem mostly as a search problem, and we measure the performance of our methodologies by examining how effective the search is, in terms of expanded nodes. To our knowledge, there is no other service composition approach using search techniques that has published experimental results, as of this writing; thus, immediate comparisons cannot be made easily – however, more work will be done in this area in the future.

By comparing composition performance with PR growth rates within our service repository, we saw that performance is higher when PR growth rates are higher at increasing depths – this means that the composition methodologies examined in this paper are more efficient in condense service repositories, rather than sparse ones. In this context, a major future work direction would be to apply the heuristic methodologies presented in this paper in simulated service repositories that follow different degree distribution and scalability patterns. In addition, the behaviour of different search algorithms can be evaluated as well (such as bi-directional search, or backward chaining, working with backward PR figures) and their performance can be measured, using the heuristic models we have described. We also have to note that combining more than one of the above methodologies within a single search algorithm / composer and examining the combined performance is also an interesting option. Finally, an interesting work direction would be to re-assess each of the above methodologies in relation to their computational costs: from the three methodologies used in our experimental setting, global importance PR is the only one that does not require any composition-time processing, as the PR figures are pre-calculated. Topic-Sensitive PR and SDT semantic distance require some more processing at composition time, so it would be interesting to examine their performance benefits in relation to the computation cost they entail.

Furthermore, we need to note that several other issues concerning the service composition domain in general have on purpose been left out of this paper. One major issue is of course, workflow control constructs: repetition structures, splits and decision blocks. We have done some work in this area as well, especially regarding the nature of use cases that can cause different control structures to appear in the workflow. An example could be request-specific constraints or dynamic constraints inherent to a particular composition request resulting in decision blocks within a workflow. However, such cases have not been discussed in this paper - instead, we have focused on the core *business logic* of the workflow.

References

[ANOKOLEKAR *et al*., 2002] International Semantic Web Conference 2002 (ISWC'02), Sardinia, Italy.

[BERNERS-LEE, 2001]. Keynote presentation on web services and the future of the web. Software Development Expo 2001 Visionary Keynote,

[BHARAT and MIHAILA, 2000] Hilltop: A Search Engine based on Expert Documents, *Ninth International World Wide Web Conference (WWW9)*, Amsterdam, 2000.

[CASATI *et al*., 2001] Fabio Casati, Ming-Chien Shan and D. Georgakopoulos (2001). "E-Services - Guest editorial." The VLDB Journal 10(1): 1.

[FOO *et al.,* 1992] Norman Foo, Brian J. Garner, Anand Rao, Eric Tsui, Semantic Distance in Conceptual Graphs, Conceptual structures: current research and practice, pages 149-154, 1992, ISBN no: 0-13-175878-0.

[HAVELIWALA, 2002] Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search, WWW 2002, THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE.

[HESS and KUSMERICK, 2005] Anderas Hess and Nickolas Kusmerick, Machine learning techniques for annotating semantic web services, *Dagstuhl Seminar, Machine Learning for the Semantic Web*, 13-18 February 2005.

[ITO *et al*., 2004] Application of Kernels to Link Analysis: First Results, In Proceedings of the Second Workshop on Mining Graphs, Trees and Sequences (MGTS'04), pp.13-24, Pisa, September 2004.

[KHOSHAFIAN, 2002] Web Services and Virtual Enterprises. Published by *Tect Enterprises.*

[KLEINBERG, 1997] J. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM, 1999. Also appears as IBM Research Report RJ 10076, May 1997. http://www.cs.cornell.edu/home/kleinber/auth.ps.

[KOPENA and REGLI, 2003] DAMLJessKB: A Tool For Reasoning With The Semantic Web, *2nd International Semantic Web Conference (ISWC2003)*, October 2003, Florida, USA.

[HOGG *et al*., 2004] An Evaluation of Web Services in the Design of a B2B Application. *The 27th Australasian Computer Science Conference, The University of Otago, Dunedin, New Zealand.*

[KAWAMURA *et al*., 2003] Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry, *ICSOC '03*, 2003.

[LIMTHANMAPHON and ZHANG, 2003] Limthanmaphon, B. and Zhang, Y: Web Service Composition with Case-Based Reasoning. In Proc. *Fourteenth Australasian Database Conference, Adelaide, Australia. Conferences in Research and Practice in Information Technology*, 2003. Schewe, K.-D. and Zhou, X., (eds). 4 – 7.

[MIN and BJORNSSON, 2004] Construction Supply Chain Visualization Through Web Services Integration, *CIFE Technical Report #149, MAY 2004*
STANFORD UNIVERSITY.

[MONTEBELLO and ABELA, 2002] DAML Enabled Web Services and Agents in the Semantic Web. *NODe 2002 Web and Database-Related Workshops on the Web*, p.46-58, 2002.

[PAGE and BRIN, 1998] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In WWW Conference, volume 7, 1998. http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm

[PAOLUCCI *et al*., 2002] Semantic Matching of Web Service Capabilities, *ISWC '02*, 2002.

[PEER, 2003] Peer, Joachim: Towards Automatic Web Service Composition Using AI planning Techniques (first draft), 2003.

[PONNEKANTI and FOX, 2002] SWORD: A Developer Toolkit for Web Service Composition, *WWW2002, THE ELEVENTH INTERNATIONAL WORLD WIDE WEB CONFERENCE,* Honolulu, Hawaii, USA, 7-11 May 2002.

[RICHARDS *et al*, 2003] Richards, Debbie *et al*: Composing Web Services Using an Agent Factory, *AAMAS 2003, Workshop on Web Services and Agent-based Engineering*, Melbourne, Australia, July 2003, p.1.

[RODDICK *et al*., 2003] A Unifying Semantic Distance Model for Determining the Similarity of Attribute Values, *26th Australasian computer science conference on Conference in research and practice in information technology*, 2003.

[SEMANTIC WEB, 2001] http://www.w3.org/2001/sw/.

[SITGES, 2002] Statistical Mechanics of Complex Networks, Eds. R. Pastor-Satorras, J. M. Rubi, and A. Diaz-Guilera (Springer, Berlin, 2003).

[TUT and EDMOND, 2002] The use of Patterns in Service Composition. *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, 2002, p.28-40.

[WU *et al*., 2003] Automating DAML-S Web Services Composition Using SHOP2, *13Th International Conference on Automated Planning & Scheduling*, Trento (Italy), June 9-13 2003.

Two Reasoning Methods for Extended Fuzzy ALCH*

Dazhou Kang[1,2], Jianjiang Lu[1,2,3], Baowen Xu[1,2,4], Yanhui Li[1,2], and Yanxiang He[4]

[1] Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China
[2] Jiangsu Institute of Software Quality, Nanjing 210096, China
[3] PLA University of Science and Technology, Nanjing 210007, China
[4] State Key Laboratory of Software, Wuhan University, Wuhan 430072, China
bwxu@seu.edu.cn

Abstract. This paper introduces cut sets of fuzzy concepts and fuzzy roles as atomic concepts and atomic roles to build extended fuzzy $\mathcal{ALCH}$ ($\mathcal{EFALCH}$), a new fuzzy extension of $\mathcal{ALCH}$. It talks about two reasoning methods for $\mathcal{EFALCH}$. The first one presents a sound and complete algorithm for $\mathcal{EFALCH}$ reasoning tasks and proves the complexity is PSPACE-complete. The second one reduces $\mathcal{EFALCH}$ into $\mathcal{ALCH}$, which can be performed in polynomial time and keep the semantic consistency, and then uses $\mathcal{ALCH}$ reasoning method to solve $\mathcal{EFALCH}$ reasoning tasks.

1 Introduction

Description logics (DLs) provide a logical reconstruction of object-centric and frame-based knowledge representation languages [1], with a simple well-established Tarski-style declarative semantics to capture the meaning of popular features in structured knowledge representation. Nowadays a lot of knowledge representation systems have been built by DLs in a variety of applications.

Some DLs applications, such as Web multimedia information retrieval, often need management of fuzzy information; but DLs are limited to dealing with crisp concepts and crisp roles. Therefore, it is necessary to extend description logics with fuzzy capability.

Compared with concepts and roles in classical description logics, fuzzy description logics [2, 3] contain fuzzy concepts and fuzzy roles that describe fuzzy sets. In fuzzy set theory, a fuzzy set S w.r.t a universe U is defined as a membership function $\mu_S : U \to [0,1]$, and the λ-cut set of S is defined as $S_{[\lambda]} = \{d \in U \mid \mu_s(d) \geq \lambda\}$, where $0 < \lambda \leq 1$. Based on the idea of that the cut sets are indeed crisp sets, but

* This work was supported in part by the NSFC (60373066, 60425206, 90412003), National Grand Fundamental Research 973 Program of China (2002CB312000), National Research Foundation for the Doctoral Program of Higher Education of China (20020286004), Excellent Ph.D. Thesis Fund of Southeast University, and Advanced Armament Research Project (51406020105JB8103).

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1588 – 1595, 2005.

facilitate a normative theory for formalizing fuzzy set theory, our fuzzy extension of description logics uses cut sets of fuzzy concepts and fuzzy cut roles instead of fuzzy concepts and fuzzy roles.

Definition 1. Consider three disjoint sets: a set N_C of fuzzy concept names (denoted A, B), a set N_R of fuzzy role names (denoted R, S), and a set N_I of individual names (denoted a, b). For any $A \in N_C$, $R \in N_R$ and $0 < \lambda \leq 1$, we call $A_{[\lambda]}$ an atomic cut concept and $R_{[\lambda]}$ an atomic cut role, where A and R are the prefixes of λ, and λ is the suffix of A and R.

The semantics of fuzzy concept names and their cut sets are defined in terms of an interpretation $\mathcal{I} =< \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$. The domain $\Delta^{\mathcal{I}}$ is a nonempty set and the interpretation function $\cdot^{\mathcal{I}}$ maps every individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, every fuzzy concept name $A \in N_C$ to a membership function $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0,1]$, and every fuzzy role name $R \in N_R$ to a membership function $R^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0,1]$. And $\cdot^{\mathcal{I}}$ maps $A_{[\lambda]}$ and $R_{[\lambda]}$ such that $0 < \lambda \leq 1$ to sets over $\Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$\begin{aligned} (A_{[\lambda]})^{\mathcal{I}} &= \{d \mid d \in \Delta^{\mathcal{I}} \wedge A^{\mathcal{I}}(d) \geq \lambda\}; \\ (R_{[\lambda]})^{\mathcal{I}} &= \{(d, d') \mid d, d' \in \Delta^{\mathcal{I}} \wedge R^{\mathcal{I}}(d, d') \geq \lambda\}. \end{aligned} \tag{1}$$

From Equation 1, for any λ_1, λ_2 such that $0 \leq \lambda_2 \leq \lambda_1 \leq 1$, it must be true that $(A_{[\lambda_1]})^{\mathcal{I}} \subseteq (A_{[\lambda_2]})^{\mathcal{I}}$ and $(R_{[\lambda_1]})^{\mathcal{I}} \subseteq (R_{[\lambda_2]})^{\mathcal{I}}$ for any interpretation $\mathcal{I}$. Generally, a collection of $(A_{[\lambda_1]})^{\mathcal{I}}, (A_{[\lambda_2]})^{\mathcal{I}}, \cdots, (A_{[\lambda_k]})^{\mathcal{I}}$ and $(R_{[\lambda_{k+1}]})^{\mathcal{I}}, (R_{[\lambda_{k+2}]})^{\mathcal{I}}, \cdots, (R_{[\lambda_n]})^{\mathcal{I}}$ can describe the semantics of $A^{\mathcal{I}}$ and $R^{\mathcal{I}}$ completely or at an acceptable degree.

2 Extended Fuzzy Description Logic $\mathcal{EFALCH}$

2.1 Cut Concepts

We propose a fuzzy extension of the description logic $\mathcal{ALCH}$ [1], called $\mathcal{EFALCH}$, by introducing cut concepts and cut roles. $\mathcal{EFALCH}$ inherits all the concept constructors from $\mathcal{ALCH}$, including negation, conjunction, disjunction, value restriction, and existential restriction. There is no role constructor in either $\mathcal{ALCH}$ or $\mathcal{EFALCH}$.

Definition 2. The cut concepts in $\mathcal{EFALCH}$ are syntactically defined as

$$C, D ::= A_{[\lambda]} \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R_{[\lambda]}.C \mid \exists R_{[\lambda]}.C \tag{2}$$

where $A \in N_C$, $R \in N_R$, and $0 < \lambda \leq 1$. The semantics of cut concepts are defined in terms of $\mathcal{I} =< \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$ in Definition 1 with the extension of $\cdot^{\mathcal{I}}$ to arbitrary cut concepts, which are inductively defined as following:

$$
\begin{aligned}
&(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}; (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \bigcap D^{\mathcal{I}}; (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \bigcup D^{\mathcal{I}};\\
&(\exists R_{[\lambda]}.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists d' \in \Delta^{\mathcal{I}}, (d,d') \in (R_{[\lambda]})^{\mathcal{I}} \wedge d' \in C^{\mathcal{I}}\};\\
&(\forall R_{[\lambda]}.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \forall d' \in \Delta^{\mathcal{I}}, (d,d') \in (R_{[\lambda]})^{\mathcal{I}} \rightarrow d' \in C^{\mathcal{I}}\}.
\end{aligned} \tag{3}
$$

2.2 Axioms and Assertions

An $\mathcal{EFALCH}$ knowledge base contains alterable terminological axioms of concepts represented in a TBox, alterable role inclusion axioms represented in a role hierarchy, as well as assertions represented in an ABox.

Definition 3. For any $A \in N_C$ and $R \in N_R$, $A_{[f(u)]}$ ($R_{[f(u)]}$) is an alterable atomic cut concept (role) decided by u, if u is a variable in a continuous domain $V \subseteq (0,1]$ and f is a function from V to $(0,1]$. Starting with alterable atomic cut concepts and roles, alterable cut concepts (denoted E, F) can be inductively defined as: $E, F ::= A_{[f(u)]} \mid \neg E \mid E \sqcap F \mid E \sqcup F \mid \forall R_{[f(u)]}.C \mid \exists R_{[f(u)]}.E$.

Now we make some constraints of alterable cut concept E:

1) All alterable atomic cut concepts and atomic cut roles in E must be decided by a single variable u with same domain V, then we say E is decided by u (denoted $E_{(u)}$), and V is the domain of $E_{(u)}$. Let c be a constant in V, then $E_{(c)}$ is a cut concept which replaces any suffix $f(u)$ in E with $f(c)$ (denoted $E|_{u=c}$).

2) For $E_{(u)}$ and its domain V, $E_{(u)}$ is monotonous (either increasing or decreasing) over V. $E_{(u)}$ is increasing (decreasing) over V if for any $\lambda_1, \lambda_2 \in V$, $\lambda_1 \le \lambda_2$ and any interpretation I, $(E|_{u=\lambda_1})^I \supseteq (\subseteq) (E|_{u=\lambda_2})^I$ holds. And the monotony constraint holds for any alterable atomic cut role.

Definition 4. An $\mathcal{EFALCH}$ TBox is a finite set of terminological axioms of the form $< E_{(u)} \sqsubseteq F_{(u)}, u \in V >$, where $E_{(u)}, F_{(u)}$ are alterable cut concepts and have the same domain V and monotony ($E_{(u)}, F_{(u)}$ are both increasing or decreasing over V). An interpretation $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ iff $\forall c \in V, (E|_{u=c})^{\mathcal{I}} \subseteq (F|_{u=c})^{\mathcal{I}}$ holds for every $< E \sqsubseteq F, u \in V >$ in $\mathcal{T}$. A TBox is consistent iff it has a model.

Definition 5. An alterable cut role inclusion axiom is of the following form $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V >$, where $R_{[f(u)]}, S_{[f'(u)]}$ are alterable cut roles and have the same domain V and monotony ($R_{[f(u)]}$ and $S_{[f'(u)]}$ are both increasing or decreasing over V. Without loss of generality, we assume $f(u)$ and $f'(u)$ are monotone and increasing). And an $\mathcal{EFALCH}$ role hierarchy is a finite set of cut role inclusion axioms with the following two restrictions:

1) For any fuzzy role R in $\mathcal{H}$, R appears in the left side of cut role inclusion once at most.

2) $\mathcal{H}$ does not contain cyclic inclusions, such as $< R_{1[f_1(u)]} \doteq R_{2[f_2(u)]}, u \in V_2 >, \ldots,$ $< R_{k[f(u)]} \doteq R_{1[f_{k+1}(u)]}, u \in V_k >$, which means a fuzzy role is included by itself.

An interpretation $\mathcal{I}$ is a model of a role hierarchy $\mathcal{H}$ iff $\forall c \in V$, $(R_{[f(c)]})^{\mathcal{I}} \subseteq (S_{[f'(c)]})^{\mathcal{I}}$ for every $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V >$ in $\mathcal{H}$. A role hierarchy is consistent iff it has a model.

The assertions in $\mathcal{EFALCH}$ is similar to assertions in $\mathcal{ALCH}$ by replacing crisp concept and role with cut concept and role.

Definition 6. An $\mathcal{EFALCH}$ ABox is a finite set of assertions of the form $a:C$ (cut concept assertion) or $(a,b):R_{[\lambda]}$ (cut role assertion), where C is a cut concept, $R_{[\lambda]}$ a cut role, and $a,b \in N_I$. An interpretation $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all $a:C$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in (R_{[\lambda]})^{\mathcal{I}}$ for all $(a,b):R_{[\lambda]}$ in $\mathcal{A}$. An ABox is consistent iff it has a model.

An interpretation $\mathcal{I}$ is a model of $\sum = (\mathcal{A}, \mathcal{T}, \mathcal{H})$, iff $\mathcal{I}$ is a model of $\mathcal{T}$, $\mathcal{A}$ and $\mathcal{H}$. $\sum$ is consistent iff it has a model.

3 Two Reasoning Methods for $\mathcal{EFALCH}$

In classical description logics, *satisfiability* is considered as the main reasoning task, and reasoning algorithms are usually firstly developed for *satisfiability* and secondly extended to solve the other reasoning tasks [1, 4]. Similarly, in $\mathcal{EFALCH}$ case, we consider *satisfiability* of cut concept as a basic reasoning task.

Satisfiability: a cut concept C_0 is satisfiable w.r.t $\mathcal{T}$ and $\mathcal{H}$ iff there is an interpretation $\mathcal{I}$ such that $C_0^{\mathcal{I}} \neq \varnothing$, and $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{H}$.

This section will talk about two reasoning methods for *satisfiability* w.r.t empty TBox. The first one gives a tableau algorithm, and the second one reduces $\mathcal{EFALCH}$ to $\mathcal{ALCH}$ and reuses the $\mathcal{ALCH}$ reasoning method.

3.1 The Tableau Algorithm

For $\mathcal{EFALCH}$, the tableau algorithm for *satisfiability* starts with the ABox A_0={x_0:C_0}, then extends A_0 by translation rules and tries to create complete and not-closed ABoxes, which are directly converted into fuzzy interpretations satisfying C_0.

Because $\mathcal{EFALCH}$ supports alterable cut role inclusion axioms, we use "$R_{k[\lambda_k]}$-successor" to propagate the constraint of alterable cut role inclusion. $R_{k[\lambda_k]}$-successor is defined as: y is a $R_{k[\lambda_k]}$-successor of x if $\mathcal{A}$ contains $(x,y):R_{1[\lambda_1]}$ and there are a cut role sequence $R_{1[\lambda_1]}, R_{2[\lambda_2]}, \cdots, R_{k[\lambda_k]}$ such that for any $R_{i[\lambda_i]}$ and $R_{i+1[\lambda_{i+1}]}$, one of the following two conditions must be satisfied:

1) $R_i = R_{i+1}$ and $\lambda_i \geq \lambda_{i+1}$;

2) $< R_{i[f(u)]} \sqsubseteq R_{i+1[f'(u)]}, u \in V >$ is in $\mathcal{H}$. Let f^{-1} be the inverse function of f (for the monotony constraints, f is monotone). $f'(f^{-1}(\lambda_i)) \geq \lambda_{i+1}$ holds.

Now, we express our tableau algorithm as follows. The tableau algorithm starts with a single ABox $A_0=\{x_0:C_0\}$ and propagates constraints by applying translation rules (Fig.1.) to ABoxes.

For any ABox A which is not closed

The $\sqcap$ -rule

Condition: A contains x: $D \sqcap E$, but not both x: D and x: E .

Action: $A^1=A \bigcup \{x: D , x: E \}$.

The $\sqcup$ -rule

Condition: A contains x: $D \sqcup E$ but neither x: D nor x: E .

Action: $A^1=A \bigcup \{x: D \}$; $A^2=A \bigcup \{x: E \}$.

The $\exists$ -rule

Condition: A contains x: $\exists R_{[\lambda]}.D$, but not a $R_{[\lambda]}$ -successor y of x satisfying y: D .

Action: $A^1 =A \bigcup \{z: D , (x, z): R_{[\lambda]} \}$, where z is a new generated individual.

The $\forall$ -rule

Condition: A contains x: $\forall R_{[\lambda]}.D$ and a $R_{[\lambda]}$ -successor y of x without y: D .

Action: $A^1=A \bigcup \{y: D \}$.

The $\neg$ -rule

Condition: A contains x: $\neg B_{[\lambda_1]}$ and x: $B_{[\lambda_2]}$, where $\lambda_1 \leq \lambda_2$.

Action: Announce A is closed.

Fig. 1. Translation rules of the tableau algorithm for $\mathcal{EFALCH}$

New ABox A^1 or A^2 is called a subsequence of A. By applying some translation rules, a given ABox A may be translated into two subsequences. Therefore we define a set of ABoxes S. The tableau algorithm starts with a single set $S_0=\{\{x_0:C_0\}\}$. And the translation rules are applied to a given set S with replacing one alterable ABox by its subsequences.

Definition 7. An alterable ABox A is complete iff none of the translation rules can be applied to it. An alterable ABox is not closed if no $\neg$ -rule announces it is closed. An alterable ABox is open iff it is complete and not closed. A finite set of alterable ABoxes S is complete iff any ABox A in S is complete.

Now we refine process of tableau algorithm in the following steps.

1) Let C_0 be an $\mathcal{EFALCH}$ cut concept in *NNF* [1, 4]. The tableau algorithm starts with a set of a single alterable ABox: $S_0=\{\{x_0:C_0\}\}$;

2) The tableau algorithm exhaustively applies the translation rules to current S_i. We denote the subsequence of S_i by S_{i+1}. So there is a chain of S_i by rules application: $S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_i \rightarrow \ldots$;

3) If the current S_i is complete, if there is an ABox A^* in S_i is open, the tableau algorithm returns "C_0 is satisfiable", otherwise "C_0 is unsatisfiable".

The completeness of the algorithm is guaranteed for any translation rule is based on constraint propagation. And the soundness is proved by that for any complete and not-closed ABox A, there exists a model of A. The algorithm could be executed in polynomial space as a similar consequence of the algorithm for $\mathcal{ALCH}$ concept satisfiability w.r.t empty TBox. For $\mathcal{ALCH}$ concept satisfiability is PSPACE-complete [1], $\mathcal{EFALCH}$ concept satisfiability w.r.t empty TBox is PSPACE-hard. We have the following theorem.

Theorem 1. Satisfiability of $\mathcal{ALCH}$ cut concepts w.r.t. empty TBox is PSPACE-complete.

3.2 Reducing $\mathcal{EFALCH}$ to $\mathcal{ALCH}$

In this subsection, for any $\mathcal{EFALCH}$ cut concept C_0 and role hierarchy $\mathcal{H}$, we use a $\mathcal{ALCH}$ knowledge base to simulate them. The simulation process consists of the following steps.

For any cut concept C_0 and role hierarchy $\mathcal{H}$

1) Let $T(C_0)=_{def}\{ B_{[\lambda_1]} \sqsubseteq B_{[\lambda_2]} \mid B_{[\lambda_1]}$ and $B_{[\lambda_2]}$ are in C_0, and $\lambda_1 \geq \lambda_2 \}$.

2) Initialize Role(C_0, $\mathcal{H}$)$=\{ R_{[\lambda_1]} \mid R_{[\lambda_1]}$ is in $C_0\}$ and $H^*(C_0, \mathcal{H})=\varnothing$.

3) For any $R_{[\lambda_1]} \in$ Role(C_0, $\mathcal{H}$), if $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V > \in \mathcal{H}$ and $\lambda_1 \geq \inf(V)$, then add $S_{[f'(\lambda)]}$ into Role(C_0, $\mathcal{H}$) and $R_{[f(\lambda_1)]} \sqsubseteq S_{[f'(\lambda)]}$ into $H^*(C_0, \mathcal{H})$, where $\lambda = \min(\lambda_1, \sup(V))$.

4) Initialize $H(C_0, \mathcal{H})=_{def}\{ R_{[\lambda_1]} \sqsubseteq R_{[\lambda_2]} \mid R_{[\lambda_1]}$ and $R_{[\lambda_2]}$ are in Role(C_0, $\mathcal{H}$), and $\lambda_1 \geq \lambda_2 \}$. Let $H(C_0, \mathcal{H})= H(C_0, \mathcal{H}) \bigcup H^*(C_0, \mathcal{H})$.

5) Construct a $\mathcal{ALCH}$ knowledge base $\sum(T(C_0), H(C_0, \mathcal{H}), \{x{:}C_0\})$, where $B_{[\lambda_1]}$ and $R_{[\lambda_1]}$ are considered as $\mathcal{ALCH}$ concept and role.

Fig. 2. Steps of simulation process

For C_0 contains linear number of atomic cut concepts, step 1 can be performed in polynomial time. For $\mathcal{H}$ has two restriction (definition 5), step 2 and 3 can be performed in polynomial time, and the cardinality of the sets Role(C_0, $\mathcal{H}$) and $H^*(C_0$, $\mathcal{H}$) are also polynomial. Therefore step 4 and 5 can also be accomplished in polynomial time. Above all, the total time of simulation process is polynomial. Now we will prove the completeness and soundness of the simulation process.

Theorem 2. An $\mathcal{EFALCH}$ cut concept C_0 is satisfiable w.r.t a role hierarchy $\mathcal{H}$ iff the $\mathcal{ALCH}$ knowledge base $\sum(T(C_0), H(C_0, \mathcal{H}), \{x{:}C_0\})$ has a model.

Proof.

$\Rightarrow$.) Let $I_E(\Delta^{I_E}, \bullet^{I_E})$ be an $\mathcal{EFALCH}$ model of $\mathcal{H}$ and $(C_0)^{I_E} \neq \varnothing$. We create an $\mathcal{ALCH}$ interpretation $I(\Delta^I, \bullet^I)$ satisfying $\sum(T(C_0), H(C_0, \mathcal{H}), \{x{:}C_0\})$ from I_E in the

following steps: 1) $\Delta^I = \Delta^{I_E}$; 2) for any atomic concept $B_{[\lambda_i]}$ and atomic role $R_{[\lambda_i]}$ in $\sum$, $(B_{[\lambda_i]})^I = (B_{[\lambda_i]})^{I_E}$, $(R_{[\lambda_i]})^I = (R_{[\lambda_i]})^{I_E}$.

Now, we prove I satisfies $\sum(T(C_0), H(C_0, \mathcal{H}), \{x{:}C_0\})$.

1) For any $B_{[\lambda_1]} \sqsubseteq B_{[\lambda_2]}$ in $T(C_0)$: since $\lambda_1 \geq \lambda_2$, $(B_{[\lambda_1]})^{I_E} \subseteq (B_{[\lambda_2]})^{I_E}$. For $(B_{[\lambda_i]})^I = (B_{[\lambda_i]})^{I_E}$, we can induce that $(B_{[\lambda_1]})^I \subseteq (B_{[\lambda_2]})^I$. So I satisfies $T(C_0)$.

2) For any $R_{[f(\lambda)]} \sqsubseteq S_{[f'(\lambda)]}$ in $H(C_0, \mathcal{H})$: if it is added into $H(C_0)$ in step 3, $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V > \in \mathcal{H}$ and $\lambda \in V$. For I_E satisfies $\mathcal{H}$, $(R_{[f(\lambda)]})^{I_E} \subseteq (S_{[f'(\lambda)]})^{I_E}$. Therefore $(R_{[f(\lambda)]})^I \subseteq (S_{[f'(\lambda)]})^I$ holds. If this inclusion is added into $H(C_0)$ in step 4, the proof is similar to case 1. Above all I satisfies $H(C_0, \mathcal{H})$.

3) For the semantics of $\mathcal{ALCH}$ concept constructors are the same as $\mathcal{EFALCH}$ and for any $B_{[\lambda_i]}$ and $R_{[\lambda_i]}$, $(B_{[\lambda_i]})^I = (B_{[\lambda_i]})^{I_E}$, $(R_{[\lambda_i]})^I = (R_{[\lambda_i]})^{I_E}$ holds, for any concept C in $\sum$, $(C)^I = (C)^{I_E}$. Then $(C_0)^I = (C_0)^{I_E} \neq \varnothing$. Choose one element d in $(C_0)^I$ and additional define $x^I = d$. Obviously I satisfies $\{x{:}C_0\}$.

$\Leftarrow$.) Let $I(\Delta^I, \bullet^I)$ be a model of $\sum(T(C_0), H(C_0, \mathcal{H}), \{x{:}C_0\})$. We create an $\mathcal{EFALCH}$ model I_E of $\mathcal{H}$ with $(C_0)^{I_E} \neq \varnothing$ from I in the following steps: 1) $\Delta^{I_E} = \Delta^I$; 2) for any atomic fuzzy concept B, atomic fuzzy role R and $d, d' \in \Delta^{I_E}$,

$$B^{I_E}(d) = max\{\{\lambda_i \mid d \in (B_{[\lambda_i]})^I\} \cup \{0\}\},$$

$$R^{I_E}(d,d') = max\{\{\lambda_i \mid (d,d') \in (R_{[\lambda_i]})^I\} \cup \{0\}\}.$$

Now, we prove I_E satisfies $\mathcal{H}$ and C_0.

1) I_E satisfies $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V >$ iff for any pair $(d,d') \in \Delta^{I_E} \times \Delta^{I_E}$ and $u \in V$, $R^{I_E}(d,d') \geq f(u) \rightarrow S^{I_E}(d,d') \geq f'(u)$. From the definition of R^{I_E}, we can get that $R^{I_E}(d,d') \geq f(u) \rightarrow \exists \lambda_j, (d,d') \in (R_{[f(\lambda_j)]})^I$ and $\lambda_j \geq u$. From the definition of step 3, there should be $R_{[f(\lambda_j)]} \sqsubseteq S_{[f'(\lambda)]}$ in $H(C_0, \mathcal{H})$, where $\lambda = \min(\lambda_j, \sup(V))$. As a consequence, $S^{I_E}(d,d') \geq f'(\lambda) \geq f'(u)$. Therefore I_E satisfies any alterable cut role inclusion axiom $< R_{[f(u)]} \sqsubseteq S_{[f'(u)]}, u \in V >$. And I_E is a model of $\mathcal{H}$.

2) From B^{I_E}, $(B_{[\lambda_i]})^{I_E} = \{d \mid B^{I_E}(d) \geq \lambda_i\} = \{d \mid \exists \lambda_j, d \in (B_{[\lambda_j]})^I \text{ and } \lambda_j \geq \lambda_i\}$. As a consequence, $(B_{[\lambda_i]})^{I_E} = (B_{[\lambda_i]})^I$. And $(R_{[\lambda_i]})^{I_E} = (R_{[\lambda_i]})^I$ is similarly proved. Then for any concept C in C_0, $(C)^{I_E} = (C)^I$. Therefore $(C_0)^{I_E} \neq \varnothing$.

4 Related Work

Lots of endeavors have done for extensions of DLs with fuzzy features. Meghini et al proposed a preliminary fuzzy DL [5], which lacks reasoning algorithm, as a modeling tool for multimedia information retrieval.

Straccia [2] presented fuzzy $\mathcal{ALCH}$, which is a fuzzy extension of $\mathcal{ALCH}$ by adopting fuzzy interpretation to redefine the semantics and extending the axiom forms. However, Fuzzy $\mathcal{ALCH}$ only supports limited and insufficient expressive power of both assertional and terminological fuzzy knowledge.

For a fuzzy concept $\exists R.C$ and an individual a, fuzzy $\mathcal{ALCH}$ supports fuzzy assertion of the form $a:\exists R.C \geq \lambda$, which means $\exists b^{\mathcal{I}} \in \Delta^{\mathcal{I}}, R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \lambda$ and $C^{\mathcal{I}}(b^{\mathcal{I}}) \geq \lambda$. However, it cannot describe an individual a such that $\exists b^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \lambda_1$ and $C^{\mathcal{I}}(b^{\mathcal{I}}) \geq \lambda_2$, where $\lambda_1 \neq \lambda_2$. Generally, fuzzy $\mathcal{ALCH}$ is not able to describe different membership degrees of concepts and roles in a single assertion. Such complex fuzzy assertion information can be described as $a: \exists R_{[\lambda_1]}.C_{[\lambda_2]}$ in $\mathcal{EFALCH}$ ABox.

Similarly, the same problem happens in the axioms of fuzzy $\mathcal{ALCH}$. They can not express complex inclusions based on various membership degrees. For example, the axiom $C \sqsubseteq D$ only means $\forall d \in \Delta^{\mathcal{I}}$, $C^{\mathcal{I}}(d) \geq \lambda \rightarrow D^{\mathcal{I}}(d) \geq \lambda$. But sometimes, it is necessary to use $C^{\mathcal{I}}(d) \geq \lambda_1 \rightarrow D^{\mathcal{I}}(d) \geq \lambda_2$, where $\lambda_1 \neq \lambda_2$. Such inclusion can be described as: $C_{[\lambda_1]} \sqsubseteq D_{[\lambda_2]}$ in $\mathcal{EFALCH}$ TBox.

5 Conclusions

This paper talks about two reasoning methods for $\mathcal{EFALCH}$. The first one presents a sound and complete algorithm for $\mathcal{EFALCH}$ reasoning tasks and proves the complexity is PSPACE-complete. The second one reduces $\mathcal{EFALCH}$ into $\mathcal{ALCH}$, which can be performed in polynomial time and keep the semantic consistency, and then uses $\mathcal{ALCH}$ reasoning method to solve $\mathcal{EFALCH}$ reasoning tasks. Further work includes the extension of $\mathcal{EFALCH}$ with more concept and role constructors.

References

[1] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.(Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)

[2] Straccia, U.: Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research, no.14 (2001): 137-166

[3] Straccia, U.: Transforming fuzzy description logics into classical description logics. In: Proceedings of JELIA 2004, Lisbon, (2004) 385-399

[4] Baader, F., Sattler, U.: An Overview of Tableau Algorithms for Description Logics. Studia Logica, Vol. 69, no. 1 (2001): 5-40

[5] Meghini, C., Sebastiani, F., Straccia, U.: Reasoning about the form and content for multimedia objects. In: Proceedings of AAAI 1997 Spring Symposium on Intelligent Integration and Use of Text, Image, Video and Audio, California (1997) 89-94

Expressing Preferences in a Viewpoint Ontology

Rallou Thomopoulos

INRA, UMR IATE, Bâtiment 31,
2 place Viala, 34060 Montpellier Cedex 1, France
Rallou.Thomopoulos@ensam.inra.fr

Abstract. This paper proposes a definition of viewpoints in a "kind of" ontology. The use of viewpoints allows one to simplify user interface and to facilitate the expression of user preferences on such an ontology. This work has been applied in the framework of an information system dedicated to the quality of food products.

1 Introduction

This study has taken place in a French project whose mission is to create a decision-making tool for the analysis of the nutritional and sanitary quality of food products. As a first step of the project, scientific data from several hundreds of publications concerning the impact of technological processes on nutritional or toxic components have been gathered in a database and a querying system has been built in order to explore them.

The question we deal with rises from two characteristics of the data: (i) the data are not abundant enough to answer every query. This characteristic led us to propose a flexible way of expressing the queries, by allowing the user to indicate levels of preference in his search. For instance, the user may ask for milk as a first choice or yoghourt as a second choice; (ii) the data (food products, bacteria, nutritional components, ...) are organized in ontologies. For instance, milk and yoghourt (quoted in the example above), as well as the other food products, are part of a taxonomy of substrates, in which *Whole milk* is a kind of *Milk*, which is a kind of *Milk product*, etc. For the user, asking for milk as a first choice and yoghourt as a second choice means associating preference degrees with the elements of the taxonomy of substrates.

Previous results [1, 2] concerning the expression of preferences in an ontology led to two issues: firstly, the need to simplify graphical user interface, and secondly, the necessity of expressing preferences on domains composed of exclusive and exhaustive elements. Therefore in Section 2 we introduce the notion of viewpoint in an ontology. Section 3 deals with the expression of preferences in a viewpoint ontology, in regard to user interface and to the semantics of such preferences for the querying.

2 Viewpoints in Ontologies

We focus on ontologies defined as sets of elements partially ordered by the "kind of" relation. An example of such an ontology is given (partially) in Figure 1.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1596–1604, 2005.

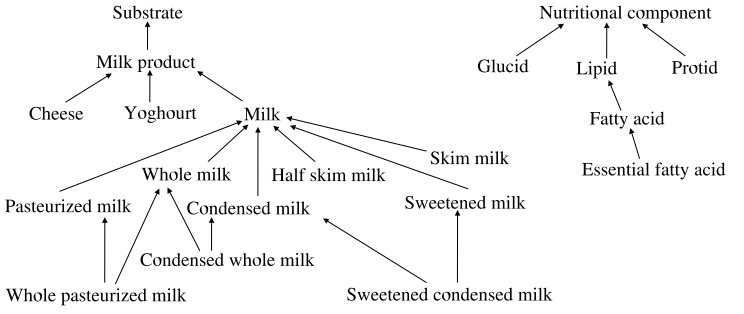


Fig. 1. Part of an ontology

In this section we give a brief introduction to viewpoints, then we propose a definition of a viewpoint ontology.

2.1 Introduction to Viewpoints

In the ontology of Figure 1, we can notice that the subelements of *Milk*, for instance, do not all have the same "statute". Any instance of *Milk* is necessarily an instance of either *Whole milk* or *Half skim milk* or *Skim milk*. That is, these three elements are exclusive and, moreover, they cover all cases of *Milk*. Thus they form a partition of *Milk*. This partition corresponds to a "creaming" criterion. Another partition of *Milk* could be obtained with *Pasteurized milk*, *Raw milk* and *Sterilized milk* (the latter two are not represented in Figure 1). This partition corresponds to a "thermization" criterion. On the contrary, an instance of *Milk* may be an instance of both *Whole milk* and *Pasteurized milk*. These two elements are not exclusive. Therefore they have a common subelement *Whole pasteurized milk*. The "creaming" and "thermization" partitions are complementary. Any instance of *Milk* can be represented in both.

For a given domain of knowledge, several criteria can be used to observe an object. These different perceptions of the world are called viewpoints or perspectives. One of the first references to viewpoints was proposed by [3] with a spacial connotation. Examples of systems that implement viewpoints in object representations are [4, 5, 6, 7]. A good overview is given in [8]. [9] introduces viewpoints in the conceptual graph model, in a "corporate memory" context. In UML (Unified Modeling Language), the specification of viewpoints is possible through the use of labels in multiple generalization: a partition can be represented by the "disjoint" and "complete" generalization constraints.

However viewpoints rely on semantic and subjective notions that are difficult to formalize. Therefore most previous approaches are unformal or operational – they focus on a particular implementation of viewpoints. Systems that do not explicitely deal with viewpoints use multiple inheritance to model them, as in the ontology of Figure 1 for instance. In Figure 1, the "creaming" and "thermization" viewpoints do not explicitly appear, although they could help

the user define querying preferences. Moreover they may have different levels of importance for the user.

2.2 A Definition of Viewpoints

We propose a definition of a viewpoint ontology that relies on specializations of the "kind of" relation. A specialization of the "kind of" relation is based on the criterion used to establish the "kind of" relation between two elements. This citerion can be general or specific, thus leading to several levels of specialization of the "kind of" relation. In formalisms from the family of semantic networks, like description logics [10] or conceptual graphs [11], relations between concepts can be specialized. However the "kind of" relation plays a particular part in these models, as specialization is based on it. Here we propose to specialize the "kind of" relation itself, just as other relations can be specialized.

Definition 1. *A* **specialization** *of the "kind of" relation is a restriction of the "kind of" relation obtained by specifiying the criterion used to establish the "kind of" relation between elements. A particular specialization of the "kind of" relation, denoted "kind of by conjunction", is used to indicate that a common sub-element is obtained by multiple inheritance.*

Remark 1. *The "kind of by conjunction" relation may itself be specialized.*

Example 1. *The "kind of, in regard to thermization" relation and the "kind of, in regard to creaming" relation are specializations of the "kind of, in regard to process" relation, which is a specialization of the "kind of" relation. They are used in the ontology of Figure 2. The abbreviations "ko", "koP", "koT", "koC", "conj" are respectively used for "kind of", "kind of, in regard to process", 'kind of, in regard to thermization", "kind of, in regard to creaming", "kind of by conjunction".*

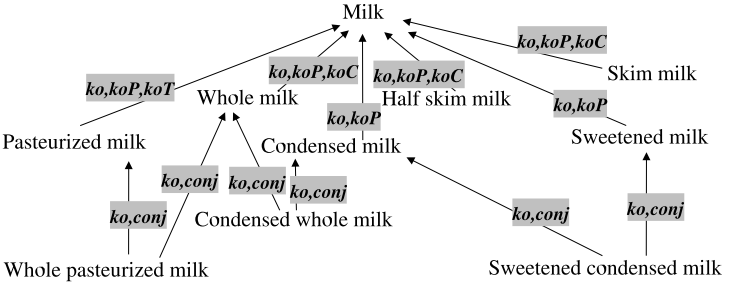


Fig. 2. Examples of specializations of the "kind of" relation in the *Milk* ontology

Example 2. *Figure 3 shows an ontology about fatty acids, in which the following specializations of the "kind of" relation are used:*

– the "kind of, in regard to chemistry" relation (denoted "koCh"), which is itself specialized into the "kind of, in regard to the presence of double bonds" (denoted "koPr") and "kind of, in regard to the number of double bonds" (denoted "koN") relations, is used to distinguish the elements Saturated fatty acid, Unsaturated fatty acid, Mono-unsaturated fatty acid *and* Poly-unsaturated fatty acid*;*
– the "kind of, in regard to state at ambient temperature" relation (denoted "koSt"), is used to distinguish between Solid at ambient temperature fatty acid *and* Liquid at ambient temperature fatty acid*;*
– the "kind of, in regard to origin" relation (denoted "koO"), is used to distinguish between Essential fatty acid *(which cannot be synthetized by the human organism) and* Synthetized fatty acid *(which can be synthetized by the human organism);*
– the "kind of, by conjunction with state at ambient temperature" relation (denoted "conjSt") and the "kind of, by conjunction with presence of double bounds" relation (denoted "conjPr") are both specializations of the "kind of by conjunction" relation. They are used to obtain common subelements of saturated/unsaturated fatty acids and solid/liquid fatty acids (multiple inheritance).

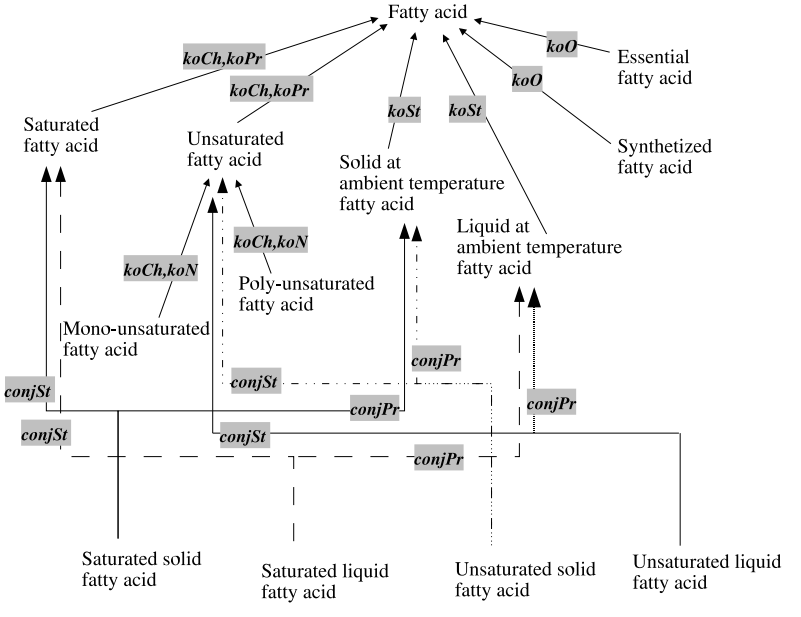


Fig. 3. Examples of viewpoints in the *Fatty acid* ontology

We propose a recursive definition of a viewpoint.

Definition 2. *Let Ω be a set of elements partially ordered by the "kind of" relation and by a set S of its specializations. A* **viewpoint** *on Ω is a pair $(elt, s) \in \Omega \times S$ such that the set of the direct predecessors of elt through s, denoted P, is either empty or satisfies:*

– *P forms a partition of elt;*
– *$\forall p \in P$, (p, s) is a viewpoint.*
The set composed of elt and of its predecessors through s is then called the **view** *induced by* (elt, s).

Remark 2. *Knowledge on partition satisfaction is not derived from the ontology, but declared as expert knowledge (this point is not developed here).*

Example 3. *In Figure 2,* (Milk,koC) *is a viewpoint because the set of direct predecessors of* Milk *through* koC*, that is* {Whole milk, Half skim milk, Skim milk} *forms a partition of* Milk. Whole milk*,* Half skim milk *and* Skim milk *have no predecessors through* koC.

On the contrary, (Milk,koT) *is not a viewpoint: the set of direct predecessors of* Milk *through* koT*, that is* {Pasteurized milk}*, is not a partition of* Milk *because it is not complete, it does not cover all cases of* Milk. (Milk,koP) *is not a viewpoint either: the set of direct predecessors of* Milk *through* koP*, that is* {Pasteurized milk, Whole milk, Condensed milk, Half skim milk, Sweetened milk, Skim milk}*, is not a partition of* Milk *as its elements are not exclusive.*

Example 4. *In Figure 3, all the pairs* (elt, s) *composed of an element of the ontology and a relation among the represented specializations of the "kind of" relation ("koCh", "koPr", "koN", "koSt", "koO", "conjSt" and "conjPr") are viewpoints. We have the three following cases:*
– *elt has no predecessors through s. This is the case, for instance, of the pairs* (Essential fatty acid, koSt)*,* (Saturated fatty acid, koN)*, etc.*
– *elt has direct predecessors through s (that form a partition of elt) and these predecessors have no predecessors through s. This is the case, for instance, of* (Fatty acid, koSt)*,* (Fatty acid, koPr)*,* (Unsaturated fatty acid, koCh)*,* (Saturated fatty acid, conjSt)*, etc.*
– *elt has direct predecessors through s (that form a partition of elt) and some of these predecessors also have predecessors through s. This is the case for* (Fatty acid, koCh).

Property 1. *There is no multiple inheritance within a given view.*

Proof 1. *Having multiple inheritance within a given view v would imply that there exists an element a in v that has two successors b and c through s, such that b and c are not comparable through s but have a non-empty intersection. This is excluded by definition 2, as both b and c are obtained by successive partitions of elt, where non-comparable elements are all exclusive by construction.*

Example 5. *In Figure 3, the element* Saturated solid fatty acid *is obtained by multiple inheritance of both* Saturated fatty acid *and* Solid at ambient temperature fatty acid*, which do not belong to the same views.*

Saturated fatty acid *belongs to the view* {Fatty acid, Saturated fatty acid, Unsaturated fatty acid} *induced by the viewpoint* (Fatty acid, koPr)*, to the view* {Fatty acid, Saturated fatty acid, Unsaturated fatty acid, Mono-unsaturated fatty acid, Poly-unsaturated fatty acid} *induced by the viewpoint* (Fatty acid,

koCh) *and to the view* {Saturated fatty acid, Saturated solid fatty acid, Saturated liquid fatty acid} *induced by the viewpoint* (Saturated fatty acid, conjSt).

Solid at ambient temperature fatty acid *belongs to the view* {Fatty acid, Solid at ambient temperature fatty acid, Liquid at ambient temperature fatty acid} *induced by the viewpoint* (Fatty acid, koSt) *and to the view* {Solid at ambient temperature fatty acid, Saturated solid fatty acid, Unsaturated solid fatty acid} *induced by the viewpoint* (Solid at ambient temperature fatty acid, conjPr).

Definition 3. *A* **viewpoint ontology** Ω_v *is a set of elements partially ordered by a set* S *of specializations of the "kind of" relation, such that each pair* $(elt, s) \in \Omega_v \times S$ *is a viewpoint.*

Example 6. *Figure 3 is an example of a viewpoint ontology.*

3 Preferences in a Viewpoint Ontology

3.1 Simplifying User Interface

In Figure 3, four elements obtained by multiple inheritance have been represented: *Saturated solid fatty acid*, *Saturated liquid fatty acid*, *Unsaturated solid fatty acid* and *Saturated liquid fatty acid*. Their graphical representation makes the ontology much more difficult to read, compared to the same ontology without multiple inheritance being represented, i.e. without conjunction viewpoints.

Moreover Figure 3 is far from being complete. Elements like *Essential unsaturated fatty acid*, *Mono-unsaturated solid fatty acid*, *Mono-unsaturated liquid synthetized fatty acid*, etc., are not represented. Representing all possible conjunction viewpoints would lead to an unreadable result. Indeed, predecessors of *Fatty acid* from different viewpoints on *Fatty acid* using non comparable specializations of the "kind of" relation can be combined to create a common subelement. The viewpoint V_1 = *(Fatty acid, koCh)* provides 4 (strict) predecessors of *Fatty acid*. The viewpoint V_2 = *(Fatty acid, koSt)* provides 2. The viewpoint V_3 = *(Fatty acid, koO)* also provides 2. Thus the number of elements obtained by double inheritance is $card(V1) \times card(V2) + card(V1) \times card(V3) + card(V2) \times card(V3) = 8 + 8 + 4 = 20$ and the number of elements obtained by triple inheritance is $card(V1) \times card(V2) \times card(V3) = 16$. That would lead to 36 common subelements to represent.

Our choice is to simplify the graphical user interface by not visualizing elements that are obtained by multiple inheritance, as they are simply conjunctions of other viewpoints. We thus obtain ontologies that have tree structures and may easily be handled in a browser, as proposed in Figure 4 for instance.

3.2 Clarifying the Semantics of Preferences

Furthur to previous studies [1, 2], expressing preferences using fuzzy sets [12] in an unambiguous way implies that the definition domains of these fuzzy sets were exhaustive and composed of exclusive elements. Both properties are characteristics of partitions. The scope of this section is to extend the expression of

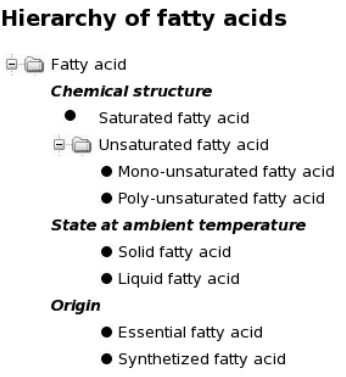


Fig. 4. Viewpoint ontologies without conjunction viewpoints have tree structures that can easily be handled in a browser

preferences to the case of a viewpoint ontology (thus handling partitions) where conjunction viewpoints are not represented (thus simplifying user interface as seen in Section 3.1). We propose a two-step method:

1. Intra-viewpoint preferences. After choosing an element of interest *elt* in the ontology (e.g. *Fatty acid*), the user can visualize its sub-elements through the different viewpoints and thus, within a partition of the chosen element, indicate the querying preferences by ordering the elements that compose the partition. This ordering is computed as a fuzzy set [12]. The left part of Figure 5 gives an example of intra-viewpoint preferences. They are computed as the fuzzy set *1/Liquid at ambient temperature fatty acid + 0.5/Solid at ambient temperature fatty acid*.

2. Inter-viewpoint preferences. If preferences have been defined on several viewpoints, the user can specify an order of importance between these viewpoints. This ordering is computed as weights associated with the selected viewpoints. The right part of Figure 5 gives an example of inter-viewpoint preferences. They are computed as weights 3, 1 and 1 respectively attributed to viewpoints *(Fatty acid, koCh)*, *(Fatty acid, koSt)* and *(Fatty acid, koO)*.

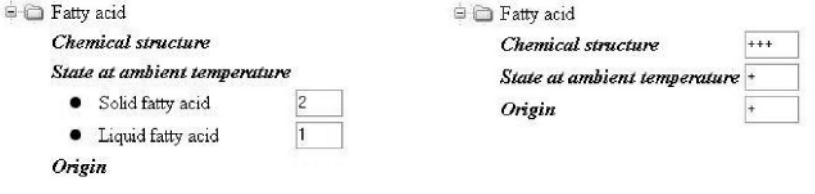


Fig. 5. Intra and inter-viewpoint preferences

Definition 4. *An expression of preferences on a viewpoint ontology is a set* $\{< V_1, w_1, F_1 >, \ldots, < V_n, w_n, F_n >\}$*, where* $V_1, \ldots, V_n$ *are viewpoints on a given element elt,* $w_1, \ldots, w_n$ *are weights respectively asociated with these viewpoints and* $F_1, \ldots, F_n$ *are fuzzy sets respectively defined on partitions of elt in the views induced by* $V_1, \ldots, V_n$*.*

Example 7. *An example of preferences expressed on the* Fatty acid *viewpoint ontology is given by:*

{<(Fatty acid, koCh), 3, 0.3/Saturated fatty acid + 0.7/Mono-unsaturated fatty acid + 1/Poly-unsaturated fatty acid>, <(Fatty acid, koSt), 1, 1/Liquid at ambient temperature fatty acid + 0.5/Solid at ambient temperature fatty acid>, <(Fatty acid, koO), 1, 1/Essential fatty acid + 0.5/Synthetized fatty acid>}.

Remark 3. *The agregation of the preferences criteria to order the data that are being queried according to their relevance is not discussed here (see e.g. [13]).*

4 Conclusion

This paper has proposed a definition of a viewpoint ontology that relies on specializations of the "kind of" relation. A viewpoint corresponds to a partition of an element of the ontology using a given specialization of the "kind of" relation. Partitions are exploited to clarify the semantics of preferences expressed on an ontology, by providing exhaustive domains of exclusive values, used as definition domains of preference fuzzy sets. Moreover within a given viewpoint there is no multiple inheritance, the latter being the result of a conjunction between several viewpoints. This property is used to propose a simplification of user interface. Finally we propose the use of weights – called inter-viewpoint preferences – to specify levels of importance between viewpoints in which preferences have been defined as fuzzy sets – called intra-viewpoint preferences.

The proposed methodology is based on the use of specializations of the "kind of" relation that produce a partition of possible cases. This is not the case for all of the possible specializations of the "kind of" relation. A future work will consist in specifying the conditions that lead to the obtention of partitions.

References

1. Thomopoulos, R.: Représentation et interrogation élargie de données imprécises et faiblement structurées. PhD thesis, INA P-G, Paris, France (2003)
2. Thomopoulos, R., Buche, P., Haemmerlé, O.: Different kinds of comparisons between fuzzy conceptual graphs. In: ICCS'2003, Dresden, Springer (2003) 54–68
3. Minsky, M.: A framework for representing knowledge. In Winston, P., ed.: The Psychology of Computer Vision. McGraw Hill, New York (1975)
4. Bobrow, D., Winograd, T.: An overview of KRL, a knowledge representation language. Cognitive Science **1** (1977) 3–45
5. Stefik, M., Bobrow, D.: Object-oriented programming : Themes and variations. The A.I. Magazine **6** (1985) 40–62

6. Carre, B.: Méthodologie orientée objet pour la représentation des connaissances. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, France (1989)
7. Davis, H.: VIEWS: Multiple perspectives and structured objects in a knowledge representation language. Bachelor and Master of Science Thesis, MIT (1987)
8. Marino, O.: Raisonnement classificatoire dans une représentation objets multi-points de vue. PhD thesis, Université Grenoble 1, France (1993)
9. Ribière, M., Matta, N.: Virtual enterprise and corporate memory. In: ECAI'98 Workshop on Building, Maintaining and Using Organizational Memories. (1998)
10. Brachman, R., Schmolze, J.: An overview of the KL-ONE knowledge representation system. Cognitive Science **9** (1985) 171–216
11. Sowa, J.: Conceptual structures - Information processing in Mind and Machine. Addison-Welsey (1984)
12. Zadeh, L.: Fuzzy sets. Information and Control **8** (1965) 338–353
13. Dubois, D., Prade, H.: A review of fuzzy sets aggregation connectives. Information Sciences (1985) 85–121

Architecting Ontology for Scalability and Versatility

Gang Zhao and Robert Meersman

Semantic Technology and Applications Research Laboratory,
Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium
{gang.zhao, robert.meersman}@vub.ac.be

Abstract. This paper discusses methodological strategies for architecting ontologies. The development context is an EC IST project, aimed at the use of ontology to help detect and prevent financial frauds. The ontology engineering in this context faces challenges of the scalability of ontology, its applicability in multiple applications, its semantic consistency and comprehensiveness, among others. The paper discusses architecting strategies: layering and modularizing with design patterns, used in producing topical ontologies of fraud and VAT.

1 Introduction

This paper discusses the strategies of architecting ontologies in the project, FF POIROT (IST2001-38248). The project seeks to build ontologies and deploy them to enable intelligent functionalities in the information system for detecting and preventing financial frauds (www.ffpoirot.org). The ontology development is based on the ontology conceptualization framework, *DOGMA* (*Developing Ontology Guided Mediation Agent*) [7], [8]. It is an development activity in the knowledge engineering lifecycle, preceded by activities such as *knowledge resource collection*, *scoping*, *knowledge breakdown* and *elaboration*, in the *Application Knowledge Engineering Methodology* (*AKEM*) [13].

1.1 Approaches and Perspectives in Modeling Ontology

Different approaches and perspectives often cause contention in ontology engineering. They reflect different sets of needs and assumptions from the stakeholder and are justifiable in aspects of problem-oriented ontology modeling. This section summarizes those that are encountered in the ontology engineering in FF POIROT.

1.1.1 Conceptual vs. Formal Modeling

Knowledge engineering often faces the contention of purposes in either capturing domain expertise as it is with understatements, ambiguity, multi-perspective, fuzziness and inconsistency or representing its application as mathematical objects and processes, consistent, unambiguous, and rigorous for a dedicated use in a given problem-solving paradigm. We refer to the former as 'conceptual modeling' in the recognition of human cognitive potential and the latter 'formal' for its axiomatic emphasis and computational requirements.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1605 – 1614, 2005.

The contention is not only an issue of representational languages: either an expressive semiotics for conceptualization or rigorous formalism in the support of inference and computation. It is also an issue of modeling approaches to knowledge. The former emphasizes a comprehensive coverage whereas the latter takes a reductionistic stance. A similar contrast of approaches exists in linguistics. The mathematical approach to linguistic description promises rigorous neat models of language structure, but fails to deliver one required for comprehensive linguistic description. The result of a descriptive approach, on the other hand, lacks rigorousness and formalization expected in natural sciences and software processing. Similar to natural languages, knowledge as a semiotic system is not a mathematical system, though aspects and perspectives of relations and operations can be formalized. It is hardly doubtful that a computational system based on a large-scale knowledge model requires both conceptual and formal modeling for its production.

1.1.2 Descriptive, Prescriptive or Operative Purposes

Ontology modeling can be performed with three purposes in mind: descriptive, prescriptive and operative. The descriptive purpose seeks to capture and document domain expertise of various dimensions and perspectives: a representation of plausible worlds. The comprehensiveness is an essential goal. The prescriptive purpose aims at a model of standards, such as that of business processes, meta data of web pages or web service descriptions. It produces a commonly accepted unambiguous semantic protocol for interoperation and communication. Ontology modeling can be dedicated to a particular operative purpose to fulfill an intelligent functionality by a particular inferencing or computational paradigm. The knowledge model in this case is application-specific and applied under the close world assumption. The validity, decidability and computability are a few of properties of the model to consider. These three purposes are not necessarily mutually exclusive in a knowledge development lifecycle model, but they provide different methodological motivations.

1.2 Perspectives in FF POIROT

The perspective of ontology development in FF POIROT is essentially an application semantics perspective. It requires a model of ontological objects, relationships, constraints and axioms. The ontology model is not used in data-intensive operations with database management systems, but for specifying fraud evidences and patterns. A fraud evidence model consists of entities, processes and inferential links. The approach to it is both descriptive to capture the knowledge and operative to deploy it to automate some tasks of fraud detection and prevention. The model created during the knowledge elicitation is conceptual but the model to deploy for an operative purpose is formal. The ontology process is therefore characterized with two main contentions: descriptive vs. operative purposes, conceptual vs. formal conceptualization. The DOGMA ontology representation framework and AKEM methodology are used to structure and integrate various needs, perspectives and approaches.

2 Challenges in Ontology Scalability and Versatility

Some challenges in ontology development come from multiple approaches, perspectives and applications. Others come from the development management. This section

discusses three main challenges in the process and management of ontology modeling: design scalability, model versatility and integration of different approaches.

2.1 Design Scalability

The design scalability is concerned with the size and complication of ontology models. The size of the model, measured by the number of vertices and arcs, is determined by the requirement of its comprehensive coverage. The complication of the model is the complexity of logical relationship among the vertices, arcs and axioms. It grows in proportion to the size. The formal requirement of logical soundness and validity of the model has a big impact on the size-complication relation. The question is whether the model can be extended to cover adequately the subject domain before it reaches a saturation point when the effect of amendment is unpredictable conceptually due to a web of subtle logical relationships in the model. In other words, the design scalability can be restricted when the ontology needs to be consistent logically to support valid reasoning while a required comprehensive coverage introduces ambiguity, inconsistency and invalid inference. In contrast with the database conceptual model, the universe of discourse of the ontology for fraud detection and prevention is not so clear-cut. It turns out to be a combination of conceptions from multiple subject domains: finance, economics, law, criminology, sociology information technology and common sense. Its contents are not homogeneous as in database models or logical model in formal inference systems. In addition, a large set of object and relationship types are involved as well as a large population of their instances. Since its application is for the recognition, interpretation and reasoning about fraud evidence, the ontology also consists of complex networks of relationship. The design scalability is also concerned with effective consensus building in creating the ontology model in a team development context. In FF POIROT, the team is not only multi-disciplinary but also geographically distributed. The adequate documentation and communication of rich semantics is essential to reach and maintain consensus.

In summary, the challenge is two fold. One is how to compromise between the necessity for a conceptual model of comprehensive coverage over a heterogeneous semantic space and the requirement of formal semiotic system for valid and consistent inference. The other is to keep track of the development footprints to make the model comprehensible at different time and location and by different teams.

2.2 Model Versatility

As the design scalability is an issue of productivity in knowledge engineering, the model versatility is one of efficiency. The model versatility is concerned with the question of whether the ontology model can be adapted (augmented or constrained) to variable perspectives of representation and (re-)used for different application specific processing tasks, functionalities or systems. It is not the intention to create ontology usable by any application, but by a family or product line of knowledge systems. In the current context, the ontology application can vary from monitoring in (un-)structured data, to decision support or knowledge management in fraud detection, prevention and prosecution. The model versatility of ontology is comparable to the creativity of natural languages. Words and expressions are reused in different linguistic contexts to convey open-ended possibilities of unique meanings and nuances. In short, the challenge is thus how to facilitate and maximize ontology reuse.

3 Architecting Ontology

Architecting ontology is one of the development activities in AKEM, which features five sets of methodological strategies: ontology scoping, extraction, abstraction, organization and deployment strategies [11]. The architectural design is part of the ontology organization strategies. In this respect, the ontology for fraud detection and prevention is treated as a semiotic system, similar to computational, social and organizational systems. It is assumed to be a heterogeneous possibly inconsistent system fulfilling the multiple (possibly contradictory) needs, rather than a homogeneous single-dimensional logical system. The layered approach is adopted in architecting such a system to manage different viewpoints of system requirements and the complexity of the system architecture. The framework of layered architecture also serves a methodology to organize and manage the role of developers, task of development and work results.

3.1 DOGMA Approach to Ontology

The scalability and versatility are not necessarily complementary requirements that can be met at the same time and in the single-dimensional approach. In recognition of their necessity and conflict, the DOGMA approach to ontology is adopted to provide the first cut of the layered architecture paradigm. It stratifies ontology on two representational layers to disentangle the needs of design scalability with flexible representation and rigorous models for versatile applications. It institutionalizes two viewpoints of ontology conceptualization: lexons of the plausible conceptual world and application commitments to lexons in the context of specific tasks, systems or solutions.

A lexon is context-specific conceptual relationship. It is a 5-tuple $l = <\gamma, t_i, r_{i\text{-}j}, t_j, r_{j\text{-}i}>$. In this formalization $\gamma \in \Gamma$ is the context identifier; $t_i, t_j \in T$ are terms referring to entities in the conceptual relationship; $r_{i\text{-}j}, r_{j\text{-}i} \in R$ are labels of roles in a conceptual relationship and Γ, T and R are strings over an alphabet A^+. The context identifier γ refers to the ideational context in which the terms t_i, t_j and roles $r_{i\text{-}j}$, $r_{j\text{-}i}$ are meaningful. It instills the lexon with particular semantic contents. In practice it consists of one or more knowledge resources such as documents and figures which have been established, communicated, documented and agreed upon by a community of ontology engineers. Intuitively, a lexon $l = <\gamma, t_i, r_{i\text{-}j}, t_j, r_{j\text{-}i}>$ may be interpreted as follows: in the ideational context γ the head term t_i plays the role of $r_{i\text{-}j}$ with respect to the term t_j. Conversely, the term t_j plays the co-role $r_{j\text{-}i}$ with respect to the head term t_i. Here the term refers to a semantic type rather than a token. The lexon, therefore, embodies relationship or fact types instead of instances in an application domain. It is a declarative a statement about relationship, leaving out axiomatic constraints, well-formedness restrictions and inferential implications. It has a flat structure and makes a similar argument against premature conceptual packaging as that against the use of attributes in entity modeling for databases. No assumption is made about the backbone structure of ontology being a tree structure.

If lexons address the 'what' in the subject domain, commitments depict how the what is contextualized and rendered in particular applications with well-formed reference and valid proofs. The interpretation function together with axioms, constraints and derivation rules are left out of the lexon specification. They are handled as commitments and in commitment processing. A commitment is a particular interpretation

of one or more lexons in the light of specific application contexts. Structurally it consists of a subset of lexons from the ontology base. They form a context-specific, application-specific semantic network embodying the underlying application logic. In a commitment, the terms and roles of lexons are qualified with axiomatic constraints, specific value constraints and instantiations relevant to a given application context of tasks or systems. It is semantically consistent and unambiguous and well structured in the application context. If lexons are likened to global variables and structures, the commitments can be seen to instantiate a subset of global variables and structures. Their validity is local: in a given semantic network created for a particular application task [2], [12]. The lexons and its commitments puts consequently the scalability and versatility requirements in orthogonal dimensions, advocating achieving the requirements in different contexts of consideration to enable focused development at a time, reuse and transformation of the deliverables along the two viewpoints.

Table 1. Examples of lexons in the application ontology

Context	Term	Role	Term	Role
Decree 58 A1	Definition	ReferTo	Regulation	ReferredToBy
IOSCO-IRRPG	Disclaimer	Exclude	Community	ExcludedBy
IOSCO-IRRPG	Disseminate_Information	ByMeansOf	Propaganda	
IOSCO-IRRPG	FinancialInformation	CharacterisedBy	Truth	Characterise
Decree 58 A1	FinancialInstrument	SubtypeOf	Product	SupertypeOf
Decree 58 A94	FinancialInstrument	CharacterisedBy	Definition	Characterise
Decree 58 A2	Form_BusinessInstitution	At	Place	

3.1.1 Design Scalability with Lexons

The lexon base is an unstructured collection of lexons. It is not necessarily logically homogeneous coherent collection of semantic relations. In semiotic terms, lexons, as symbolic resources, carry generic ideational meanings. Their interpretation can be logically inconsistent and contradictory with each other. Their denotation can overlap partially or wholly, depending on various possible perspectives and dimensions necessary. The scalability of the ontology base lies in two facts. Firstly it imposes a simple logical structure on the collection of lexons as a bag of lexons. It enlists the semantic potentialities and possibilities of application semantics, leaving its particular usage and associated well-formed semantics to commitments. Similar to natural languages, the lexical semiotic resources only become specific and constrained in their denotation in the syntagmatic contexts of sentences and texts. The development of lexons is thus a monotonic and accumulative process of the ontology base expansion. This facilitates the descriptive modeling of non-trivial and multi-disciplinary domains and the achievement of a comprehensive coverage. Secondly, the ontology base is a representation of commonality (not only generality) of application semantics in a family of applications and systems. Conceptual specification is factored out to individual local scopes of consideration in the form of commitment specification, in which a subset of lexons are grounded for a specific purpose, in a single perspective and dimension, consistently and unambiguously in view of application semantics. Since the lexons are used to capture the semantic essence common in diversified contexts, the underspecifying lexons facilitates reaching consensus on the essential among stakeholders.

The ontology base thus constitutes a system of options from which commitments are created with specific scopes of consideration with no need for complete homogeneous instantiation of the whole ontology. The global potential and local instantiation is good service to the design scalability.

3.1.2 Under-Specification of Lexons and Model Versatility

The natural language, the most effective medium of knowledge representation [4], uses the mechanism of semantic under-specification to enable the expression of a maximal number of themes with a minimal number of symbolic resources. Lexons are a conceptual construct to capture underspecified concepts and relations for the purpose of reusability and flexibility. *Discount* as a term is under-specified in its definition: 'an abatement or reduction made from the gross amount or value of anything' [3]. What is not specified is whether it is trade or cash discount or for advance payment or customer loyalty, or under what conditions, in what percentage, where, when and to who. *Notify* as a role in lexon can be qualified with specific values of Boolean, time, aspect or cardinality.

Particular perspectives of an application may need the commitment not only making lexons more specific, but also more abstract. The latter is achievable through lexon reification by 'masking' parts of lexons. In the following, γ is the context. t_1 and t_2 are the terms and r_1 and r_2 the roles of lexons.

$$<\gamma, t_1, r_1, t_2, \sim> \text{ or } <\gamma, t_1, \sim, t_2, r_2>$$
$$<\gamma, t_1, r_1, \sim, \sim> \text{ or } <\gamma, \sim, \sim, t_2, r_2>$$
$$<\gamma, \sim, r_1, \sim, \sim> \text{ or } <\gamma, \sim, \sim, \sim, r_2>$$

For example, when lexons express business processes, their masking can be used to represent views of the process such as action, data, organizational [12]. Representationally, the reification of lexons is anchored on the role signature of the lexon. It is an abstraction of an abstraction. While lexons signify abstract fact types, their reification encapsulates a type of lexons, similar to *interface* as compared with *class* in object-oriented programming languages, such as Java and C#. It is found frequently in natural languages in the form of nominalizations in formal texts such as legislatures. The number of elements masked indicates explicitly the granularity of abstraction.

3.2 Topical Ontology to Layer Ontologies

The lexon base and commitment layer serve to divide the attention over the problem space and solution space. The idea of topical ontology as distinguished from the application ontology, upper or foundational ontology is to introduce another viewpoint in the architecture design. The purpose is to identify the lexons necessary to describe the knowledge structure, assumptions, frameworks, and principles about the central theme of an application domain, such as intellectual property rights [1], value added tax [5]. The lexons are not application specific and typically involve multiple subject domains. They embody the ideational perspectives, the knowledge structure and principles in the expertise about the theme.

The experience in FF POIROT illustrates the motivation of the topical ontology. Its financial fraud ontology is initially built from real-life cases of fraudulent solicitation of financial instruments and relevant legislations by the techniques of lexon *extraction*, *abstraction* and *organization* in AKEM [11]. Table 1 shows a few examples of

900 lexons. An application ontology is extracted and abstracted bottom-up. The need is felt to structure these lexons to facilitate further modeling. The case-specific ontology modeling does not reveal the conceptualization underlying the knowledge structure and principles of the theme. In the emphasis on the organization of ontology an experiment is conducted to organize the lexons with upper-level and domain ontologies to see if the resulting model provides a conceptual structure to represent the knowledge structure and principles about frauds. Relevant to the coverage of the lexons, 186 concepts are imported from SUMO base and domain ontologies [9], and 123 lexons specified for alignments between the SUMO and the application ontology. The conclusion is that the integrated ontology model does not provide a systematic conceptual framework to describe the knowledge structure and principles about a particular topic. Key conceptions special to the theme but common vocabulary of the community are not in the application ontology or in the domain and base ontologies.

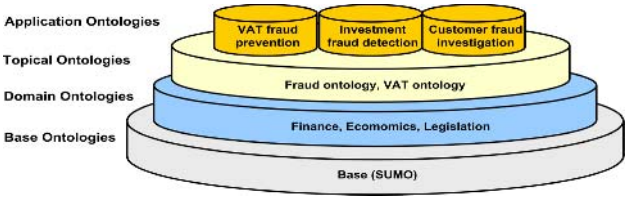


Fig. 1. Layered architecture of ontology

The general conceptions in the upper-ontology assume denotations and associations specific to the topic about frauds. Architecturally, there is a missing layer between application ontologies and upper level/domain ontologies, hence the topical ontology. The layer is not simply a collection of concepts from domain or base ontologies as SUMO MILO, but also is expected to reflect the conceptual framework essential with the topic. In short, the topical ontology captures the knowledge structure of the domain expert on a particular set of themes with specific conceptualization and perspectives of the subject matter (not the application).

3.3 Modularizing Topical Ontology

Modules of the topical ontology componentizes the subsystems into packages of attention and development. Its motivation is largely two fold. One is the use of components and packages to manage different conceptualization viewpoints similar to software system architecting [10]. The other is the use of divide-conquer strategies to manage the complexity of conceptual modeling. Maximizing the conceptual homogeneity within components and packages facilitate both the design scalability and versatility.

The architecture of a topical ontology can be structured from a central topic. A pattern of architecting topical ontologies is used in creating fraud ontology and VAT ontology in FF POIROT, shown Figure 2. The central theme, *Topical Concept*, is structured by semantic *Configuration* made up of semantic *Component*. The *Configuration* instantiates the abstract *Schema*. The *Topical Concept* can be categorized by *Type* from the perspective of or by the criterion of the *Component* of its semantic *Configuration*. The *Topical Concept* is involved in the *Activity* or *Process*. It is qualified by the *Quality*.

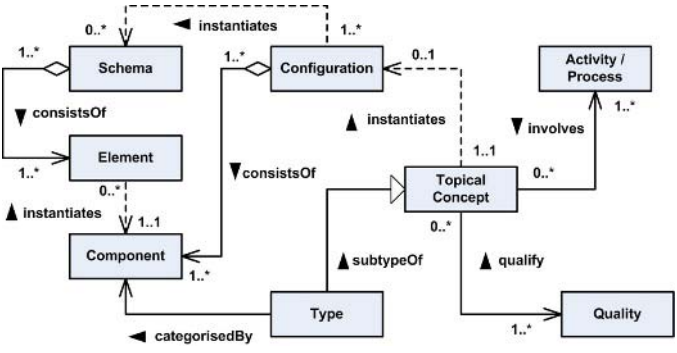


Fig. 2. A design pattern in UML for the architecture of topical ontology

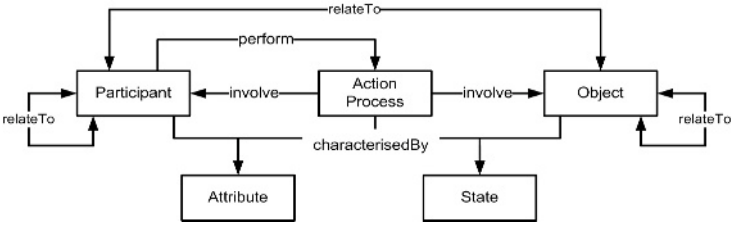


Fig. 3. Schema of a topical concept as an event

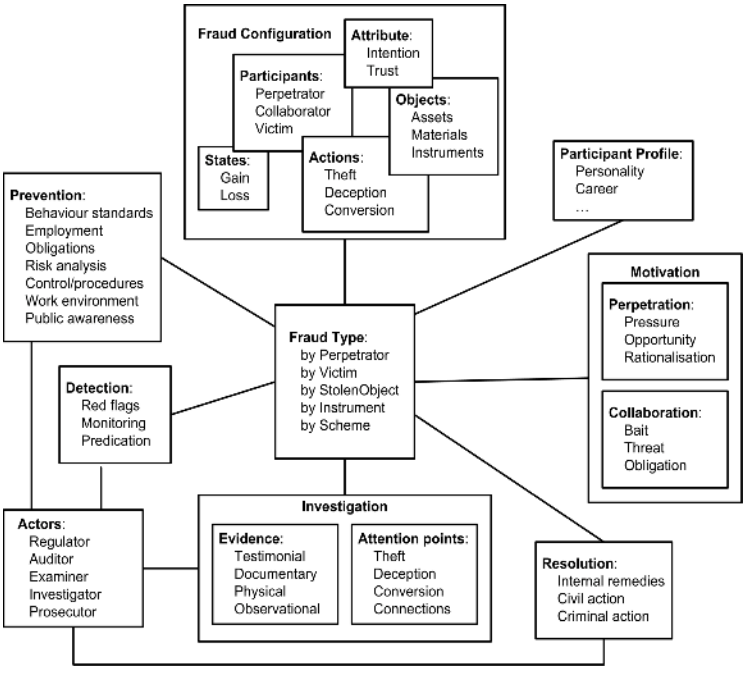


Fig. 4. Architecture of the fraud ontology

The topical concept in the fraud ontology is *fraud*. The abstract schema, for example, is illustrated in Figure 3. It consists of nine abstract schematic entities and relations necessary to depict the concept of fraud as an event.

The configuration of the topical concept thus consists of five main packages concerning *Participant*, *Action*, *Object*, *Attribute* and *State*. Each consists of modules specific to fraud domains, as shown in the box *Fraud Configuration* in Figure 4.

Architecting from the topical concept of *fraud*, activities and properties are packaged as indicated in Figure 5. The current version of the fraud ontology architecture is therefore made up of one package about fraud typology, two packages about qualities: *Participant Profile* and their *Motivation* and 4 packages about the activities in the lifecycle of antifraud activities. Each package consists of sub-packages, thus partitioning the conceptual space and modularizing the ontology base for systematic ways to scale up the ontology model.

4 Conclusion

The team development of a comprehensive ontology of given themes faces three key challenges: design scalability in multiple perspectives with the necessity of both model validity and comprehensiveness and model versatility. A key strategy to manage the multiple perspectives and uses is the principle of system architecture. Similar to the art of systems architecture [7], the strategy is a toolset of heuristics concerning conceptual framework, design patterns and best practices. This paper summarises the study of three heuristic principles: a layering method to manage viewpoints, the principle of topical ontology to manage multiple ontologies and design patterns for modularization. They are based the DOGMA ontology representation framework as architecting tool to manage viewpoints of the conceptual system. As methodological strategies, they are set in the pragmatic knowledge engineering methodology: AKEM for their application and significance in team, methodical engineering lifecycle.

The methodological properties of topical ontology requires further study in practical and comprehensive ontology development aimed at a family of intelligent system applications. The design pattern of organizing ontologies as recommendation is an important methodological instrument to manage team consensus and effectiveness in ontology model. In order for it to be effective methodically, further study in practical engineering context need to be conducted.

Acknowledgement

This study is partially funded by the EU 5th framework program, IST 2001-38248.

References

1. Delgado, J., Gallego, I., Garcia, R., and Gil, R. 2002. An Ontology for Intellectual Property Rights: IPROnto. In Proceedings of the 1st International Semantic Web Conference (ISWC2002), Sardinia, Italy, June 9-12th, 2002.
2. Deray T. , P. Verheyden, Towards a Semantic Integration of Medical Relational Databases by Using Ontologies: a Case Study. On the Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, Lecture Notes in Computer Science, Vol. 2889/2003. Springer-Verlag, Heidelberg (2003) 137 – 150

3. Gove, P. B. (ed) Webster's Third New International Dictionary of the English Language Unabridged, Merriam-Webster Inc, Massachusetts (1986)
4. Iwanska L. , S. Shapiro (eds.): Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language, AAAI Press/MIT Press, Massachusetts (2000)
5. Kerremans, K., Zhao, G.: Topical Ontology of VAT, FF POIROT Deliverable 2.3, 2005.
6. Maier, M., Rechtin, E.: Art of Systems Architecting, CRC Press, Boca Raton (2000)
7. Meersman, R.: Reusing Certain Database Design Principles, Methods and Techniques for Ontology Theory, Construction and Methodology, STARLab Technical Report, Vrije Universiteit Brussel (2000)
8. Meersman, R.: Ontologies and Databases: More than a Fleeting Resemblance?, In D'Atri, A. and Missikoff, M. (eds.): OES/SEO 2001 Workshop, Rome (2001)
9. Niles, I., Pease, A: Towards a Standard Upper Ontology. In Welty, C. and Smith, B. (eds.): Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Ogunquit, Maine (2001)
10. Putman J.: Architecting with RM-ODP, Prentice Hall, New Jersey (2001)
11. Zhao, G.: AKEM: an ontology engineering methodology in FF POIROT, Deliverable 6.8 (2005)
12. Zhao, G., Gao, Y., Meersman, R.: An Ontology-based Approach to Business Modeling, In Proceedings of the International Conference of Knowledge Engineering and Decision Support (2004)
13. Zhao, G., Kingston J., Kerremans K., Coppens F., Verlinden R., Temmerman R. & Meersman R.: Engineering an Ontology of Financial Securities Fraud. In, Meersman R., Tari Z. et al.,(eds.), On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops, LNCS 3292, Springer Verlag (2004) 605 – 620

OWL-Based User Preference and Behavior Routine Ontology for Ubiquitous System

Kim Anh Pham Ngoc, Young-Koo Lee, and Sung-Young Lee

Department of Computer Engineering, Kyung Hee University, Korea
anhpnk@oslab.khu.ac.kr, yklee@khu.ac.kr, sylee@oslab.khu.ac.kr

Abstract. In ubiquitous computing, behavior routine learning is the process of mining the context-aware data to find interesting rules on the user's behavior, while preference learning tries to utilize the user's behavior information to infer user interests, intention and desires. An intelligent environment should be adaptive, i.e. it is should be able to learn the routine and preference of user, then provide user with the suitable service. Developing intelligent ubiquitous environment requires not only good learning algorithms but also appropriate reusable models of user preference and behavior routine, which are not fully covered by current projects. In this paper, we propose a formal and comprehensive ontology-based model of user preference and behavior routine. The implementation of the ontology using OWL[14] enhances the expressiveness, support inference, knowledge reuse and knowledge sharing, which we can not achieve by normal models. The main benefit of this model is the ability to reason over context data to predict what the user wants the system to do. Based on our model, we also present a rule learning mechanism to learn the preference and behavior rules from context data. [1]

1 Introduction

Intelligent ubiquitous computing focuses on merging intelligent and agent-based system with the ubiquitous computing paradigm. An intelligent environment is a space where ordinary human activities mix seamlessly with computation in a way that enhances the functions of both system and user. That means when a user enters a smart-space, the system can recognize who the user is, what he is doing, "guess" what he intends to do, and how he desires the system to assist him. By other words, the system should be able to learn about user preference and behavior routine so that it can provide services to the user seamlessly & invisibly without any explicit user intervention.

There are many methods to learn user preference and user routine. One of them is association rule mining. User preference and routine learning is considered as associating different contexts to each other. These associations are derived as IF-THEN rules, or association rules. Furthermore, we can apply Bayesian net, or Hidden Markov model for preference and routine learning [7][10]. Many prototyping systems and architectures have successfully introduces novel algorithms for learning user routine from GPS location [7]. In those approaches, important places are identified by

[1] This work was supported by MIC Korea. Dr. S.Y.Lee is the corresponding author.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1615 – 1622, 2005.

monitoring the user's travel patterns and learning his frequented locations. Then these important places will be named by user.

However, a formal model to memorize the routines of user has *not* been defined; or by other words, there is *no* formal model for user routine. Similarly, the term "user preference" so far is just related to the user interest for some very specific subjects, such as the web links, music, presentation material, etc, as well as situation independent [10]. *None* of current user preference model fully represents the preferences of user in a ubiquitous environment, where the user interest, desire and intention vary by time and place. The requirement of having a formal model of user preference and behavior routine to use in ubiquitous computing systems is obvious. SOUPA [5] has already defined a user preference model. Nevertheless, this model is specified only for meeting room scenario, and rather simple to be considered as a *formal* and *general* user preference model.

Recently, many systems model context data using ontology and semantic web technique [6][12]. Web Ontology Language OWL [14] is preferred due to its ability to represent explicitly semantics associated with the knowledge, and to provide reasoning capabilities used by intelligent systems and agents to infer useful contexts. Our CAMUS middleware follows this ontology-based modeling approach. We have already defined and used OWL ontologies for basic entities in context-aware systems including agent, time, location, device and environment, as well as for domain data representing [1].

In this paper, to address the issue of user preference and behavior routine formal modeling, we propose additional *OWL-based user preference ontology* includes modular component vocabularies to represent user beliefs, desires, and intentions related to different times and places, together with the *behavior routine ontology* which is a sequence of location with the expected interval for each location, and allows developer to express the recurrence of the routine.

The rest of this paper is organized as follows. In section 2, we describe the user preference and behavior routine model and its ontological structure. Section 3 gives a detailed description of our idea by discussing the learning and reasoning mechanism with the support of ontology and OWL. We conclude our paper with a summary and outlook in section 4.

2 Spatio-Temporal Ontology of User Preference and User Routine

2.1 Using OWL Ontologies for Formal Context Modeling

Within the domain of knowledge representation, the term ontology refers to the formal, explicit description of concepts, which are often conceived as a set of entities, relations, instances, functions, and axioms, leading to shared and common understanding that can be communicated between people and application systems [2]. Traditionally, ontologies are only used to describe domains (as mentioned above) but in W3C's OWL (web ontology language) [14], the horizon of ontology has been broadened to include instance data as well.

There are several potential advantages for developing context models based on Semantic Web Ontology, such as its e*xpressiveness*, the capability of *Knowledge Sharing* and *Knowledge Reuse*; the support to various existing *logic inference* mechanisms, and lastly, its e*xtensibility.*

2.2 Spatio-Temporal Ontology of User Preference (STOUP)

In many personalized e-applications, the preference model is merely a strict partial order of a set of attributes, expressing "for attribute A, value *y* is better than *x*". However, in a ubiquitous system, not only the pure preference of users, but also the interest, desire and intention of the users should be considered. Besides, situation also plays an important role, i.e., user preference alternates from time to time and from place to place. A formal user preference model should cover all these aspects.

Our user preference and routine model is an additional part of an existing context model for ubiquitous system, Contel, which is already defined by our research group, and currently used in CAMUS [1]. In Contel, all the entities in a context-aware system are categorized into agents, devices, environment, location and time. These categories consist of following main classes (or concepts): Agent, Activity, Devices, Environment, Location Description, Place, Time (Time Interval and Time Instant) and Event. There are also many auxiliary classes, generalized and specialized classes to enrich the semantic capability.

To represent the user preference model, we add a *Preference* class, which has relationship with all main classes, and its subclasses as illustrated in Fig.1.

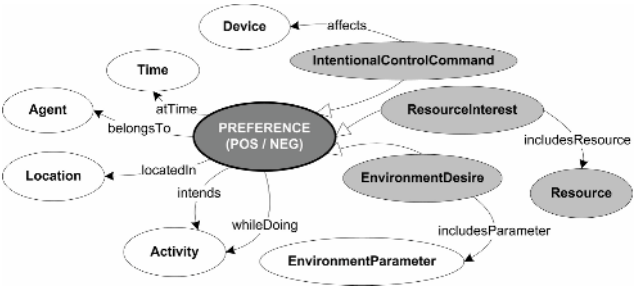


Fig. 1. STOUP structure. Preference class and its subclasses are new classes added to Contel.

An agent (which is a user, group or organization) can have *POS* or *NEG Preference*, which represents the like and dislike, or the best choice and the worst choice. Each preference is related to a certain time and place. Preference also depends on what the agent is doing, and/or what it intends to do next. There are 3 sub-classes of Preference: *ResourceInterest*, *EnvironmentDesire* and *IntentionalControlCommand*. *ResourceInterest* represents the interest of agent in some Resources, such as *MusicGenre* or *TVChannel*. *EnvironmentDesire* expresses how an agent wants the environment to be, for example the desired temperature or the preferred light. *IntentionalControlCommand* specifies the operations which an agent wants the devices to perform, such as turning on the television or rolling down the curtain.

All the properties of *Preference* class which are related to *Agent*, *Time*, *Location* and *Activity* have minimal cardinality 0. The absence of any specified relationship is understood as "for all kind of this". For example, if a preference is defined without any location, it means that this preference can be applied everywhere. *Preference* class also has the *noAgent* property to make it become a general rule which will be applied whenever there is no user around, such as turning off the light or change the

software program running on computer into stand-by mode. Similarly, the *noActivity* property denotes the idle state of agents.

Another significant property of *Preference* class is the *probability* property, which expresses the importance or the priority of a preference. This property has major influence on system decision making process. For example, if the probability of a user giving "Light.TurnOff" control command before going to bed is 99%, the system can infer that whenever the user is sleeping, the light should be off, i.e. the light intensity should be Dark. Or that the equal probability of user watching Music channel or Movie channel at night makes the system ask the user before selecting one of those channel.

2.3 Spatio-Temporal Ontology of User Routine (STOUR)

There are some requirements for a user routine model in ubiquitous computing systems. First, a routine is recursive. It can be a daily routine, or weekly, or monthly routine, etc. Second, a user routine includes a sequence of user locations, or user activities, each of which has expected time interval, i.e. the average time interval user spends doing an activity at a location. Finally, the routine model should support reasoning, which means it should help predicting the next location, or the intended activity.

Fig. 2 illustrates the structure of our user routine ontology.

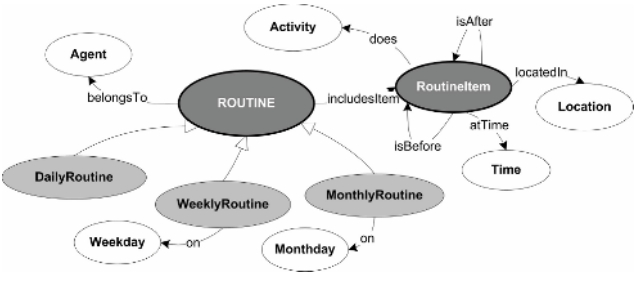


Fig. 2. STOUR structure. Routine, RoutineItem class and Routine's subclasses are new classes added to Contel.

An *Agent* can have one to many *Routine*. There are 3 types of routine: *DailyRoutine*, which is the most common, *WeeklyRoutine* and *MonthlyRoutine*. For *WeeklyRoutine*, some weekdays are included, for example the user goes to gym every Tuesday, Thursday and Saturday. Similarly, *MonthlyRoutine* comes with some days in the month.

Each *Routine* has a sequence of *RoutineItem*. Each item is a state in which user is located in a certain place or is doing a certain work, in an expected time interval. *RoutineItems* go in sequence, so each of them can be before of after another.

Because routine is also an uncertain concept, one *routineProbability* property is attached to each *RoutineItem*.

The following example shows the context ontology that describes a preference and routine of a user named Bilbo using OWL.

```
<POSResourceInterest rdf:ID="BilboMusicInterest">
 <probability>0.9</probability>
 <belongsTo
      rdf:resource="http://ucg.khu.ac.kr/ca#Bilbo"/ >
 <locatedIn
      rdf:resource="http://ucg.khu.ac.kr/cl#LivingRoom"/ >
 <atTime>
  <TimeInterval>
   <startTime>18:00</startTime>
   <endTime>20:00</endTime>
  <TimeInterval>
 </atTime>
<includesResource>
 <MusicResource>
   <resourcePropertyName>MusicGenre
   </resourcePropertyName>
   <resourcePropertyValue>Classical Music
   </resourcePropertyValue>
 </MusicResource>
 </includesResource>
</POSResourceInterest>
```

```
<WeeklyRoutine rdf:ID="BilboWeeklyMeetingRoutine">
 <belongsTo
      rdf:resource="http://ucg.khu.ac.kr/ca#Bilbo"/ >
 <on
      rdf:resource="http://ucg.khu.ac.kr/cr#Tuesday"/ >
 <includesItem
      rdf:resource="http://ucg.khu.ac.kr/br#TuesdayMeeting"/ >
</WeeklyRoutine >
<TuesdayMeeting>
  <routineProbability>0.86</routineProbability>
  <atTime>
  <TimeInterval>
   <startTime>14:00</startTime>
   <endTime>16:00</endTime>
  <TimeInterval>
 </atTime>
 <does rdf:resource="http://ucg.khu.ac.kr/cac#OSLabMeeting"/ >
 <locatedIn rdf:resource="http://ucg.khu.ac.kr/cl#Room505"/ >
</TuesdayMeeting>
```

Fig. 3. example of user preference and routine data in OWL

3 OWL-Based Reasoning and Learning Mechanism

In this session, to show the advantage of ontology-based modeling approach using OWL, we will illustrate the OWL-support reasoning mechanism, and how the system can learn user routine and preference through the control commands of user and the history context database.

3.1 OWL-Support Reasoning Mechanism

3.1.1 Ontology Reasoning Mechanisms

High valued ontologies depend heavily on the availability of well-defined semantics and powerful reasoning modules. The expressive power and the efficiency of reasoning provided by OWL, (the semantics of OWL can be defined via a translation into an expressive Description Logics (DL)), make it an ideal candidate for ontology constructs. The facts gathered from context entities make a factual world in OWL, consisting of individuals and their relationships asserted through binary relations.

Ontology reasoning helps us to find subsumption relationships (between subconcept-superconcept), instance relationships (an individual i is an instance of concept C), and consistency of context knowledge base. In the design phase of formalizing the context entities, OWL reasoning services (such as satisfiability and subsumption) can test whether concepts are non-contradictory and can derive implied relations between concepts.

Let us take an example to see how ontology reasoning can help deducing implied context. In preference ontology, the class PianoMusic is a subClassOf ClassicalMusic. So when knowing that Bilbo is interestedIn ClassicalMusic, and “Hungarian Sonata” is a song which has type PianoMusic, the system can deduce that Bilbo is interestedIn “Hungarian Sonata”.

3.1.2 Context Reasoning Mechanisms

However, many types of contextual information cannot be easily deduced using only ontology inference. In addition to ontology reasoning, we can also use logic inference. A set of rules can be defined to assert additional constraints for context entity instances when certain conditions (represented by a concept term) are met.

Over the concepts and relations defined in our ontologies, we can do a lot of reasoning based on many types of logics, such as description logic, description temporal logic, and spatial logic.

There are many reasoning engines work over OWL format data, such as Racer[13], SWI-Prolog[11], Pellet[9], etc. A list of OWL implementations can be found at [8]. In our current implementation, we use Jena library [4] to handle OWL format context data and ontologies. Therefore we exploit the Jena generic rule reasoner [4] to make inference over our context data.

Following is an example of Jena rule to infer the control command which user intends to give in a certain situation.

[r1: (?user agt:personName "Bilbo"), (?x time:currentTime ?t),
(?user act:currentActivity ?curact), (?curact rdf:type act:WatchingTV),
(?curact act:actionObject ?tv), (?user act:intendedActivity ?intact),
(?intact rdf:type act:GoingToWork) AND 520 <=?t AND ?t <= 530
-> [(?cmd dev:cmdObject ?tv), (?cmd dev:cmdTime ?t)
<- (makeInstance(?user,agt:givesCommand, dev:TVTurnOff, ?cmd)]][2] (1)

with *agt, time, act, rdf, dev* are the aliases for the namespaces of ontologies (agent, time, activity, RDF, device) which are currently used to define data models in our system. Details of those ontologies are described in [1]. This rule is matched when there are OWL markups about user Bilbo watching TV and intending to go to work, then a new instance of class TVTurnOff command is created and the properties cmdObject and cmdTime of that instance is assigned with the current watched television and current time.

However, most developers find building the rules like this the most difficult task in building ubiquitous computing systems, particularly in intelligent environments such as smart homes, where the system has to learn a lot about users preference, behavior, routine, etc. In order to minimize the burden for developer in building context-aware applications, our middleware architecture provide support to learn the inference rules from context data and build reasoning engines using those rules, as described in next section.

3.2 Learning the Rules for Context Reasoning

In this scenario, the user preference is learned through user control commands and responses to the messages from system. User can control the home devices by remote controls, or send command messages through some computer software interfaces. The commands are stored in the history database together with relevant information i.e.

[2] This is a temporal rule. In Jena generic reasoner[3], we can infer about time by continuously updating the currentTime property of Time class with current timestamp. The time value is converted into minutes (or second, or millisecond, depend on the purpose of system).

user location, timestamp, current activity, intended activity (if this information is available), environment state. The tuple {user=Bilbo; place=DiningRoom; timestamp = 2005/04/01 8:40; currentActivity=WatchingTV; intendedActivity= GoingToWork; command= TV.TurnOff} is an example.

Using this kind of information as the training data, the system can learn rules like:

personName=Bilbo; currenttime=[8:40-8:50];
currentActivity=WatchingTV; intendedActivity=GoingToWork
$\Rightarrow$ *command=TV.TurnOff (Utility=0.86)*

The rules can then be converted into suitable format for the reasoning engine which is used. In our prototype system, a rule like (1) is produced.

Moreover, we store the rule into database as a Preference object so that the knowledge about user preference can be shared and reused.

There are many algorithms to learn a rule from example data set. Currently we apply two algorithms for two cases:

- *Rule learning when knowing the desired output*

If the output is defined, we learn the rule from example data set by an approach as in decision tree learning but by following the branch with best score in terms of splitting function.

The *Utility* of a new candidate can be computed using information-theoretic measures like entropy:

$$Utility(r) = entropy(\text{the subset of examples covered by } r)$$

- *Rule learning without knowing the desired output*

If the output is undefined, we can't use any classification algorithm to learn the rule sets. In this case, Apriori association rule mining algorithm [3] is more suitable. Among a large number of learned rules, we select the "right" rules by assigning a utility function to calculate the value of each rule based on confidence and support.

$$Utility(r) = \alpha.Conf(r) + \beta.Sup(r)$$

With *Conf(r)* is the confidence *Sup(r)* is the support of the rule.

The α and β coefficients are related to each other by $\alpha + \beta = 1$, and define the type of rule which is more interested. Normally $\alpha = 1$ and $\beta = 0$, showing that a rule which has high confidence will be chosen even if it rarely happens.

Only the rules with high utility will be selected.

4 Summary and Outlook

In this paper, we have presented a formal Spatio-Temporal Ontology of User Preference and Behavior Routine. We discussed how it can be used for heterogeneous ubiquitous computing environment to support knowledge sharing, reuse, and logical reasoning with the help of Smart Home scenario.

Our next steps include the integration of various machine learning techniques and reasoning engines into our framework. Different machine learning techniques have different input and output format, and different use. By implementing the wrapper for all the techniques and defining a common format for input data, we hope to enable the system developers to handle the machine learning techniques more easily.

References

[1] Anjum S., et al.: Formal Modeling in Context Aware Systems. In proceedings: Workshop on Modeling and Retrieval of Context, CEUR, ISSN 613-0073, Vol-114, 2004.
[2] J. Davies, et al.: Towards the Semantic Web, Ontology-Driven Knowledge Management, John Wiley & Sons. (Nov. 2002)
[3] Jiawei Han,Micheline Kamber. Data Mining: Concepts and Techniques
[4] Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/
[5] Harry C., et al.: SOUPA: Standard Ontology for Ubiquitous and Ubiquitous Applications. In Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004), Boston, MA, August 22-26, 2004.
[6] Harry Chen et al., "Intelligent Agents Meet the Semantic Web in Smart Spaces", Article, IEEE Internet Computing, November 2004
[7] Liao L.,et al.: Learning and Inferring Transportation Routines. In Proceedings: AAAI-0, 2004.
[8] OWL Implementations. http://www.w3.org/2001/sw/WebOnt/impls
[9] Pellet OWL Reasoner. http://www.mindswap.org/2003/pellet/index.shtml
[10] Stefan H., et al.: Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. PKDD 2003
[11] SWI-Prolog/XPCE Semantic Web Library. http://www.swi-prolog.org/packages/semweb. html
[12] T. Gu et. al. An Ontology-based Context Model in Intelligent Environments. In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, pp. 270-275. San Diego, California, USA, January 2004
[13] Volker Haarslev et. al. Querying the Semantic Web with Racer + nRQL. Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04), Ulm, Germany, September 24, 2004.
[14] W3C Web Ontology Working Group: The Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/

Reasoning on Dynamically Built Reasoning Space with Ontology Modules

Fabio Porto

EPFL - Ecole Polytechnique Fédérale de Lausanne,
School of Computer and Communication Sciences,
Database Laboratory, 1015 Lausanne, Switzerland
Fabio.porto@epfl.ch

Abstract. Several applications require reasoning over autonomously developed ontologies. Initially conceived to make explicit the semantics of a certain domain, these ontologies become a powerful tool for supporting business interactions, once heterogeneities have been solved and inconsistencies eliminated. Unfortunately, a stable coherent logical state is hard to maintain in such an environment, due to normal evolution carried out independently over individual ontologies. As a result, reasoning over autonomously developed ontologies has to face with both heterogeneity and inconsistency, in order to assure correct answering. In this paper we study the problem arising in these settings. We propose an incremental reasoning approach based on a virtual reasoning space that is filled with relevant ontology entities as query answering progresses. We show how to compute the set of relevant entities with respect to a user query and present an algorithm for reasoning with dynamically built reasoning spaces.

1 Introduction

Reasoning over distributed and autonomous developed ontologies has to face a number of new challenges. Firstly, current reasoners [1] consider ontology as forming a single logical theory. Unfortunately, both distribution and autonomy adversely contribute to such a view. Therefore in order to use current reasoning software, the set of autonomous developed ontologies must be aligned and integrated into a single consistent one. Secondly, as in the context of database integration [2], and to allow building a single logical theory, definition on different ontologies must be aligned by the use of correspondence expressions. Thirdly, some ontology definitions represent quite a voluminous amount of data. As a result, a naïve solution of transferring all ontologies to a location and then proceed with local reasoning does not scale up. Finally, autonomously defined ontologies may assert contradictory definitions, which some authors classify as conflicts in the integration process. Conflicts identification is, in fact, a tool for fixing correspondence assertions and applying ontology alignment. So, reasoning under this setting should be capable of identifying such conflicts and acting appropriately.

In this paper, we present a new strategy for reasoning over a set of autonomously managed ontologies linked through correspondences. Our approach evaluates an

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1623 – 1628, 2005.

ontology conjunctive query through an incrementally built reasoning space, including relevant distributed ontology entities.

The remaining of this paper is structured as follows. Section 2 gives some introductory definitions. Next, section 3 defines the concepts of an ontology space and modules, and introduces the query model. Section 4 develops the strategy for building a reasoning space. In section 5, we comment on relevant related work. Finally, section 6 gives our conclusions and points to some future work.

2 Preliminaries

In order to simplify the presentation, we assume that ontologies are specified using a single DL language L, defined according to the $\mathcal{SHOIQ}$ dialect. An ontology O is modeled by an interpretation $\mathcal{I}$. The language is built over a signature of distinct sets of concepts, roles and individuals, as described in [3]. In addition, the symbols $\bot$, $\top$ are concepts and, if C,B are concepts, then $\neg$C, (C $\sqcap$ B), (C $\sqcup$ B) are also concepts. The constructors´ semantics are given by the interpretation $\mathcal{I}=(\Delta^{\mathcal{I}}, (\cdot^{\mathcal{I}}))$, where $\Delta^{\mathcal{I}}$ is a non-empty domain and a map function $(\cdot^{\mathcal{I}})$ associates concepts into subsets of $\Delta^{\mathcal{I}}$. Concept names are interpreted as subsets of $\Delta^{\mathcal{I}}$, while complex expressions are interpreted according to the following equations[3]:

$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$; $\bot^{\mathcal{I}} = \varnothing$; $(C \sqcap B)^{\mathcal{I}} = (C^{\mathcal{I}} \cap B^{\mathcal{I}})$; $(C \cup B)^{\mathcal{I}} = (C^{\mathcal{I}} \cup B^{\mathcal{I}})$;

$\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$[1].

Once concepts have been defined, the ontology in enriched by expressing statements about those concepts. Statements in $\mathcal{SHOIQ}$ structure concept hierarchies through subsumption relationship between concepts and classify individuals as of a specific concept or role.

Similarly to the interpretation for concept constructors, an interpretation $\mathcal{I}=(\Delta^{\mathcal{I}}, (\cdot^{\mathcal{I}}))$ satisfies a subsumption statement (C $\sqsubseteq$ B) iff ($C^{\mathcal{I}} \subseteq B^{\mathcal{I}}$) and is represented by $\mathcal{I} :\approx \alpha$, where α = (C $\sqsubseteq$ B).

3 Ontology Space and Modules

We name a set of autonomously specified ontologies over which a hypothetical reasoner could evaluate an ontology query an *ontology space* (*OS*). Giving two ontologies taking part in a *OS*, we say that they intersect if there is a known correspondence assertion associating entities in both ontologies.

The set of entities specified in a ontology together with a set of correspondences expressed with entities in other ontologies define an ontology module (M). The underlying ontology of a module is named its *base ontology*. An ontology entity in a

[1] We intentionally omit role expressions. Interested readers may look for further information at [3].

module is either defined in its *base ontology*, local entity, or added to it by an equivalence correspondence with an external entity, specified in a different ontology. The concept of modules is similar to context in C-OWL [4].

Definition 1: A module is a tuple Mo = <id, D, L, C, Ob, Os>, where id corresponds to a Unique Resource Identifier (URI) for the module, D is the description of the module, either expressed in natural language or by means of an ontology language; L is the ontology language used in Mo[2]; C is a set of correspondences (defined below) associating local entities with entities defined in external modules; O_b is the base ontology and Os is the set of external ontologies to which correspondences with local entities are specified.

Definition 2: A *peer* P=<Mo,QL> models a software component capable of answering ontology queries expressed in *QL* language over an ontology module Mo. A *peer system* is a set PS= $\cup_{i=1,n} P_i$.

Definition 3: An ontology correspondence is a relation in one of the following forms:

- $id_i:C \equiv id_j:D$ (for concept equivalence)
- $id_i:C \subseteq id_j:D$ (for subsumption)
- $id_j:C \supseteq id_j:D$ (for superset)
- $id_i:R \equiv id_j:S$ (for relationship equivalence)
- $id_i:v \equiv id_j:t$ (for instance equivalence)

where (C, R, v) and (D, S, t) are, respectively, local and external entities with respect to a module. *C* is of type concept, *D* is a concept expression of the form $f(t_1,...,t_n)$, where the terms t_i are either concept names or concept expressions and *f* is an n-ary concept builder operator, *R* and *S* are ontology roles, and *v* and *t* are instances [4].

3.1 Interpretation with Ontology Modules

Individual modules have their own interpretation $\mathcal{I}^{id}=(\Delta^{\mathcal{I}d}, (\cdot^{\mathcal{I}d}))$, with *id* being the corresponding module *id*. The semantics of symbols is the one presented in section 2 localized by the domains $\Delta^{\mathcal{I}d}$ and function $(\cdot^{\mathcal{I}d})$. The correspondences in *C* associate external symbols (concepts, instances and roles), with respect to a local ontology O_{id}, with a local interpretation $\mathcal{I}^{id}$, in symbols:

$\mathcal{I}^{ij}(id_j:B)=\mathcal{I}^{i}(f_{Ci}(id_j:B))$, where f_{Ci} is a correspondence defined in ontology module *i*, mapping the external entity $id_j:B$ to an ontology entity in O_i and $\mathcal{I}^{i}$ is its local interpretation.

Finally, we do not rely on the unique name assumption, as in different ontologies the interpretation function may map distinct names to the same domain entity and, conversely, the same name to distinct domain entities. The semantic ambiguity must be solved by asserting correspondences between ontologies.

---

[2] We assume all ontologies expressed in the $\mathcal{SHOIQ}$ dialect.

3.2 Ontology Query Model

We consider boolean DL conjunctive queries, whose terms are assertional statements, similar to [3]. Terms are of the type x:C or ⟨x,y⟩:R, where C is a concept expression and R is a role, and x,y are variables or individual names. An evaluation of a query q corresponds to mapping values or individuals from C or R into the domain $\Delta^{\mathcal{I}d}$ and testing for satisfiability of the assignment.

A query $Q = q_1 \wedge q_2 \wedge \ldots \wedge q_t$, where q_i, $1 \leq i \leq t$, are terms, is satisfied by $\mathcal{I}$, iff $\mathcal{I} := q_i$, for $1 \leq i \leq t$. In particular, considering an ontology space *OS*, we want to build a reasoning space *RS* with interpretation $\mathcal{I}$, and test for satisfiability of $\mathcal{I} := Q$.

4 Reasoning Space

We use the term *reasoning space* (RS) to denote a virtual ontology that is dynamically built to answer an ontology query over an ontology space. A reasoning space includes the base ontology and relevant entities gathered from external ontologies. Entities of a reasoning space share the same ontology language and form a single ontology.

Definition 4: A *Reasoning space* RS is defined as: RS $\subseteq$ {OS $\cup$ C}, where OS is an ontology space and C is the set of correspondences associating elements in OS.

Definition 5: We also define a reasoning space *mapping function f(Q,RS,OS):RS'* that given: a ontology query *Q*, a reasoning space *RS* and a ontology space *OS*, produces a new *reasoning space RS'*.

4.1 Finding Relevant Entities on the Ontology Space

As discussed above, the mapping function identifies relevant entities on the ontology space to be considered in extending the *reasoning space*.
Identifying relevant entities is achieved in two steps: in the first step, we check for relevant correspondences in the current reasoning space; in the second step, relevant entities from external modules are revealed.

Theorem 1: a relevant ontology entity, with respect to a conjunctive query, holds a non empty intersection with some of the query terms.

The intuition behind Theorem 1 is that if an entity (concept or role) is relevant to a conjunctive query, as defined in section 3.2, then its interpretation can not be disjoint with the union of those from all the query terms.

Thus, relevant entities are revealed by evaluating a new conjunctive query formed by two terms, in which one is a term from the original query and the second one is a candidate relevant entity. If the query is satisfied, then the term is relevant.

There are four main query term types to be analyzed, for which relevant queries are generated:

(a) x:C, where x is a variable and C is a concept expression –queries produced: q_i= x:C $\wedge$x:B_i, $1 \leq i \leq m$, where B_i appears in RS as: a:B_i; B_i or C $\sqsubseteq^*$ B_i, for

some value *a*, * modeling the transitive closure and *m* is the total number of entities in OS;

(b) a:C, where *a* is a value and C is a concept expression - queries produced: q_i= a:C ∧a:B_i, q_i= x:C ∧x:B_i, with C ⊑* B_i;

(c) ⟨x,y⟩:R, where *x,y* are variables and *R* is a role – queries produced: q_i=⟨x,y⟩:R ∧⟨x,y⟩:S_i, where S_i is a role and R ⊑* S_i;

(d) ⟨a,b⟩:R, where *a,b* are values and *R* is a role – queries produced: q_i=⟨a,b⟩:R ∧⟨a,b⟩:S_i, where S_i is a role and R ⊑* S_i;

4.2 Reasoning Space Algorithm

The *reasoningspace* algorithm returns the answer for a reasoning conjunctive query q, submitted to a peer Pi, Figure 1.

```
reasonspace(query Q,ontology Ob,OS,correspondence C) : answer
{       RS´:= {Ob};RS=∅;
        q:= ∪i=1,t qt ;     /* qt terms of query Q */
        answer:={evaluate(q,RS)};
        q:=q – {satisfied(q)};
        While (q ≠∅ and RS ≠ RS') {
          RS=RS´;
          RS´= f(q,RS,OS);
          answer:= answer ∪ {evaluate(q ,RS´)};
          q:=q – {satisfied(q)};   }
        return answer; }
```

Fig. 1. Algorithm:ReasoningSpace

Initially a reasoner evaluates query *q* on the base ontology O_b. If the query is satisfied by the base ontology, then the answer is returned and the algorithm finishes, otherwise a main loop extends the reasoning space. On each loop, query *q* is re-evaluated over the current representation of the reasoningspace. The mapping function identifies relevant entities in the ontology space OS-RS and produces the current RS. Satisfied terms are extracted out from the query that is re-evaluated with the remaining terms. The process terminates when all query terms have been satisfied or there are no more relevant entities in the ontology space that are not in the reasoning space.

The algorithm has an upper bound of *O(nm)*, considering the maximum number of subqueries issued to other peers required to answer an ontology query, in which *n* is the number of ontologies in the ontology space and *m* is the total number of entities.

5 Related Work

An interesting approach [5] proposes a modularization approach where self-contained modules are cross connected through materialized views expressed as conjunctive queries. A procedure for managing updates in an external ontology definition is also

proposed. In [6] a proposal for reasoning on distributed ontologies with correspondences is presented. The goal is to define a theoretical solution for the problem of global subsumption and to propose a P2P implementation that assesses the practical adequacy of the proposal. Their main result is to prove that subsumption between remote ontology entities can be proved using local subsumption relationships and correspondences between relevant entities on both ontologies. This leads to global entailment and offers a solution for the problems we investigate in this paper. The strategy can be seen as based on query rewriting approach, similar to what is done in database integration, with distributed reasoning applied using distributed local tableau. Local inconsistencies are treated as holes[4].

6 Conclusion

In this paper, we presented a strategy for reasoning over a set of autonomously managed ontologies with correspondences defining local interpretations for foreign defined ontology entities. In our approach, a reasoning space is built including relevant ontology entities, with respect to an ontology conjunctive query.

The approach presents solutions to all identified problems but also brings to light new questions. Clearly, a more precise comparison of our approach with other distributed ontology reasoning based on query rewriting [6] is of primordial importance to evaluate the benefits of building a reasoning space. This is in our list of future work. We also plan to implement our approach in a P2P system. Finally, we want to investigate a cost model for expanding the reasoning space. The main intuition is that there are innumerous equivalent paths to follow in exploring the ontology space. A cost model based on previous reasoning tasks and statistics regarding individual ontology entities should certainly contribute to reduce query elapsed-time.

References

1. V. Haarslev and R. M. Oller, “Racer: An OWL Reasoning Agent for the Semantic Web”, In Proc. Int´l Workshop on Applications, Products and Services of Web-based Support Systems, Halifax, Canada, October 13, pages 91–95, 2003.
2. T. Devogele, C. Parent, S. Spaccapietra, “On spatial database integration”, Int´l. J. Geographical Information Science, v(12), n(4), pp. 335-352, 1998.
3. I. Horrocks, S. Tessaris, “Querying the Semantic Web: A Formal Approach”, Proc. 1st Int’l Semantic Web Conference, Sardinia, Italy, 2002.
4. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini and H. Stuckenschmidt, C-OWL: Contextualizing Ontologies, Proc. 2nd Int´l. Semantic Web Conference, Sanibel Island, Florida, USA, pp. 164—179, 2003.
5. H. Stuckenschmidt, M. Klein, “Modularization of Ontologies, WonderWeb: Ontology Infrastructure for the semantic Web”, Del 21, V.0.6, May 14,2003 .
6. L.Serafini and A.Tamilin. Distributed reasoning services for multiple ontologies. Technical Report DIT-04-029, University of Trento, 2004.

An Efficient Branch Query Rewriting Algorithm for XML Query Optimization*

Hyoseop Shin[1] and Minsoo Lee[2]

[1] Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea
hsshin@konkuk.ac.kr
[2] Department of Computer Science and Engineering, Ewha Womans University, Seoul, Korea
mlee@ewha.ac.kr

Abstract. XML data usually consists of tree-structured hierarchical data, which affects the storing and searching mechanisms for XML. When storing XML data into databases the hierarchical relationships among XML nodes need to be considered. User's search queries that specify hierarchical relationships among the nodes also require appropriate processing mechanisms. Structural join operations provide a solution to this problem by efficiently computing hierarchical relationships in XML databases based on the node numbering storage scheme. However, in order to process a branch query containing several hierarchical relationships on XML data, many structural joins need to be sequentially carried out and result in a high query execution cost. This paper proposes mechanisms to reduce the cost of processing branch pattern XML queries requiring multiple structural joins. We discuss two approaches for rewriting a query composed of a single branch, and then apply these approaches to general branch queries. The first approach uses the concept of equivalence class relationships among regular path expression queries. The second approach uses a bottom-up approach to reduce the overhead identified in the first scheme. Experimental results show that the proposed schemes can reduce the query execution cost by up to an order of magnitude of the original execution cost.

1 Introduction

In general, XML data consists of tree-structured hierarchical data, which affects the storage and search mechanisms for XML. For example, the XPath query "Paper[Title='XML']/Author/Name" contains as conditions the hierarchical relationships among 'Paper' and 'Title', 'Paper' and 'Author', and 'Author' and 'Name', etc. Therefore, when XML is stored in databases the hierarchical structure among the nodes need to be considered, and user queries also need to be processed with consideration of the hierarchical relationships specified in the query.

When representing XML data as a tree structure, the node numbering scheme can be used to show the hierarchical relationship among the XML elements. Each XML

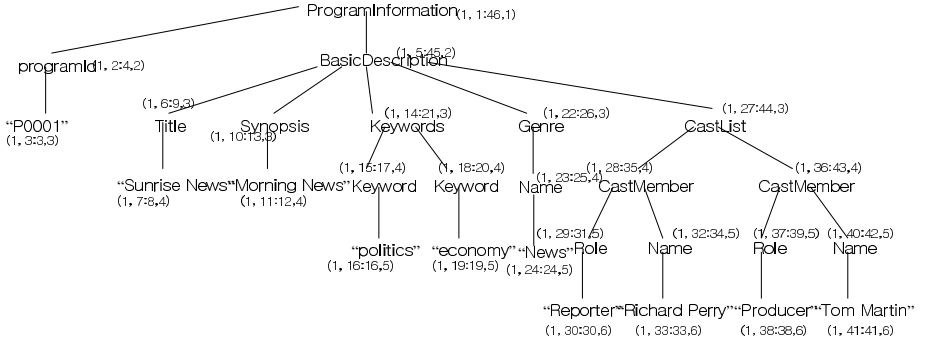
* This work was supported by the faculty research fund of Konkuk University in 2005.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1629 – 1639, 2005.

node can be represented with <docid, begin_pos, end_pos, level> information [5,8](see Figure 1.(b)). The *docid* is the document identifier which is used when more than one documents exist, and *begin_pos* and *end_pos* are the starting and ending offset of the XML nodes within a document, and *level* is the depth of the node starting from the root node. The hierarchical relationship among the nodes can be represented by ancestor-descendant relationships and parent-child relationships. Structural joins [7] can calculate node pairs that satisfy a hierarchical relationship using the node

```
<ProgramTable>
<ProgramInformation>
  <programId>P0001</programId>
  <BasicDescription>
    <Title>Sunrise News</Title>
    <Synopsis>Morning News</Synopsis>
    <Keywords>
      <Keyword>politics</Keyword>
      <Keyword>economy</Keyword>
    </Keywords>
    <Genre>
        <Name>News</Name>
     </Genre>
    <CastList>
      <CastMember>
        <Role>Reporter</Role>
        <Name>Richard Perry</Name>
            <Role>Producer</Role>
        <Name>Richard Perry</Name>
      </CastMember>
    </CastList>
  </BasicDescription>
</ProgramInformation>
<ProgramInformation>
…
</ProgramTable>
```

(a) Content of XML document



(b) Tree representation of XML nodes

Fig. 1. An example XML document

information. Assuming two XML node sets R and S, the structural join that calculates the node pairs that satisfy the ancestor-descendant relationship between R and S can be defined as the following.

$$StructuralJoin(R, S) = \{ <r,s> \mid (r \in R) \wedge (s \in S) \wedge (r.docid = s.docid) \wedge (r.begin < s.begin) \wedge (r.end > s.end) \}$$

For parent-child relationships, the condition r.level=s.level-1 needs to be added.

Structural joins are different from equi-joins in relational databases in terms of the existence of multiple predicates and the nonequi-join characteristic. Most recently, several researches have proposed mechanisms to efficiently process structural joins[1,7] and also indexing methods[11,12].

Although the proposed structural join method can efficiently process the hierarchical relationships among XML elements, the XML query processor still needs to carry out multiple structural joins and experiences a high query execution cost. This is more severe when branch pattern queries containing many hierarchical relationships need to be processed.

In this paper, we discuss two approaches for rewriting a query composed of a single branch, and then apply these approaches to general branch queries. The first approach uses the concept of equivalence class relationships among regular path expression queries. The second approach uses a bottom-up approach to reduce the overhead identified in the first scheme. The proposed XML query rewriting schemes can reduce the number of joins and thus can contribute to the query optimizer for obtaining a more efficient query execution plan.

The organization of the paper is as follows. Section 2 sketches a motivating example of this paper. Section 3 discusses the related research. Section 4 explains the basic data structure called the XIP tree and describes the XML query reduction algorithms. Section 5 shows the experimental results and section 6 gives the conclusion.

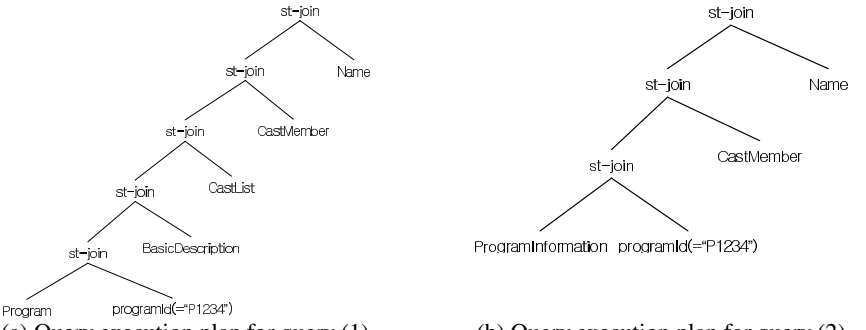
2 Motivating Example

We first explain the concept and the effects of XML query reduction. Assume that we have stored into the database an XML document that includes several 'ProgramInformation' elements and has 'ProgramTable' as the root and has the structure shown in Figure 1-(a). The Figure 1-(b) shows the <docid, (begin_pos, end_pos), level> information of each node based on the node numbering scheme for the XML document shown in Figure 1-(a). A user can give the following two different forms of the same query for "Retrieve all agents of the program of which id is 'P1234'.

(1) //ProgramInformation[@programId ='P1234']/BasicDescription/CastList/CastMember/Name
(2) //ProgramInformation[@programId ='P1234']//CastMember/Name

However, if the query processor takes these different inputs, the resulting query execution plans from these inputs could be totally different. Query (1) needs to perform 5 structural joins (as shown in Figure 2-(a)), while query (2) only needs to perform 3 structural joins (as shown in Figure 2-(b)) and obtain the same query results. Figure 2-(b) definitely is a better query execution plan in terms of the response time. As shown in this example, the XML query that is specified by the user

could be in any arbitrary form while the query processor prefers a reduced number of nodes in the query. In this sense, a query reduction stage is necessary when optimizing user given XML queries.



(a) Query execution plan for query (1) (b) Query execution plan for query (2)

Fig. 2. Different query execution plans for identical XML queries

3 Related Research

Several node numbering schemes for storing large amounts of XML documents in relational databases have been proposed. Zhang et al. [8] suggested to use the offsets of the beginning and ending word of each node to represent the 'containment' relationship between nodes in an XML document. The node numbering scheme proposed by Li et al. [5] provides flexibility and efficiency when updating XML documents. Tatarinov et al. [15] proposed a node numbering scheme to support querying based on orders between XML nodes.

Srivastava et al. [7] proposed efficient algorithms for structural joins that can be used to retrieve XML data that is organized using a node numbering scheme. Chien et al. [1] proposed structural join algorithms for indexed XML documents. Indexing structures for efficient structural joins[11,12] have been also proposed. However, XML queries containing paths or branches may cause several consecutive structural joins in query execution time. Join order selection methods [13] can be used to reduce the cost caused by several joins. Holistic twig joins[16,17] was proposed to process paths or branches at a time instead of stitching several structural joins, but the cost of algorithms are also expected to get higher as the number of nodes involved in XML queries increases. Other approaches use path indices [3,2] to avoid consecutive structural joins in processing XML path queries in relational databases. In this respect, the query reduction algorithms presented in this paper can contribute to reducing the number of structural joins or the cost of holistic twig joins required for path or branch pattern XML queries by reducing the number of relevant nodes in XML queries.

As for query rewriting techniques, Fernandez et al. [9] proposed query pruning and rewriting techniques for regular path expressions using graph schemas which represent partial knowledge about the structures of the semistructured documents. Their query rewriting methods are based on state extents over the graph schema,

while our query reduction is centered into reduction of query itself. Meanwhile, we use the concept of XIP trees which is similar to strong dataguides introduced by Goldman et al. [4]. The difference is that the dataguides summarize the schema of the database for use of query formulation and processing, whereas the XIP tree is created on an XML document instance base and used for query reduction purpose.

4 XML Query Reduction

4.1 Equivalance Classes Among Paths and XIP Trees

In order to perform query reduction on XML queries, we introduce the equivalence class concept between regular path expression XML queries. A group of regular path expressions that are interchangeable among each other (i.e., expressions that yield the same result) is defined to form an equivalence class. To identify regular path expressions that belong to the same equivalence class, a structure called the XML instance path (XIP) tree is used. The XIP tree is dynamically created from the input XML documents and merges structurally similar paths together by holding only a single node in the XIP tree for those child nodes that appear multiple times under the same parent node in the original XML document. Also, the XIP tree differs from the DTD or XML Schema in that the XIP tree is determined dynamically according to input XML documents, while the DTD or XML Schema is pre-defined. Fig. 3 shows the XIP tree generated from the XML document in Fig. 1. Note that nodes like Keyword appear multiple times in Fig. 1 but appear only once in Fig. 3.

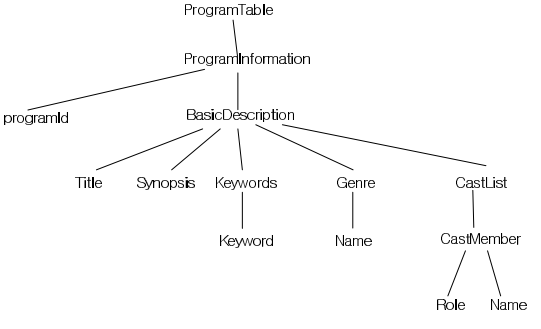


Fig. 3. An example XML Instance Path(XIP) Tree

The size as well as the building cost of the XIP tree can be various depending on the XML document. However, in many cases a XIP tree is expected to be small enough to be kept in main memory like the DTD or XML Schema, where similar analysis has been performed for the dataguide in [4]. The additional overhead of building XIP trees when loading data into a database is also very small.

We now give preliminary definitions to simplify the formal definition of the equivalence class among regular path expressions.

Definition 1. *Complete path expression*
A path expression C is a complete path expression if it is both an absolute path expression (i.e., starts with the '/' axis) and does not include any '//' axis within the path expression. □

Definition 2. *Matching complete path expression*
If a complete path expression C results in a path down the XIP tree where the arbitrary path expression X (which may contain a '//' axis) also represents in the XIP tree, then C is a 'matching complete path expression' of X. □

Definition 3. *Expand*
An *Expand* function takes an arbitrary regular path expression as input and returns the set of all matching complete path expressions of the input path expression. Given an arbitrary path expression X, Expand(X) = {C_1, C_2, … , C_k} where C_i (1 <= i <= k) is a matching complete path expression of X. Every matching complete path expression C of X is always an element of Expand(X). As an example, assume that a regular path expression X is given as '/ProgramInformation//Name', then using the XIP tree in Fig. 3, Expand(X)={/ProgramTable/ProgramInformation/BasicDescription/Genre/Name,
/ProgramTable/ProgramInformation/BasicDescription/CastList/CastMember/Name}.
The function *Expand*() can be implemented by an algorithm that traverses the XIP tree. □

A formal definition of equivalence class among regular path expressions is given below.

Definition 4. *Equivalence class*
A set of path expressions, X_{set} = {X_1, X_2, … , X_m} (where m >= 1), form an Equivalence class iff there exists a set of complete path expressions, C_{set} = {C_1, C_2,…, C_n } (where n >= 1), such that for all X_i (1 <= i <= m) in X_{set}, Expand(X_i) = C_{set}. In other words, for all X_i (1 <= i <= m) in X_{set}, Expand(X_1) = Expand(X_2) = … = Expand(X_i) = … = Expand(X_m) = C_{set}. □

Conceptually, two regular path expressions form an equivalence class if and only if both are expanded into the same set of complete expression paths within a XIP tree. For example, the two regular expressions (1), (2) in section 2 belong to the same equivalence class because both of them are expanded into {/ProgramTable/ProgramInformation[@programId='P1234']/BasicDescription/CastList/CastMember/Name }.

4.2 Path Reduction Algorithm: A Top-Down Approach

The basic idea of path reduction is that a chain of *parent-child* axes can be replaced with an *ancestor-descendant* axis, on condition that the resulting regular path expression belongs to the same equivalence class as the original regular path expression.

Given a regular path expression, possible expressions within the same equivalence class are too many to evaluate each expression one by one in order to find the shortest one (i.e., ${}_{2k-1}C_k$ paths for k-length paths). To simplify the path reduction process, at

first we make use of a greedy algorithm. The algorithm sequentially probes each node of the given regular path expression from left-to-right(i.e., top-down order in XIP tree) and determines whether it can be removed or not. When a node is removed, its preceding axis is replaced with '//' accordingly. A node can be removed only if the resulting regular path expression where the node is removed still belongs to the original equivalence class. If an arbitrary node is removed from the regular expression, the resulting one could represent a path that is not a member of the equivalence class of the original regular path expression. The proposed path reduction algorithm is specified in Fig. 4.

```
TopDownPathReduction()
input: P = A1N1A2N2 … ApNp

Pset ← Expand(P);
initialize head_expr1 ← NULL;
for each i from 1 to p-1 do
        cur_node ← AiNi;
        Ai+1' ← '//';
        tail_expr ← Ai+1'Ni+1 ... ApNp;
        candidate_expr ← head_expr + tail_expr;
        Pset' ← Expand(candidate_expr);
        if Pset ≡ Pset' then                    /* equivalence class */
                Ai+1 ← '//';                    /* remove node */
        end
        else                                    /* non-equivalence class */
                head_expr ← head_expr + AiNi;   /* do not remove node*/
        end
end
head_expr ← head_expr + ApNp;

return head_expr;
```

Fig. 4. The Top-down Path Reduction Algorithm

4.3 Path Reduction Algorithm: A Bottom-Up Approach

The top-down approach previously discussed has several problems. One problem is that the execution time of the algorithm could be significantly large when the path tree is complex, due to the fact that the Expand() function needs to be called every time a node is traversed. Another problem is that an efficient shortest path may not be found because the algorithm considers the nodes in the original path expression in a left-to-right manner. Eliminating nodes closer to the leaves of the XIP tree (i.e., nodes on the right side within a path expression) is more effective than eliminating the ones near the root (i.e., nodes in the left side). We therefore propose an improved algorithm using a bottom-up approach. Assume that there exists a hash table which uses the node names in the path tree as the hash key. The buckets of the hash table contain the ID's of the nodes that have the name in the path tree. The algorithm is shown in Fig. 5.

The algorithm will consider the elimination of the nodes starting from the right-most node in the original path expression whereas the top-down algorithm starts with the left-most node. The algorithm starts by looking up the hash table to find those

path tree nodes that have a matching name to the right-most node in the original path expression. The nodes found are stored in cur_node_list. Each node in the original path is considered in a right-to-left order, each repeating the following steps. While inspecting the nodes in the original path expression from right-to-left, we also traverse the corresponding nodes in the XIP tree in a bottom-up manner to check if any nodes in the original path expression could be transformed into '//', resulting in a shorter path expression. This requires the identification of anchor nodes. When we move one node from right-to-left in the original expression, we also try to follow up the corresponding paths in the XIP tree and keep these paths. If any of the paths that are built so far cannot traverse up the tree corresponding to the left move in the original path expression, we designate such a node in the original path expression as the anchor node. In this case, the anchor node should be maintained in the shortened path expression.

```
BottomUpPathReduction()
input:  P = A_pN_p ... A_2N_2A_1N_1 , XIPtree, HashTable /* stores node ids for element name */
cur_node_list ← NULL;  shortest_path ← NULL;  has_doublebackslash←FALSE;
for each i from 1 to p do
  cur_target_node ← N_i;
  if i=p then  /* prepend to shortest path and finish */
       if i=1 then shortest_path←A_iN_i+shortest_path; end else shortest_path←A_i+shortest_path; end
  end
  else     /* not the final node in the path expression */
       if i=1 then  /* lookup Ids from hash table for current node list */
         cur_node_list ←lookupNodes(HashTable, cur_target_node);
          shortest_path←cur_target_node; end
       else     /* process preceding nodes */
         cur_node_list ← preceding_node_list;
         preceding_target_node← N_i+1;  /* set preceding node of target node */
         preceding_axis← A_i;  preceding_node_list← NULL;  /* preceding nodes of cur_node_list */
         found_anchor_node ← false;   /* reset flag */
         if preceding_axis='//' then  /* retrieve ancestors with same name as preceding_target_node */
                  for each id in cur_node_list do
                     ancestor_set←getAncestorsWithName(XIPtree, id, preceding_target_node);
                      if ancestor_set={} then found_anchor_node←TRUE; end
                      else preceding_node_list←preceding_node_list U ancestor_set; end end end
         else          /* traverse one level up to parent */
                     for each id in cur_node_list do
                        parent_node← getParent(XIPtree, id);
                       if parent_node=preceding_target_node then
                          preceding_node_list←preceding_node_list U parent_node;  end
                else /* found anchor node */  found_anchor_node ← TRUE;  end end end
         if found_anchor_node=TRUE then /* prepend anchor node */
                shortest_path← preceding_target_node+ A_i+shortest_path;
                     has_doublebackslash←FALSE; end
         else                  /* prepend //, or leave as is */
                     if has_doublebackslash=FALSE then
                        shortest_path← '//'+ shortest_path;
                       has_doublebackslash←TRUE; end end
       end
   end
end
   return shortest_path;
end
```

Fig. 5. Algorithm for bottom-up path reduction

4.4 Branch Query Reduction Algorithm

The branch query reduction algorithm uses the previously discussed path reduction algorithms to reduce linear paths. The idea behind the branch query reduction is that the branching node (i.e., ProgramInformation in the query (1)) will divide the branch query into several sub-paths and each sub-path could be recursively processed via the branch query reduction algorithm until the sub-path becomes a linear path. Once the linear path is identified, the previously discussed path reduction algorithms, either top-down or bottom-up, can be applied. The merging of these individual results from the paths can be done by comparing the participating node ids.

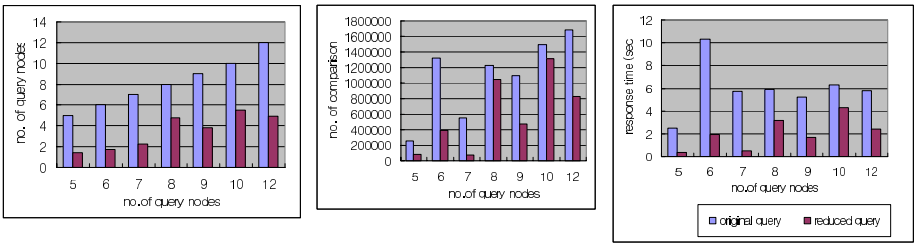
In some cases, the branching node itself could be eliminated and the branch query could be reduced to a more optimal form. But semantic correctness cannot be guaranteed and checking for the deletion of the branching node incurs a significant amount of overhead.

```
BranchQueryReduction()
input:  P = A_p N_p ... A_j[P_j] ... A_2 N_2 A_1 N_1  /* A_j is branching node, P_j is branch path */
        XIPtree /* XIPtree data */, HashTable /* stores node ids for each element name */
result_path ← NULL:
( branching_node, path_set ) ← getBranchingNodeAndPathSet(P);  /* Identify branch node */
if branching_node=NULL then
   result_path = LinearPathReduction(P, XIPtree, HashTable);
                                        /* either top-down or bottom-up path reduction */
end
else
    for each path in path_set
      reduced_path=BranchQueryPathReduction(path, XIPtree, HashTable);
      result_path=Merge(result_path, branching_node, reduced_path);
    end
  end
  return result_path;
  end
```

Fig. 6. Algorithm for branch query reduction

5 Experimental Results

We evaluated the performance of the XML query reduction algorithm presented in section 4 in terms of the overhead of evaluating the XIP trees for query reduction and the benefit in query execution time. About 1G bytes of an XML document generated from the XMark benchmark[6] were populated into the Berkeley DB [10] using the node numbering scheme proposed by Zhang et al. [8]. For an exhaustive performance evaluation of the proposed algorithms, we used as many as 312 XML branch queries that are available from the generated XIP trees. The number of nodes within the queries varied from 5 to 12. We implemented the branch query reduction and the structural join algorithm by Srivastava et.al. [7].



(a) Number of query nodes (b) Number of node comparisons (c) Response Time

Fig. 7. Experimental Results of XML Query Reduction Algorithm with Xmark

Fig. 7 summarizes the experimental results. For each query set of the same number of query nodes that is between 5 and 12, we measured the average number of record comparisons that have been done during structural joins and the average response time, against both the original queries and the reduced queries. Fig. 7-(a) shows that the reduced queries resulting from the path reduction were shortened up to 28 % of the length of the original queries and in one example queries with length 7 were shortened to an average of 2. Fig. 7-(b) shows that in the best case the number of record comparisons during the structural joins also decreased to only 14 % of the number of record comparisons by the original queries. Accordingly, the response time was reduced to 8 % of the original query as shown in Fig. 7-(c).

Meanwhile, the average query reduction cost was about 0.04 sec, which was less than 1 % of the average query execution time. As the result shows, up to 92% of the original query execution time could be eliminated with only a marginal overhead in query reduction time. Though the effectiveness of the query reduction algorithm could vary depending on the structure of the XIP tree and the queries in the domain area, in most cases, it is expected to improve the query execution time with very little overhead in query reduction time.

6 Conclusion

This paper proposed two XML path reduction algorithms and an XML branch query reduction algorithm that can reduce the number of query nodes in a complex XML query so that an XML query processor exploiting node number schemes and structural joins can more efficiently execute the query. The schemes use XIP trees, which reflect the summarized structure of input XML document instances. The equivalence class concept among regular path expressions is very useful for reducing path expressions. Experimental results show that the presented branch query reduction algorithm could eliminate up to 92 % of the original query execution time with only a little extra cost for query reduction. As a result, the performance of the XML query execution was enhanced by up to an order of magnitude.

References

1. S.-Y. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo. Efficient structural joins on indexed XML documents. In Proc. of the 28th VLDB conference, pages 263-274, Hong Kong, China, Aug. 2002.
2. V. Christophides, S. Cluet, and G. Moerkotte. Evaluating queries with generalized path expressions. In Proc. of the 1996 ACM-SIGMOD conference, pages 413-422, Montreal, Canada, Jun. 1996.
3. B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In Proc. of the 27th VLDB conference, pages 341-350, Rome, Italy, Sep. 2001.
4. R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In Proc. of the 23rd VLDB conference, pages 436-445, Athens, Greece, Aug. 1997.
5. Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In Proc. of the 27th VLDB conference, Rome, Italy, Sep. 2001.
6. A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse. XMark: A Benchmark for XML Data Management. In Proc. of the 28th VLDB conference, pages 974-985, Hong Kong, China, Aug. 2002.
7. D. Srivastava, S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, and Y. Wu. Structural joins: A primitive for efficient XML query pattern matching. In Proc. of the 2002 IEEE conference on Data Engineering, Feb. 2002.
8. C. Zhang, J. F. Naughton, Q. Luo, and D. J. DeWitt, and G. M. Lohman. On supporting containment queries in relational database management systems. In Proc. of the 2001 ACM-SIGMOD conference, May 2001.
9. M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In Proc. of the 1998 IEEE Conference on Data Engineering, pages 4-13, Orlando, Florida, Feb. 1998.
10. Sleepycat Software Inc., http://www.sleepycat.com.
11. H. Jiang, H. Lu, W. Wang, and B. C. Ooi, XR-Tree: Indexing XML Data for Efficient Structural Joins. In Proc. of the 2003 IEEE conference on Data Engineering, pages 253-263, Bangalore, India, March 2003.
12. H. Li, M. Lee, W. Hsu, and C. Chen, An Evaluation of XML Indexes for Structural Join. SIGMOD Record 33(3), pages 28-33, 2004.
13. Y. Wu, J. M. Patel, and H. V. Jagadish, Structural Join Order Selection for XML Query Optimization. In Proc. of the 2003 IEEE conference on Data Engineering, pages 443-454, Bangalore, India, March 2003.
14. C.-W. Chung, J.-K. Min, and K. Shim, APEX: an adaptive path index for XML data, In Proc. of the 2002 ACM-SIGMOD conference, pages 121-132, Madison, Wisconsin, USA, Jun. 2002.
15. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, Storing and querying ordered XML using a relational database system, In Proc. of the 2002 ACM-SIGMOD conference, Jun. 2002.
16. N. Bruno, N. Koudas, and D. Srivastava, Holistic twig joins: optimal XML pattern matching, In Proc. of the 2002 ACM-SIGMOD conference, pages 311-321, Madison, Wisconsin, USA, Jun. 2002.
17. H. Jiang, W. Wang, H. Lu, and J. X. Yu, Holistic Twig Joins on Indexed XML Documents, In Proc. of the 29th VLDB conference, pages 273-284, Berlin, Germany, Sep. 2003.

Automated Migration of Data-Intensive Web Pages into Ontology-Based Semantic Web: A Reverse Engineering Approach

Sidi Mohamed Benslimane, Mimoun Malki, and Djamel Amar Bensaber

EEDIS Laboratory, Computer science Department,
University of Sidi Bel Abbes 22000, Algeria
benslimane@univ-sba.dz, Malki_m@yahoo.com, am_wap2003@yahoo.fr

Abstract. The advance of the Web has significantly and rapidly changed the way of information organization, sharing and distribution. The next generation of the web, the semantic web, seeks to make information more usable by machines by introducing a more rigorous structure based on ontologies. In this context we try to propose a novel and integrated approach for an automated migration of data-intensive web pages into ontology-based semantic web and thus, make the web content machine-understandable. Our approach is based on the idea that semantics can be extracted from the structures and the instances of database forms which are the most convenient interface to communicate with relational databases on the current Web. This semantics is exploited to help build ontology.

1 Introduction

The actual web has been moving away from static, fixed web pages to dynamically-generated at the time of user request. This kind of web site is called data-intensive web site [1], and usually realized using relational databases (i.e. e-commerce application). Data-intensive web pages are characterized by an automated update of the web content and a simplified maintenance of the web design [2]. Nevertheless they suffer from two limitations. First, they form a hidden web since its content is not easily accessible to any automatic web content processing tools including the search engine indexing robots. Second the content of the database-driven web pages presented by using HTML is not machine-understandable. The next generation of the web, the semantic web, seeks to make information more usable by machines by introducing a more rigorous structure based on ontologies. Ontology is a formal, explicit specification of a shared conceptualization [3]. In this paper we propose a novel and integrated approach for an automated migration of data-intensive web pages into semantic web and thus, make the web content machine-understandable. The best approach seems to rely on reverse engineering [4] rather than on semantic annotation [5], which is time consuming and error-prone.

This paper is organized as follows: In Section 2, we discuss some of the related works in reverse engineering relational databases into ontologies. Section 3 explains the overall reverse-engineering architecture and details our proposed approach, Whereas Section 4 contains conclusion remarks and futures works.

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1640 – 1649, 2005.

2 Related Works

Several researches have been done on relational databases reverse engineering, suggesting methods and rules for extracting entity-relationship and object models from relational databases [6, 7, 8, 9].However, there are few approaches that consider ontologies as the target for reverse engineering. These approaches fall roughly into one of the five categories: 1. **Approaches based on an analysis of user queries:** E.g. Kashyap's approach [10] builds an ontology based on an analysis of relational schema; the ontology is then refined by user queries. However, this approach does not create axioms, which are part of the ontology. 2. **Approaches based on an analysis of relational schema:** E.g. Stojanovic et al's approach [2] provides a set of rules for mapping constructs in the relational database to semantically equivalent constructs in the ontology. These rules are based on an analysis of relations, keys and inclusion dependencies. Dogan & Islamaj's approach [11] provides simple and fully automatic reverse engineering: relations map to classes, attributes in the relations map to attributes in the classes and tuples in the relational database map to instances in the ontology. However, this approach ignores inheritance, thus building the ontology that looks rather "relational". 3. **Approaches based on an analysis of tuples:** E.g. Astrova's approach [12] builds an ontology based on an analysis of relational schema. Since the relational schema often has little explicit semantics [13], this approach also analyzes tuples in the relational database to discover additional "hidden" semantics (e.g. inheritance). However, this approach is very time consuming with regard to the number of tuples of relational database. 4. **Approaches based on an analysis of HTML-table:** E.g. Tijerino's approach [14] based on conceptual modeling extraction technique attempts to understand a table's structure and conceptual content, discover the constraints that hold between concepts extracted from the table, match the recognized concepts with ones from a more general specification of related concepts, and merge the resulting structure with other similar knowledge representations. However, this approach requires auxiliary informations including dictionaries and lexical data (WordNet, Natural language parsers, and data frames library). 5. **Approaches based on an analysis of HTML-forms:** E.g. Astrova's approach [15] constructs an ontology based on an analysis of HTML-forms by analyzing the HTML-forms to extract a form model schema, transforming the form model schema into ontology and creating ontological instances from data contained in the pages. The drawback of this approach is that this approach does not offer any way to the identification of inheritance relationship which is a significant aspect in the ontology construction.

3 Our Approach

Our approach enriches the semantics of the database by providing additional ontological entities. It uses the information extracted from both HTML forms structure and instances as a databases reverse engineering input. This can be supported by the following arguments:

- HTML forms are often the most popular and convenient interfaces for entering, changing and viewing data in the actual data-intensive web pages and, therefore, important information can be obtained by analyzing an HTML forms; A form model

is a data model. Studying and analyzing an HTML form and its relationship to other forms can reveal many data dependencies and mapping; HTML forms are structured collections of fields formatted to communicate with the relational database. Therefore, data contained in the forms is usually structured, while the structure of the relation databases is often unknown in advance [16]; Field names in HTML forms are usually more explicit and meaningful than the names of corresponding attributes in the relational databases; Normally the HTML forms are accompanied by instructions which provide additional information about organisation's data and their behavior parts. These instructions are part of the contextual knowledge [17].

A form model schema was originally proposed, suitable for databases reverse engineering task [18]. The model allows abstracting any database form, that is, to make explicit its components, fields as well as objects, and their interrelationships. This model (figure 1) is similar but not identical to the models presented in [16, 19]. Basically, this model consists of: *Form type*: Is a structured collection of empty fields formatted to communicate with databases. A particular representation of form type is called *form template.* A form template defines the structure, constraints and presentation of the form fields. It represents the forms intension as perceived by users. Three basic components of any template are title, captions, and entries. *Structural units*: Is a group of homogeneous pieces of information, that is, an object that groups closely related form fields. Each structural unit is a logical sub-part of a form type. It generally corresponds with areas on a form layout. *Form instance:* Is an occurrence of a form type. This is the extensional part obtained when a form template is filled in with data. Figure 2 is an instance of the "booking form" and "program of flight" forms type. *Form fields:* Is an aggregation of a caption with its associated entry. Caption is pre-displayed on the form and serves as a clue as what is to be filled in by the respondent as well as a guide to enter or read it on the form. An entry is the actual data that is entered by a user or displayed by the form processing system. Notice that we can identify a form field where there is no caption for an entry or inversely. Each form field entry is generally linked to an attribute of one table in the underlying database. That is, the values it display (or receives) are provided by (or store in) this attribute. We use the concept of linked-attribute to designate this attribute. Some form fields are computed; others can be simply unlinked with the relational database. We distinguish three types of fields: Filling fields tag (e.g., TEXT, CHECKBOX, RADIO etc.) which are an aggregation of name and entry associated to it. Selection fields (e.g., SELECT tag) which let the user select one, or more than one choice (MULTIPLE attribute). Link fields which are used to relied tow or more forms (pages) (HREF tag). *Underlying source:* this is a structure of the relational database (i.e. a relational schema), which defines relations and attributes along with their data types. *Relationships:* this is a connection between structural units that relates one structural unit to another (or back to itself). There are two kinds of relationship: association and inheritance. *Constraint:* This is a rule that defines what data is valid for a given form field. A *cardinality constraint* specifies for an association relationship the number of instances that a structural unit can participate in.

Our approach articulates around five phases: Analysis the HTML forms to identify constructs in the form model schema; Construction of hierarchical structure of forms; Extraction of the domain semantics; Mapping of the global schema into ontology; Creating ontological instances from data contained in the page.

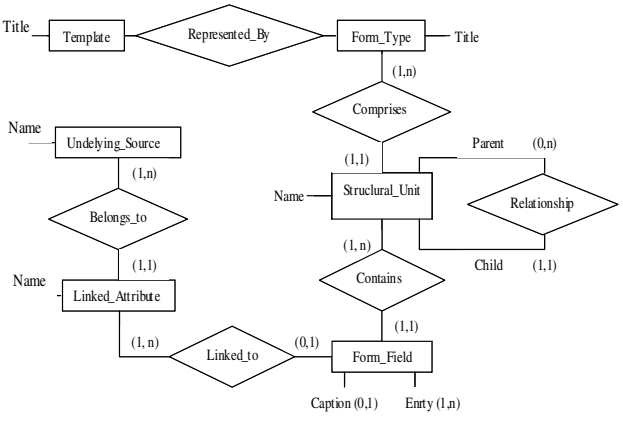


Fig. 1. Form Model

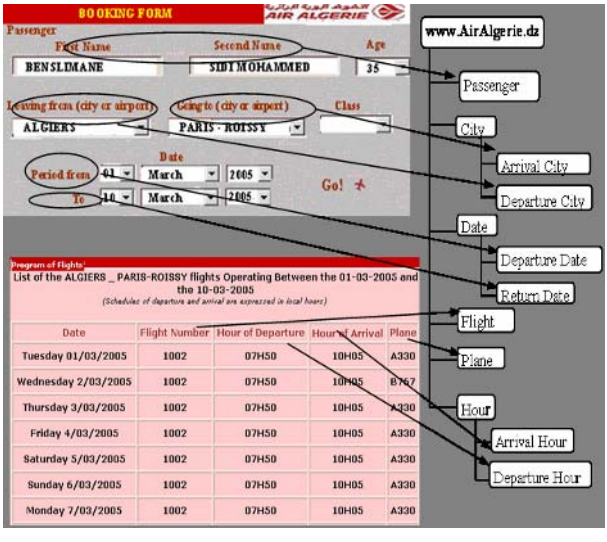


Fig. 2. Ontology extraction from an HTML forms

3.1 Analysis HTML Pages Structure

The main goal of this phase is to understand the form meaning and explicit its structure by analyzing HTML forms (both their structure and data they contain) to identify its components and their interrelationships and extract a form model schema. The analysis goes through the following steps: *1. Identifying form instances:* In order to clearly distinguish different kinds of information in the document, the web pages are usually split to multiple areas. Each area is crated using specific tags. For our approach we consider only the section between the open and closing <form> tag. *2 Identifying linked attributes.* Linked attributes are identified by examining the HTML code for structural tags such as <thead> and <th> [20]. If the linked attributes aren't

separated with the structural tags (merged data), we can use visual cues [4, 21]. This approach typically implies that there will be some separators (e.g. blank areas) that help users split the merged data. *3 Identifying structural units:* To determine a logical structure of HTML page (i.e. the real meaning of the page, as it is understood by users), we can use visual cues [21] E.g. the users might consider the FirstName, SecondName, and Age in Fig. 1 as a whole group (passenger), just because they are specifications too. *4 Identifying relationships:* The association can be indicated by the fact that the two structural units appear at the same page. If the two structural units come together, they might be logically related to each other. We would also identify an association relationship between two structural units using hyperlinks. By clicking on a hyperlink in one structural unit (at some page), we can go to another structural (possibly at another page). E.g. from Fig. 1.

3.2 Construction of Hierarchical Structure of Forms

In order to have a precise picture of the hierarchical relationships of a form and to clearly understand its meaning and facilitate the extraction of the domain semantics, this process, which is automatic and transparent to the designer, constructs the hierarchical structure in three steps: – Defining the root node whose name is the form's title. All fields are attached to the root node; – In the first, transforming all structural units into a node with level 1, i.e., their parent is the root node; -Identifying the parent-child relationships among structural units. Because there are several level of fields (e.g., 0, 1, 2, 3, etc.), this process identify recursively the substructure (Or sub-node) according to levels of their fields specified in the insertion process of forms.

3.3 Extraction of the Domain Semantics

The goal of this phase of extraction is to derive the relational sub-schema of form from their hierarchical structure and their instances according to the physical schema of the underlying database. First, the relation and their primary keys are respectively identified with regard to both structural units (or nodes) of form and underlying database, then the functional and inclusion dependencies are extracted through both their hierarchical structure and instances.

3.3.1 Form Relations Extraction

The forms either permit the updating of relations in underlying database or represent a view that is a joint of relations. Therefore, each field entry is generally linked to an attribute of one relation in the underlying database. However, the identification of form relations and their primary keys respectively, consists of determining the equivalence and/or the similarity between structural units (nodes) of hierarchical structure and relations in the underlying database. This is a basis point from a reverse engineering point of view [18].

A node of a form hierarchical structure may be either: – Equivalent to a relation in the underlying database, i.e., these two objects (node and relation) have a same set of attributes; – Similar to a relation, i.e., its set of attributes is a subset of the one of the relation; – A set of relations, i.e., its set of attributes regroups several relations in underlying database.

Also, for dependent nodes (or form relation), primary keys are formed by concatenating the primary key of its parent with its local primary key. This process of identification is semi-automated because it requires the interaction with the analyst to identify objects that do not verify proprieties of equivalence and similarity.

While applying this process on the hierarchical structure of "Booking Form" and the physical relational schema of underlying database, we extract the following relational sub-schemas:

Passenger (PassengerID, FirstName, FamillyName) City (CityID, Name)
DepartureCity (CityID) ArrivalCity (CityID) Date (DeparatueDate)

From the "program flights" form we identify the following relational sub-schemas:

DepartureHour (Dep_HourID, type) ArrivalHour (Arr_HourID, type)
Plane (PlaneID, Capacity) Flight (ID, DeparatueDate, DepartureCityID, ArrivalCityID, Dep_HourID, Arr_HourID, PlaneID)

From the relationships among hierarchical structure of "Booking Form" and "program flight" forms we identify the following relational sub-schemas:

Book (PassengerID, FlightID, DepartureDate) LeavingFrom (FlightID, DepartureCityID) GoingTo (FlightID, ArrivalCityID)

3.3.2 Functional Dependencies Extraction

The extraction of functional dependencies from the extension of database has received a great deal of attention [22, 23, 24] In our approach we use the algorithm introduced by [18] to reduce the time for exacting functional dependencies by replacing database instances with a more compact representation that is, the form instances. While applying this algorithm on the sub-schema of "program of flights" and their instances, one finds the FDs: *Flight.ID →DepartureCity.CityID ; Flight.ID →ArrivalCity.CityID*

3.3.3 Inclusion Dependencies Extraction

In our approach, we formulate possible inclusion dependencies (Inds) between relations' key of relational sub-schema of form. The time of this process is more optimized with regard to the other approaches [24, 6] because the possible Inds are verified by analyzing the form extensions which are more compact representation with regard to the database extension.

In this algorithm, attributes of dependencies are the primary keys and foreign keys. Thus, the time complexity is reduced to the test of the inclusion dependency on the form instances. The set of the Inds extracted is:

Book.FlightID <<Flight.FlightID ; Book.PassengerID <<Passenger.PassengerID

3.3.4 Integration of the Global Schema of Forms

This process performs as follows. First, the schemata are compared for overlaps in structure. This means looking for structural units and relationships with similar names, and then looking for similar structures within structural units and relationships. Second, the schemata are compared for overlaps in meaning. This means looking for structural units that correspond to the same real-world objects, but

have different names. Third, naming conflicts (i.e. synonyms and homonyms) are resolved. Conflicts can also be in different constraints on the linked attributes and different cardinality constraints on the relationships. By performing these three tasks, the integration process makes the schemata consistent with one another and brings them together into a single one that makes sense for all pages from a given website.

3.4 Mapping of Conceptual Form Schema into Ontology

The mapping process is a transformation of the form relational schema into ontology; the transformation is usually a collection of mapping rules that replace constructs in the form relational schema with (semantically equivalent) ontological entities. Our rules are similar to those used in [9] to perform a transformation into an object oriented model. Basically, the process uses as the main input, the constructs generated from the precedent step. It goes through five steps and is based on the classification of relations into one of the three categories. *Base relation:* if a relation is independent of any other relation in a form relation schema. *Example: City (CityId, CityName). Dependent relation*: if a primary key of a relation depends on another relation's primary key. Example: Book (PassengerID, FlightID, DepartureDate). Composite relation: if it is neither base nor dependent. Example: Flight (ID, DepartureCityID, ArrivalCityID, Dep_HourID, Arr_HourID, FlightDate).

3.4.1 Identification of Object Class

The general assumption is that each base relation is mapped into an object class. These object classes have the same attributes as those contained in the relations. Example:

Passenger (PassegerId, FirstName, SecondName, Age)
↓
Passenger :: Object.
Passenger [PassengerID =>> Integer, FirstName =>> String, SecondName =>> String, Age =>> Integer].

3.4.2 Identification of Roles

For each binary relation containing only a pair of foreign key attributes of class relation, we create a role attribute in each corresponding class, typed by the other class. Example:

LeavingFrom (FlightID, DepartureCityID).
Flight.ID →DepartureCity.CityID
↓
DepartureCity :: Object Flight :: Object
DepartureCity [CityID =>> Integer, FlightUD =>> Flight].
Flight [FlightUD =>> Integer, CityID =>> Integer].

3.4.3 Identification of Association Class

For every n-airy relation whose primary key is entirely composed of foreign keys, we create an association class. Example:

Book (PassegerID, FilightID, DepartureDate)
↓
Book :: Object.
Book [PassengerID =>> Integer, FilightID =>> Integer, DepartureDate =>> Integer].

3.4.4 Identification of Inheritance Relationships

Extracting inheritance relation-ship from a relational schema usually requires behavioral information. Every pair of relations ($R1$, $R2$) that have the same primary key (noted X) and the corresponding Inds (i.e., $R1.X << R2.X$) may be involved in an inheritance relationship, i.e., $R1$ "is-a" $R2$. Example: the Relations City and DepartureCity have the same primary key (CityID) and the corresponding *Ind*: DepartureCity.CityID << City.CityID, therefore City is the superclass and Departure_city is a subclass.

3.4.5 Identification of Axioms

To preserve the semantics embedded within a relational database, we crate an axiom for each constraint in the form relational schema. (e.g., PRIMARY KEY, NOT NULL, etc.). Figure 3 gives an example of mapping a constraint PRIMARY KEY. An attribute is a *primary key*, if it is unique and total. An attribute is *unique*, when no two tuples in the relation have the same value for that attribute, and no tuple in the relation has two values for that attribute. An attribute is *total* , if all tuples in the relation contain values for that attribute.

```
PrimaryKey (City, CityID).
Forall C, P PrimaryKey(C, P) <-
Unique(C, P) and Total(C, P).
Forall C, P Unique(C, P) <-
IdenticalValues(C, P) and SingleValue(C, P).
Forall C, P IdenticalValues(C, P) <-
Forall I1, I2, IP I1:C and I2:C and
I1[P->>IP] and IC[P->>IP] and Equal(I1, I2).
Forall C, P SingleValue(C, P) <-
Forall IC, IP1, IP2 IC:C and
IC[P->>IP1] and IC[P->>IP2] and Equal(IP1, IP2).
Forall C, P Total(C, P) <-
Forall IC Exists IP IC:C and IC[P->>IP].
```

Fig. 3. Mapping constraints

3.5 Migrating Data

The objective of this task is the creation of ontological instances based on the tuples of the relational database. The data migration process has to be performed in two phases. In the first phase the instances are created. To each instance is assigned a unique identifier. This translates all attributes, except for foreign-key attributes, which are not needed in the metadata. In the second phase, relations between instances are established using the information contained in the foreign keys in the database tuples. This is accomplished using a mapping function that maps keys to ontological identifiers. Figure 4, illustrates an example result of the data migration process.

Relational model

Plane		
PlaneID	Capacity	CompanyID
A330	150	1
B767	200	2
B747	300	1

Company	
CompanyID	CompanyName
1	Air Algerie
2	Air France

Ontology instances

id1:Plane[PlaneID->>A330, capacity->>150; company->> id4].
id2:Plane[PlaneID->>B767, capacity->> 200; company->> id5].
id3:Plane[PlaneID->>B747, capacity->> 300; company->> id4].
Id4:Company[companyID->>1,companyName->>'AirAlgerie'].
Id5:Company[companyID->>2,companyName->>'Air France'].

Fig. 4. Data migration example

4 Conclusion

We have developed a novel, integrated and semi-automated approach for migrating data-intensive Web applications into the Semantic Web that can be applied to a broad range of today's business Web sites. The approach starts with transforming the relational database model into corresponding ontological structures, which is then used for mapping the content of the database into an ontology-based knowledge base. There are two main reasons why further work is still necessary in the area of reverse engineering of relational databases to ontologies. First, there exist few approaches that consider ontologies as the target for reverse engineering. Second, applicability of the existing input information than it is possible to provide in practice and making unrealistic assumptions about the input. In our opinion, these limitations stem primarily from the fact that the existing approaches do not take advantage of HTML forms. Rather, the existing approaches build an ontology based on an analysis of relational schema, tuples, and user queries. As an attempt to overcome the limitations, we have proposed a novel approach. In the future, a combination between domain ontology and HTML-forms analysis technique can be exploited not only to extracted ontology but also to migrate from the current Web to the semantic Web.

References

1. Fraternali, P., "Tools and approaches for developing data-intensive web applications: a survey", ACM Computing Surveys, Vol.31, No.3, pp. 227-263, 1999.
2. L. Stojanovic, N. Stojanovic & R. Volz, "Migrating Data-intensive Web Sites into the Semantic Web", Proc. 17th ACM Symposium on Applied Computing , Madrid, Spain, 2002.
3. T.R.Gruber, "Toward principles for the design of ontologies used for knowledge sharing", Human Computer Studies, 43 (5-6), 1995, 907-928.
4. J. Wang & F. Lochovsky, "Data Extraction and Label Assignment for Web Databases" , Proc. 12th International Conference on World Wide Web , Budapest, Hungary, 2003, 187–196.
5. Hainaut, J. Henrard, J. Hick, D. Roland & V. Englebert, "Database Design Recovery", Proc. 8th Conference on Advanced Information Systems Engineering (CAiSE), Heraklion, Crete, Greece, LNCS, 1080, 1996, 272–300.

6. Chiang, R.H.L., T.M. Barron, V.C. Story. "Reverse engineering of relational databases: extraction of an EER model from a relational database".Data and Knowledge Engineering, 1994.
7. Vermeer, M., Apers. P. "Object-oriented views of relational databases incorporation behaviour", Proceedings of the 4th International Conference on databases systems for Advenced Application (DASFAA), Singapore, April 11-13, 1995, pp 26-35
8. Behm A., Geppert A., Dittrich, K. (1997) On the Migration of Relational Schemas and Data to Object-Oriented Database Systems. In Proceeding of the 5th Int. Conference on Re-Technologies for Information Systems (Klagenfurt, December 1997), pp. 13-33, 1997.
9. MALKI, M., A. FLORY, M.K. RAHMOUNI "Extraction of Object-oriented Schemas from Existing Relational Databases: a Form-driven Approach', INFORMATICA, 2001, Vol. 12, No. 4, 1.
10. V. Kashyap, "Design and Creation of Ontologies for Environmental Information Retrieval" , Proc. 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW), Banff, Alberta, Canada, 1999.
11. G. Dogan & R. Islamaj, "Importing Relational Databases into the Semantic Web", http://www.mindswap.org/webai/2002/fall/Importing_20Relational_20Databases_20into_20the_20Semantic_20Web.html, 2002.
12. I. Astrova, "Reverse Engineering of Relational Databases to Ontologies" , Proc. 1st European Semantic Web Symposium , Heraklion, Crete, Greece, LNCS, 3053, 2004, 327–341.
13. N.Noy & M.Klein, "Ontology evolution: not the same as schema evolution." Report Number: SMI-2002-0926:2002
14. Y.A Tijerino, D.W. Embly, D.W. Lonsdale, Y. Ding, G. Nagy "Towards Ontology Generation from tables". Kluwer Academic Publishers 2004
15. I. Astrova, B. Stantic "An HTML Forms driven Approach to Reverse Engineering of Relational Databases to Ontologies", in proceeding of the 23rd IASTED International Conference on Databases and Applications (DBA), eds. M. H. Hamza, Innsbruck, Austria, 2005, pp. 246- 251
16. Choobineh, J. "A form-based approach for database analysis and design" . Communication of the ACM, 35(2) 1992.
17. M. Mannino, J. Choobineh and J. Hwang, Acquisition and Use of Contextual Knowledge in a Form-Driven Database Methodology, In: Proceedings of the 5th International Conference on the Entity-Relationship Approach (1986) 141–157.
18. M. Malki, M. Ayache, M.K. Rahmouni. "Rétro-ingénierie des Bases de Données Relationnelles: Approche Basée sur l'Analyse de Formulaires ». In Proc.of Colloque of INFORSID'99. Toulon, France 1999.
19. Mfourga, N. "Extracting entity-relationship schemas from relational databases: a form-driven approach." In Proc. of Working Conf. on Reverse EngineeringWCRE'97 1997.
20. D. Embley, "Toward Semantic Understanding – An Approach Based on Information Extraction", Proc. 15th Australasian Database Conference, Dunedin, New Zealand, 2004, 3–12.
21. Y. Yang & H. Zhang, " HTML Page Analysis Based on Visual Cues", Proc. 6th International Conference on Document Analysis & Recognition , Seattle, USA, 2001, 859–864.
22. Anderson, M. "Extracting a E.R. schema from a relational database through reverse engineering" . In Proc. of the 13th Inter. Conf. on the ERA'94, pp. 403–419, 1994.
23. Mannila, H. et al. *The Design of Relational Databases.* Addison-Wesley publishing, 1994.
24. Petit, J.M., F. Toumani, J. Kouloumdjian. « Relational database reverse engineering: a method based on Query analysis". Inter. Journal of Cooperative Information System, 1995.

Author Index