Robert Meersman
Tharam Dillon
Pilar Herrero et al. (Eds.)

LNCS 7045

On the Move to Meaningful Internet Systems: OTM 2011

**Confederated International Conferences:
CoopIS, DOA-SVI, and ODBASE 2011
Hersonissos, Crete, Greece, October 2011, Proceedings, Part II**

2 Part II

Springer

Robert Meersman Tharam Dillon Pilar Herrero
Akhil Kumar Manfred Reichert Li Qing
Beng-Chin Ooi Ernesto Damiani Douglas C. Schmidt
Jules White Manfred Hauswirth Pascal Hitzler
Mukesh Mohania (Eds.)

On the Move to Meaningful Internet Systems: OTM 2011

Confederated International Conferences:
CoopIS, DOA-SVI, and ODBASE 2011
Hersonissos, Crete, Greece, October 17-21, 2011
Proceedings, Part II

Springer

Volume Editors

Robert Meersman, Vrije Universiteit Brussel, Belgium, meersman@vub.ac.be

Tharam Dillon, Curtin University of Technology, Australia, t.dillon@curtin.edu.au

Pilar Herrero, Universidad Politécnica de Madrid, Spain, pherrero@fi.upm.es

Akhil Kumar, Pennsylvania State University, USA, akhilkumar@psu.edu

Manfred Reichert, University of Ulm, Germany, manfred.reichert@uni-ulm.de

Li Qing, City University of Hong Kong, liqing.thu@gmail.com

Beng-Chin Ooi, National University of Singapore, ooibc@comp.nus.edu.sg

Ernesto Damiani, University of Milan, Italy, ernesto.damiani@unimi.it

Douglas C. Schmidt, Vanderbilt University, USA, schmidt@dre.vanderbilt.edu

Jules White, Virginia Tech, Blacksburg, USA, julesw@vt.edu

Manfred Hauswirth, DERI, Galway, Ireland, manfred.hauswirth@deri.org

Pascal Hitzler, Kno.e.sis, Wright State University, USA, pascal.hitzler@wright.edu

Mukesh Mohania, IBM India, New Delhi, mkmukesh@in.ibm.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25105-4 e-ISBN 978-3-642-25106-1
DOI 10.1007/978-3-642-25106-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011940438

CR Subject Classification (1998): C.2, D.2, H.4, I.2, H.2-3, J.1, K.6.5

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

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

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

General Co-chairs' Message for OnTheMove 2011

The OnTheMove 2011 event in Heraklion, Crete, held during October 17–21, further consolidated the growth of the conference series that was started in Irvine, California, in 2002, and held in Catania, Sicily, in 2003, in Cyprus in 2004 and 2005, in Montpellier in 2006, in Vilamoura in 2007 and 2009, in Monterrey, Mexico, in 2008, and in Heraklion 2010. The event continues to attract a diversified and representative selection of today's worldwide research on the scientific concepts underlying new computing paradigms, which, of necessity, must be distributed, heterogeneous, and autonomous yet meaningfully collaborative. Indeed, as such large, complex, and networked intelligent information systems become the focus and norm for computing, there continues to be an acute and even increasing need to address and discuss face to face in an integrated forum the implied software, system, and enterprise issues as well as methodological, semantic, theoretical, and application issues. As we all realize, email, the Internet, and even video conferences are not by themselves sufficient for effective and efficient scientific exchange.

The OnTheMove (OTM) Federated Conference series has been created to cover the scientific exchange needs of the community/ies that work in the broad yet closely connected fundamental technological spectrum of Web-based distributed computing. The OTM program every year covers data and Web semantics, distributed objects, Web services, databases, information systems, enterprise workflow and collaboration, ubiquity, interoperability, mobility, grid and high-performance computing.

OnTheMove does not consider itself a so-called multi-conference event but instead is proud to give meaning to the "federated" aspect in its full title : it aspires to be a primary scientific meeting place where all aspects of research and development of Internet- and intranet-based systems in organizations and for e-business are discussed in a scientifically motivated way, in a forum of (loosely) interconnected workshops and conferences. This tenth edition of the OTM Federated Conferences event therefore once more provided an opportunity for researchers and practitioners to understand and publish these developments within their individual as well as within their broader contexts. To further promote synergy and coherence, the main conferences of OTM 2011 were conceived against a background of three interlocking global themes:

– Virtual ("Cloud") Computing Infrastructures and Security
– The Internet of Things, and Semantic Web 2.0
– Collaborative ("Social") Computing for the Enterprise

Originally the federative structure of OTM was 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) cover the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. In 2007 the IS workshop (Information Security) was added to try cover also the specific issues of security in complex Internet-based information systems, and in this 2011 edition these security aspects became merged with DOA under the umbrella description of "secure virtual infrastructures," or DOA-SVI. Each of the main conferences specifically seeks high-quality contributions and 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 Web-based 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 six of our earlier successful workshops (EI2N, SWWS, ORM, MONET,ISDE, SeDeS) re-appeared in 2011 with in some cases a fourth or even fifth edition, often in alliance with other older or newly emerging workshops, and that three brand-new independent workshops could be selected from proposals and hosted: INBAST, RASEP, and VADER. (INBAST was merged with the new Industry Track, under the auspicious leadership of Hervé Panetto and OMG's Richard Mark Soley.)

We are also proud in particular to note the co-sponsorship of the US National Science Foundation (NSF) for the EI2N workshop (also initiated by Hervé), and which increasingly profiles itself as a successful incubator for new "CoopIS-related" research aspects and topics. Our OTM registration format ("one workshop buys all") actively intends to stimulate workshop audiences to productively mingle with each other and, optionally, with those of the main conferences.

We were again most happy to see once more in 2011 the number of quality submissions for the OnTheMove Academy (OTMA, formerly called Doctoral Consortium Workshop), our "vision for the future" in research in the areas covered by OTM, managed by a dedicated team of collaborators led by Peter Spyns and Anja Schanzenberger, and of course by the OTMA Dean, Erich Neuhold, responsible for our unique interactive formula to bring PhD students together. In the OTM Academy, PhD research proposals are submitted for peer review; selected submissions and their approaches are (eventually) presented by the students in front of a wider audience at the conference, and independently and extensively analyzed and discussed in front of the audience by a panel of senior professors.

As said, all three main conferences and the associated workshops shared the distributed aspects of modern computing systems, and the resulting application pull created by the Internet and the so-called Semantic Web. For DOA-SVI 2011, the primary emphasis stayed on the distributed object infrastructure and its virtual and security aspects; for ODBASE 2011, the focus became the knowledge bases and methods required for enabling the use of formal semantics in Web-based databases and information systems; for CoopIS 2011, the focus as usual was on the interaction of such technologies and methods with management issues, such as occur in networked organizations and enterprises. These subject areas overlap in a scientifically natural fashion and many submissions in fact also treated an envisaged mutual impact among them. As with the earlier editions, the organizers wanted to stimulate this cross-pollination by a "shared" program of famous keynote speakers around the chosen themes. We were quite proud to announce:

– Amit Sheth, Wright State University, Ohio, USA
– Schahram Dustdar, Vienna University of Technology, Austria
– Siani Pearson, Hewlett-Packard Laboratories, Bristol, UK
– Niky Riga, Raytheon BBN Technologies, Massachusetts, USA

We received a total of 141 submissions for the three main conferences and 104 submissions in total for the workshops. The numbers are comparable with those for 2010. Not only may we indeed again claim success in attracting an increasingly representative volume of scientific papers, many from the USA and Asia, but these numbers of course allow the Program Committees to compose a high-quality cross-section of current research in the areas covered by OTM. In fact, the Program Chairs of the CoopIS 2011 conferences decided to accept only approximately 1 paper for each 5 submissions, while the ODBASE 2011 PC accepted about the same number of papers for presentation and publication as in 2009 and 2010 (i.e., average 1 paper out of 3-4 submitted, not counting posters). For the workshops and DOA-SVI 2011 the acceptance rate varies but the aim was to stay consistently at about 1 accepted paper for 2-3 submitted, and this of course subordinated to peer assessment of scientific quality.

As usual we have separated the proceedings into three volumes with their own titles, two for the main conferences and one for the workshops, and we are again most grateful to the Springer LNCS team in Heidelberg for their professional suggestions and meticulous collaboration in producing the files for downloading on the USB sticks.

The reviewing process by the respective Program Committees was again performed very professionally, and each paper in the main conferences was reviewed by at least three referees, with arbitrated email discussions in the case of strongly diverging evaluations. It may be worth emphasizing that it is an explicit On-TheMove policy that all conference Program Committees and Chairs make their selections completely autonomously from the OTM organization itself. Like last year, paper proceedings were on separate request and order this year, and incurred an extra charge.

The General Chairs are once more especially grateful to the many people directly or indirectly involved in the set-up of these federated conferences. Not everyone realizes the large number of persons that need to be involved, and the huge amount of work, commitment, and in the uncertain economic and funding climate of 2011 certainly also financial risk, the organization of an event like OTM entails. Apart from the persons in their roles mentioned above, we therefore wish to thank in particular our eight main conference PC Co-chairs:

– CoopIS 2011: Manfred Reichert, Akhil Kumar, Qing Li
– ODBASE 2011: Manfred Hauswirth, Pascal Hitzler, Mukesh Mohania
– DOA-SVI 2011: Ernesto Damiani, Doug Schmidt, Beng Chin Ooi

And similarly the 2011 OTMA and Workshops PC (Co-)chairs (in arbitrary order): Hervé Panetto, Qing Li, J. Cecil, Thomas Moser, Yan Tang (2x), Alok Mishra, Jürgen Münch, Ricardo Colomo Palacios, Deepti Mishra, Patrizia Grifoni, Fernando Ferri, Irina Kondratova, Arianna D'Ulizia, Terry Halpin, Herman Balsters, Almudena Alcaide, Naoki Masuda, Esther Palomar, Arturo Ribagorda, Yan Zhang, Jan Vanthienen, Ernesto Damiani (again), Elizabeth Chang, Paolo Ceravolo, Omar Khadeer Hussain, Miguel Angel Pérez-Toledano, Carlos E. Cuesta, Renaud Pawlak, Javier Cámara, Stefanos Gritzalis, Peter Spyns, Anja Metzner, Erich J. Neuhold, Alfred Holl, and Maria Esther Vidal.

All of them together with their many PC members, performed a superb and professional job in managing the difficult yet existential process of peer review and selection of the best papers from the harvest of submissions. We are all also grateful to our supremely competent and experienced Conference Secretariat and technical support staff in Antwerp and Guadalajara, Jan Demey and Daniel Meersman, and last but certainly not least to our proceedings production team in Perth (DEBII-Curtin University) this year led by Christopher Jones.

The General Co-chairs acknowledge with gratitude the academic freedom, logistic support, and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB), Curtin University, Perth, Australia, and Universidad Politécnica de Madrid (UPM), without which such an enterprise quite simply would not be feasible. 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 event!

August 2011

Robert Meersman
Tharam Dillon
Pilar Herrero

Organization

OTM (On The Move) is a federated event involving a series of major international conferences and workshops. These proceedings contain the papers presented at the OTM 2011 Federated conferences, consisting of three conferences, namely, CoopIS 2011 (Cooperative Information Systems), DOA-SVI 2011 (Secure Virtual Infrastructures), and ODBASE 2011 (Ontologies, Databases and Applications of Semantics).

Executive Committee

General Co-chairs

Robert Meersman	VU Brussels, Belgium
Tharam Dillon	Curtin University of Technology, Australia
Pilar Herrero	Universidad Politécnica de Madrid, Spain

OnTheMove Academy Dean

Erich Neuhold	University of Vienna, Austria

Industry Case Studies Program Chairs

Hervé Panetto	Nancy University, France
Richard Mark Soley	OMG, USA

CoopIS 2011 PC Co-chairs

Akhil Kumar	Penn State University, USA
Manfred Reichert	University of Ulm, Germany
Qing Li	City University of Hong Kong

DOA-SVI 2011 PC Co-chairs

Beng Chin Ooi	National University Singapore
Ernesto Damiani	Milan University, Italy
Douglas C. Schmidt	SEI, USA
Jules White	Virginia Polytechnic Institute and State University, USA

ODBASE 2011 PC Co-chairs

Manfred Hauswirth	DERI, Ireland
Pascal Hitzler	Kno.e.sis, Wright State University, USA
Mukesh Mohania	IBM India

Publication Chair

Christopher Jones	Curtin University of Technology, Australia

Publicity-Sponsorship Chair

Ana-Cecilia Martinez Barbosa	DOA Institute, Belgium

Logistics Team

Daniel Meersman	Head of Operations
Ana-Cecilia Martinez Barbosa	
Jan Demey	

CoopIS 2011 Program Committee

Marco Aiello
Antonio Ruiz Cortés
Joonsoo Bae
Zohra Bellahsene
Brian Blake
Nacer Boudjlida
James Caverlee
Jorge Cardoso
Francisco Curbera
Vincenzo D'Andrea
Xiaoyong Du
Schahram Dustdar
Kaushik Dutta
Johann Eder
Rik Eshuis
Ling Feng
Renato Fileto
HongGao Harbin
Ted Goranson
Paul Grefen
Michael Grossniklaus
Amarnath Gupta
Mohand-Said Hacid
Jan Hidders
Birgit Hofreiter
Zhixing Huang
Stefan Jablonski
Paul Johannesson
Epaminondas Kapetanios
Dimka Karastoyanova
Rania Khalaf
Hiroyuki Kitagawa
Frank Leymann
Guohui Li
Rong Liu
ZongWei Luo
Sanjay K. Madria
Tiziana Margaria
Leo Mark
Maristella Matera
Massimo Mecella
Jan Mendling
John Miller
Arturo Molina
Nirmal Mukhi
Miyuki Nakano
Moira C.Norrie
Selmin Nurcan
Werner Nutt
Gerald Oster
Hervé Panetto
Cesare Pautasso
Barbara Pernici
Lakshmish Ramaswamy
Stefanie Rinderle-Ma
Shazia Sadiq
Ralf Schenkel
Jialie Shen
Aameek Singh
Jianwen Su
Xiaoping Sun
Susan Urban

Willem-Jan van den Heuvel
Irene Vanderfeesten
François B. Vernadat
Maria Esther Vidal
Barbara Weber
Mathias Weske
Andreas Wombacher
Jian Yang
Xiaoming Yao
Shuigeng Zhou

DOA-SVI 2011 Program Committee

Rafael Accorsi
Moataz Ahmed
Ghazi Alkhatib
Jaiganesh Balasubramanian
Massimo Banzi
Elisa Bertino
Lionel Brunie
Marco Casassa-Mont
Fabio Casati
Frederic Cuppens
Alfredo Cuzzocrea
Schahram Dustdar
Eduardo Fernandez
Elena Ferrari
Alban Gabillon
Chris Gill
Andy Gokhale
Nils Gruschka
James Hill
Patrick Hung
David Jiang
Guoliang Li
Xumin Liu
Joe Loyall
Leszek Maciaszek
Antonio Maña
Priya Narasimhan
Jean-Cristophe Pazzaglia
Nilabja Roy
Joerg Schwenk
Ahmed Serhrouchni
George Spanoudakis
Azzel Taleb-Bendiab
Sumant Tambe
Bhavani Thuraisingham
Setsuo Tsuruta
Sai Wu
Qi Yu
Xiaofang Zhou
Aoying Zhou

ODBASE 2011 Program Committee

Karl Aberer
Divyakant Agrawal
Harith Alani
Sören Auer
Payam Barnaghi
Ladjel Bellatreche
Paul-Alexandru Chirita
Sunil Choenni
Oscar Corcho
Philippe Cudre-Mauroux
Bernardo Cuenca Grau
Emanuele Della Valle
Prasad Deshpande
Jérôme Euzenat
Walid Gaaloul
Aldo Gangemi
Giancarlo Guizzardi
Peter Haase
Harry Halpin
Takahiro Hara
Andreas Harth
Manfred Hauswirth

Martin Hepp
Pascal Hitzler
Andreas Hotho
Prateek Jain
Krzysztof Janowicz
Matthias Klusch
Shin'Ichi Konomi
Manolis Koubarakis
Rajasekar Krishnamurthy
Shonali Krishnaswamy
Reto Krummenacher
Werner Kuhn
Steffen Lamparter
Wookey Lee
Sanjay Madria
Frederick Maier
Mukesh Mohania
Anirban Mondal
Jeff Z. Pan
Kalpdrum Passi
Dimitris Plexousakis
Ivana Podnar Zarko
Axel Polleres
Guilin Qi
Prasan Roy
Sourav S Bhowmick
Satya Sahoo
Nandlal Sarda
Kai-Uwe Sattler
Peter Scheuermann
Christoph Schlieder
Michael Schrefl
Wolf Siberski
Srinath Srinivasa
Heiner Stuckenschmidt
L. Venkata Subramaniam
York Sure
Kunal Verma
Wei Wang
Josiane Xavier Parreira
Guo-Qiang Zhang

Supporting and Sponsoring Institutions

OTM 2011 was proudly supported or sponsored by Vrije Universiteit Brussel in Belgium, Curtin University of Technology in Australia, Universidad Politecnica de Madrid in Spain, the Object Management Group, and Collibra.

Vrije Universiteit Brussel

Curtin University of Technology

OBJECT MANAGEMENT GROUP

Collibra

UNIVERSIDAD POLITECNICA MADRID

Computing for Human Experience: Semantics Empowered Cyber-Physical, Social and Ubiquitous Computing beyond the Web

Amit Sheth

Kno.e.sis, Wright State University, USA

Short Bio

Amit Sheth is an educator, research and entrepreneur. He is the LexisNexis Ohio Eminent Scholar at the Wright State University, Dayton OH. He directs Kno.e.sis - the Ohio Center of Excellence in Knowledge-enabled Computing which works on topics in Semantic, Social, Sensor and Services computing over Web and in social-cyber-physical systems, with the goal of transitioning from information age to meaning age. Prof. Sheth is an IEEE fellow and is one of the highly cited authors in Computer Science (h-index = 67) and World Wide Web. He is EIC of ISI indexed Intl. Journal of Semantic Web & Information Systems (http://ijswis.org), is joint-EIC of Distributed & Parallel Databases, is series co-editor of two Springer book series, and serves on several editorial boards. By licensing his funded university research, he has also founded and managed two successful companies. Several commercial products and many operationally deployed applications have resulted from his R&D.

Talk

“Computing for Human Experience: Semantics empowered Cyber-Physical, Social and Ubiquitous Computing beyond the Web”

Traditionally, we had to artificially simplify the complexity and richness of the real world to constrained computer models and languages for more efficient computation. Today, devices, sensors, human-in-the-loop participation and social interactions enable something more than a “human instructs machine” paradigm. Web as a system for information sharing is being replaced by pervasive computing with mobile, social, sensor and devices dominated interactions. Correspondingly, computing is moving from targeted tasks focused on improving efficiency and productivity to a vastly richer context that support events and situational awareness, and enrich human experiences encompassing recognition of rich sets of relationships, events and situational awareness with spatio-temporal-thematic elements, and socio-cultural-behavioral facets. Such progress positions us for what I call an emerging era of “computing for human experience” (CHE). Four of the key enablers of CHE are: (a) bridging the physical/digital (cyber) divide, (b) elevating levels of abstractions and utilizing vast background knowledge to enable integration of machine and human perception, (c) convert raw data and observations, ranging from sensors to social media, into understanding of events and situations that are meaningful to humans, and (d) doing all of the above at massive scale covering the Web and pervasive computing supported humanity. Semantic Web (conceptual models/ontologies and background knowledge, annotations, and reasoning) techniques and technologies play a central role in important tasks such as building context, integrating online and offline interactions, and help enhance human experience in their natural environment.

Privacy and the Cloud

Siani Pearson
Hewlett-Packard Laboratories

Short Bio

Siani Pearson is a senior researcher in the Cloud and Security Research Lab (HP Labs Bristol, UK), HP's major European long term applied research centre. She has an MA in Mathematics and Philosophy from Oxford and a PhD in Artificial Intelligence from Edinburgh. She was a Fellow at the Computer Lab in Cambridge University, and for the last 17 years has worked at HP Labs in a variety of research and development programs including collaborations with HP business units and EU PRIME (Privacy and Identity Management for Europe) project.

Siani's current research focus is on privacy enhancing technologies, accountability and the cloud. She is a technical lead on regulatory compliance projects with HP Privacy Office and HP Enterprise Services, and on the collaborative TSB-funded EnCoRe (Ensuring Consent and Revocation) project.

Talk

"Privacy and the Cloud"

Cloud computing offers a huge potential both for efficiency and new business opportunities (especially in service composition), and is almost certain to deeply transform our IT. However, the convenience and efficiency of this approach comes with a range of potential privacy and security risks. Indeed, a key barrier to the widespread uptake of cloud computing is the lack of trust in clouds by potential customers. This concern is shared by experts: the European Network and Information Security Agency (ENISA)'s cloud computing risk assessment report states "loss of governance" as a top risk of cloud computing, and "data loss or leakages" is one of the top seven threats the Cloud Security Alliance (CSA) lists in its Top Threats to Cloud Computing report.

In this talk I will assess how privacy, security and trust issues occur in the context of cloud computing and explain how complementary regulatory, procedural and technical provisions can be used to help address these issues. In particular, accountability is likely to become a core concept in both the cloud and in new mechanisms that help increase trust in cloud computing. It is especially helpful for protecting sensitive or confidential information, enhancing consumer trust, clarifying the legal situation in cloud computing, and facilitating cross-border data transfers. I will also talk about some of the innovative technical solutions that we are developing in HP Labs to enhance privacy in the cloud.

The Social Compute Unit

Schahram Dustdar
Vienna University of Technology (TU Wien)

Short Bio

Schahram Dustdar (ACM Distinguished Scientist), is full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Vienna University of Technology (TU Wien).

From 1999 - 2007 he worked as the co-founder and chief scientist of Caramba Labs Software AG in Vienna (acquired by Engineering NetWorld AG), a venture capital co-funded software company focused on software for collaborative processes in teams. He is Editor in Chief of Computing (Springer) and on the editorial board of IEEE Internet Computing, as well as author of some 300 publications.

Talk

"The Social Compute Unit"

Social computing is perceived mainly as a vehicle for establishing and maintaining social (private) relationships as well as utilizing political and social interests. Unsurprisingly, social computing lacks substantial adoption in enterprises. Clearly, collaborative computing is firmly established (as a niche), but no tight integration exists of social and collaborative computing approaches to facilitate mainstream problem solving in and between enterprises or teams of people. In this talk I will present a fresh look at this problem and examine how to integrate people in the form of human-based computing and software services into one composite system, which can be modeled, programmed, and instantiated on a large scale.

GENI - Global Environment for Network Innovations

Niky Riga

GENI Project Office, Raytheon BBN Technologies

Short Bio

Niky Riga is a Network Scientist at Raytheon BBN Technologies. Niky joined the GENI Project Office (GPO) in March 2010. As a member of GPO, Niky is responsible for supporting GENI experimenters in integrating and deploying their experiments as well as advocating their requirements to the rest of the GENI community.

Before joining the GPO, Niky worked on multiple innovative projects within the Network Research department of BBN. Her focus is on designing and prototyping pioneering transport services for Mobile Ad-hoc Networks, while her main goal is making innovative, research ideas practical and implementing them on real systems. She has successfully led various integration efforts. Niky earned a Diploma in Electrical and Computer Engineering at the National Technical University of Athens, and an MS degree in Computer Science at Boston University.

Talk

"GENI - Global Environment for Network Innovations"

The Global Environment for Network Innovations (GENI) is a suite of research infrastructure components rapidly taking shape in prototype form across the US. It is sponsored by the US National Science Foundation, with the goal of becoming the world's first laboratory environment for exploring future Internets at scale, promoting innovations in network science, security, technologies, services, and applications.

GENI allows academic and industrial researchers to perform a new class of experiments that tackle critically important issues in global communications networks such as (a) Science issues: we cannot currently understand or predict the behavior of complex, large-scale networks, (b) Innovation issues: we face substantial barriers to at-scale experimentation with new architectures, services, and technologies (c) Society issues: we increasingly rely on the Internet but are unsure that can we trust its security, privacy or resilience GENI is enabling

researchers to explore these issues by running large-scale, well-instrumented, end-to-end experiments engaging substantial numbers of real users. These experiments may be fully compatible with today's Internet, variations or improvements on today's Internet protocols, or indeed radically novel, clean slate designs. The GENI project is now supporting such experiments across a mesoscale build-out through more than a dozen US campuses, two national backbones, and several regional networks. If this effort proves successful, it will provide a path toward more substantial build-out.

In this keynote presentation, she will introduce GENI through a couple of example use-cases, she will review the growing suite of infrastructure and evolving control framework. She will also present previous and current experiments running in GENI.

Table of Contents – Part II

Table of Contents – Part I

DOA-SVI 2011 PC Co-chairs' Message

Welcome to the 12th International Symposium on Distributed Objects, Middleware and Applications (DOA 2010), held in Crete, Greece, October 2010. The DOA conference series has become a key forum for presenting and discussing new perspectives and exciting research results. Over the years, DOA topics have included new computing paradigms like Web Services and SOA, cloud computing, virtualization, and many others, dealing with new new problems and ideas as well as with the design and deployment of practical applications. DOA has always managed to reach the right balance between theoretical and practical aspects of IT research. Hopefully, this year is no exception. Much interest has focused on the emerging paradigm of cloud computing, with papers covering topics as diverse as virtualization management, cloud-based service environments and cloud encryption and security; but other key topics of distributed computing, especially the ones related to services, continued to be well represented among DOA submissions. Contributions based on experimental work were particularly encouraged, with early results also accepted where they were considered to be sufficiently important to a wider audience. The quality of submissions this year was very high, again following a well-established DOA tradition. All of the papers passed through a rigorous selection process, with at least three reviewers per paper and much discussion on the relative merits of accepting each paper throughout the process. At the end, we decided to accept 12 regular papers of the original 27 submissions. Two more submissions were accepted as posters. Putting together a conference like DOA is always a team effort, and many different contributions need to be acknowledged. First of all we would like to gratefully acknowledge the work of all of the authors, whether or not their paper was accepted. Thanks for choosing DOA to present your research work. Secondly, we are grateful to the dedicated work of the leading experts in the field from all over the world who served on the Program Committee and whose names appear in the proceedings. Thanks for helping us in putting together an excellent program. Finally, we would like to thank the whole OTM team for its support and guidance, including the General Co-chairs Tharam Dillon and Robert Meersman, the Publication Chairs, and the secretariat. The proceedings you hold in your hand are the result of the hard work that everyone has put into DOA. We hope you enjoy them and maybe consider submitting something in the future.

August 2011

Ernesto Damiani
Doug Schmidt
Beng Chin Ooi

Optimizing Integrated Application Performance with Cache-Aware Metascheduling*

Brian Dougherty[1], Jules White[1], Russell Kegley[2], Jonathan Preston[2], Douglas C. Schmidt[3], and Aniruddha Gokhale[3]

[1] Virginia Tech
[2] Lockheed Martin Aeronautics
[3] Vanderbilt University
{brianpd,julesw}@vt.edu,
{d.schmidt,a.gokhale}@vanderbilt.edu,
{russell.b.kegley,jonathan.d.preston}@lmco.com

Abstract. Integrated applications running in multi-tenant environments are often subject to quality-of-service (QoS) requirements, such as resource and performance constraints. It is hard to allocate resources between multiple users accessing these types of applications while meeting all QoS constraints, such as ensuring users complete execution prior to deadlines. Although a processor cache can reduce the time required for the tasks of a user to execute, multiple task execution schedules may exist that meet deadlines but differ in cache utilization efficiency. Determining which task execution schedules will utilize the processor cache most efficiently and provide the greatest reductions in execution time is hard without jeopardizing deadlines.

The work in this paper provides three key contributions to increasing the execution efficiency of integrated applications in multi-tenant environments while meeting QoS constraints. First, we present cache-aware metascheduling, which is a novel approach to modifying system execution schedules to increase cache-hit rate and reduce system execution time. Second, we apply cache-aware metascheduling to 11 simulated software systems to create 2 different execution schedules per system. Third, we empirically evaluate the impact of using cache-aware metascheduling to alter task schedules to reduce system execution time. Our results show that cache-aware metascheduling increases cache performance, reduces execution time, and satisfies scheduling constraints and safety requirements without requiring significant hardware or software changes.

1 Introduction

Current trends and challenges. Multi-tenant environments, such as Software-as-a-Service (SaaS) platforms and integrated avionics systems, are often subject to stringent quality-of-service (QoS) requirements, such as resource requirements and performance constraints [29]. To ensure that response time specified by service-level agreements (SLAs) are upheld, execution time must be minimized.

* This work was sponsored in part by the Air Force Research Lab.

One approach to reduce execution time is to reduce the time spent loading data from memory by efficiently utilizing processor caches.

Several research techniques utilize processor caches more efficiently to reduce execution time. For example, Bahar et al. [5] examined several different cache techniques for reducing execution time by increasing cache hit rate. Their experiments showed that efficiently utilizing a processor cache can result in as much as a 24% reduction in execution time. Likewise, Manjikian et al. [16] demonstrated a 25% reduction in execution time as a result of modifying the source-code of the executing software to use cache partitioning.

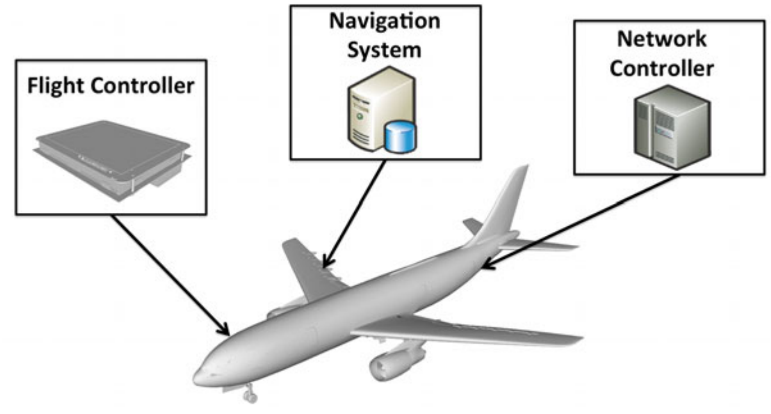


Fig. 1. Example of an Integrated Avionics Architecture

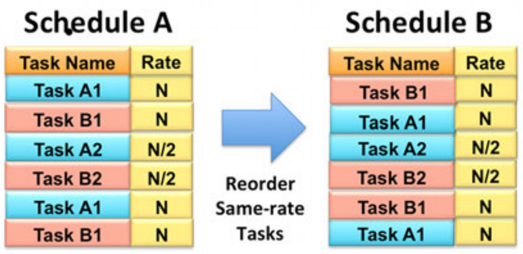
Many optimization techniques [21,17,27] increase cache hit rate by enhancing source code to increase *temporal locality* of data accesses, which defines the proximity with which shared data is accessed in terms of time [13]. For example, loop interchange and loop fusion techniques can increase temporal locality of accessed data by modifying application source code to change the order in which application data is written to and read from a processor cache [13,16]. Increasing temporal locality increases the probability that data common to multiple tasks persists in the cache, thereby reducing cache-misses and software execution time [13,16].

Open problem ⇒ Increasing cache hit rate of integrated applications without source code modifications. *Integrated applications* are a class of systems consisting of a number of computing modules capable of supporting numerous applications of differing criticality levels [14]. Like other multi-tenant environments, integrated applications prohibit data sharing between multiple concurrent users, while requiring that execution completes within predefined deadlines.

Software architectures for integrated applications are built from separate components that must be scheduled to execute in concert with one another. Prior work has generally focused on source-code level modifications for individual applications instead of integrated applications, which is problematic for multi-tenant environments built from multiple integrated applications. Systems based on the integration of multiple applications (such as the integrated avionics architecture shown in Figure 1) often prohibit code-level modifications due to restricted access to proprietary source code and the potential to violate safety certifications [23] by introducing overflow or other faulty behavior.

Solution approach → Heuristic-driven schedule alteration of same-rate tasks to increase cache hit rate. Priority-based scheduling techniques can help ensure software executes without missing deadlines. For example, rate-monotonic scheduling [20] is a technique for creating task execution schedules that satisfy timing constraints by assigning priorities to tasks based on the task periodicity and ensuring utilization bounds are not exceeded. These tasks are then split into sets that contain tasks of the same priority/rate.

Rate monotonic scheduling specifies that tasks of the same rate can be scheduled arbitrarily [8] as long as priority inversions between tasks are not introduced. Figure 2 shows two different valid task execution schedules generated with rate monotonic scheduling. Since task A2 and task B2 share the same priority, their execution order can be swapped without violating timing constraints. This paper shows how to improve cache hit rates for systems built from multiple integrated applications by intelligently ordering the execution of tasks within the same rate to increase temporal locality of task data accesses. We refer to this technique as *metascheduling*, which involves no source code modifications.



Schedule A

Task Name	Rate
Task A1	N
Task B1	N
Task A2	N/2
Task B2	N/2
Task A1	N
Task B1	N

Schedule B

Task Name	Rate
Task B1	N
Task A1	N
Task A2	N/2
Task B2	N/2
Task B1	N
Task A1	N

Reorder
Same-rate
Tasks

Fig. 2. Valid Task Execution Schedules

This paper presents *cache-aware metascheduling*, which is a novel scheduling optimization technique we developed to improve cache effects of integrated applications without violating scheduling constraints or causing priority inversions. Since this technique requires no source code modifications, it can be applied to integrated applications without requiring source software permissions or completely invalidating safety certifications.

This paper provides the following contributions to R&D on scheduling optimizations to increase the cache hit rate of integrated applications:

- We present a metascheduling technique that satisfies scheduling constraints and safety requirements, increases cache hits, and requires no new hardware or software.

- To motivate the need for scheduling enhancements to improve cache hit rate in integrated applications, we present an industry case study of an integrated avionics system in which modifications to its constituent applications are prohibitively expensive due to safety (re)certification requirements.
- We present empirical results of 2 task execution schedules performance and demonstrate that applying cache-aware metascheduling can result in increased cache-hit rates and reduced system execution time.

Paper organization. The remainder of the paper is organized as follows: Section 2 examines an integrated avionics system designed to meet scheduling deadlines and assure required safety constraints; Section 3 summarizes the challenges of creating a metric that predicts integrated application performance at design time and guides execution schedule modifications; Section 4 describes a cache-aware metascheduling strategy for increasing cache hit-rate and reducing execution time of an integrated avionics system; Section 5 analyzes empirical results that demonstrate the effectiveness of cache-aware metascheduling for increasing cache hit-rate and reducing system execution time; Section 6 compares our cache-aware metascheduling approach with related work; and Section 7 presents concluding remarks.

2 Integrated Avionics System Case Study

This section presents a case study representative of an integration avionics system provided by Lockheed Martin that shows how integrated applications are configured in modern aircraft, such as the one shown in Figure 1. This case study underscores the similarity between multi-tenant environments and integrated applications in terms of response time requirements and data sharing restrictions. It also shows how scheduling methods for integrating applications can be applied to ensure safety constraints and scheduling requirements are met. Section 5.2 describes modifications to this method that increase the cache hit-rates of integrated application architectures.

In this architecture, task execution schedules are divided into frames in which a subset of tasks execute. Two tasks are schedule to execute sequentially at the start of each base frame. The task that executes at the base frame rate is scheduled to run, followed by another task at a rate of lower frequency. For example, at Frame 0 the scheduler will execute the software that runs at 75 Hz and the software that executes at 37.5 Hz, or half as frequently. This pattern continues repeatedly until the lowest rate software in the system has completed. All scheduling of integrated application tasks in the avionics system occurs in this manner.

One method for ensuring that tasks execute with a predetermined frequency is to set a base execution rate and then set all other tasks to execute proportionately often. The execution of the tasks can the be interleaved based on the relative execution rate, as shown in Figure 3. Applications 1 and 2 both have tasks that execute at rates N, N/2, and N/4. The rate N tasks from both applications always execute before any other tasks in a given frame. Although it is

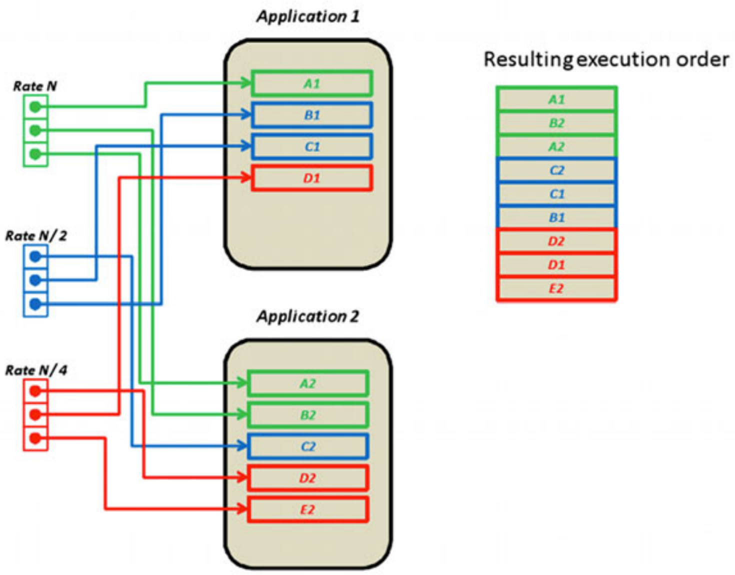


Fig. 3. Interleaved Execution Order is Repeatable

not necessarily the case that all rate N tasks from Application 1 will run before the rate N tasks from Application 2, our case study makes this order repeatable, *i.e.*, the interleaving A1/B2/A2 will not change from frame to frame after it is established when the system starts.

3 Challenges of Analyzing and Optimizing Integrated Applications for Cache Effects

This section presents the challenges faced by system integrators who attempt to optimize integrated applications to improve cache hit rate. Systems are often subject to multiple design constraints, such as safety requirements and scheduling deadlines, that may restrict which optimizations are applicable. This section describes three key challenges that must be overcome to optimize application integration by improving the cache hit rate of integrated applications.

Challenge 1: Altering application source code may invalidate safety certification. Existing cache optimization techniques, such as loop fusion and data padding [12,19], increase cache hit rate but requiring application source code modifications, which may invalidate previous safety certifications by introducing additional faults, such as overflow. Re-certification of integrated applications is a slow and expensive process, which increases cost and delays deployment. Proprietary source code also may not be accessible to integrators. Even if source code is available, moreover, integrators may not have the expertise required to make reliable modifications. What is needed, therefore, are techniques that improve cache hit rates without modifying integrated application software.

Challenge 2: Optimization techniques must satisfy scheduling constraints. Integrated applications are often subject to scheduling constraints and commonly use priority-based scheduling methods, such as rate monotonic scheduling, to ensure that software tasks execute predictably [28,10]. These constraints prohibit many simple solutions that ignore task priority, such as executing all task sets of each application, that would greatly increase cache hit-rate. These techniques can cause integrated applications to behave unpredictably, with potentially catastrophic results due to missed deadlines and priority inversions. What is needed, therefore, are techniques that can be applied and re-applied when necessary to increase the cache hit-rate and decrease integrated application execution time without violating timing constraints.

Challenge 3: System complexity and limited access to source code. Current industry practice [3] for increasing cache hit rate require collecting detailed, instruction-level information that describe integrated application behavior with respect to the memory subsystem and data structure placement. Obtaining information of this granularity, however, can be an extremely laborious and time consuming for large-scale systems, such as integrated avionics systems containing millions of lines of codes and dozens of integrated applications. Moreover, these large-scale systems may be so complex that it is not feasible to collect this information.

System integrators can more easily obtain higher level information, such as the percentage of total memory accesses made by a given task. What is needed, therefore, are techniques that allow system integrators to increase the cache hit rate of integrated applications without requiring intricate, low-level system knowledge.

4 Improving Cache Hit Rate via Cache-Aware Metascheduling

This section presents cache-aware metascheduling, which is a technique we developed to increase cache hit rate through re-ordering the execution schedule of same-rate tasks of integrated applications. Cache-ware metascheduling can potentially increase the cache hit-rate and reduce execution time of systems in which resources are not shared between concurrent executions, such as multi-tenant environments and integrated avionics systems.

4.1 Re-ordering Same-Rate Tasks with Cache-Aware Metascheduling

Rate monotonic scheduling can be used to create task execution schedules for integrated applications that ensure scheduling deadlines are met. This technique, however, allows the definition of additional rules to determine the schedule of same-rate tasks [18,4,15]. As shown in Figure 4, reordering same-rate tasks, or metascheduling, can produce multiple valid execution schedules.

For example, Figure 4 shows how Task A1 can execute before or after Task B1. Either ordering of these same rate tasks meets scheduling constraints. Since the original schedule satisfies constraints and reordering same rate tasks does not introduce priority inversions, schedules generated by metascheduling are valid. Moreover, metascheduling does not require alterations to application source code or low-level system knowledge.

The motivation behind metascheduling is that although different execution orders of same-rate tasks do not violate scheduling constraints, they can impact the cache hit-rate. For example, if two same-rate tasks that share a large amount of data execute sequentially, then the first task may “warm up” the cache for the second task by preloading data needed by the second task. This type of cache warming behavior can improve the cache hit rate of the second task.

Same-rate task orderings can also negatively affect cache hit rate. For example, tasks from integrated applications often run concurrently on the same processor. These tasks may be segregated into different processes, however, preventing tasks from different applications from sharing memory. If two tasks do not share memory there is no cache warmup benefit. Moreover, the first task may write a large amount of data to the cache and evict data needed by the second task from the cache, reducing the cache hit rate of the second task.

Cache-aware metascheduling is the process of reordering the execution of same-rate tasks to increase beneficial cache effects, such as cache warm up, and reduce negative effects, such as requiring reading data from main memory. Cache-aware metascheduling is relatively simple to implement, does not require in-depth knowledge of the instruction level execution details and memory layout of a large-scale system, and can be achieved without source code modifications to tasks, making it ideal for increasing the performance of existing integrated architectures and multi-tenant systems. Section 5 shows that reordering same-rate tasks does improve cache hit rates and reduce execution time. A key question, however, is what formal metric can be used to choose between multiple potential same-rate task execution schedules.

4.2 Deciding between Multiple Metaschedules

While cache-aware metascheduling can be used to produce multiple valid same-rate task execution schedules, it is not always apparent which schedule will produce the overall best hit-rate and application performance. For example, Figure 4 shows a schedule generated with rate monotonic scheduling and two additional valid schedules created by permuting the ordering of same-rate tasks for a flight controller (FC) application and a targeting system (TS) application.

The only difference between the task execution schedules are the order in which tasks of the same-rate are executed.

It is not obvious which task execution schedule shown in Figure 4 will produce the best cache hit-rate. For example, Metaschedule 2 in Figure 4 shows 2 tasks of Application FC executing sequentially, while no tasks of Application TS execute sequentially. If the tasks in Application FC share a large amount of data temporal locality should increase compared to the original schedule since the cache is “warmed up” for the execution of FC1 by FC2.

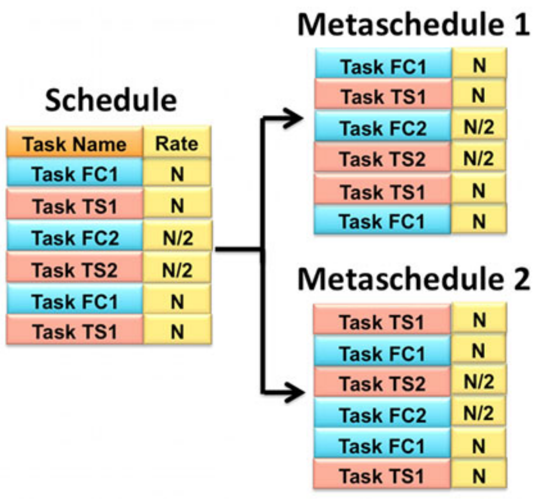


Fig. 4. Multiple Execution Schedules

In Metaschedule 1, however, 2 tasks of Application TS execute sequentially while no tasks of Application FC execute sequentially. If Application TS shares more data than Application FC, Metaschedule 1 will yield greater temporal locality than both the original schedule and schedule FC since the cached will be warmed up with more data. It may also be the case that no data is shared between any tasks of any application, in which case all three schedules would yield similar temporal locality and cache hit rates.

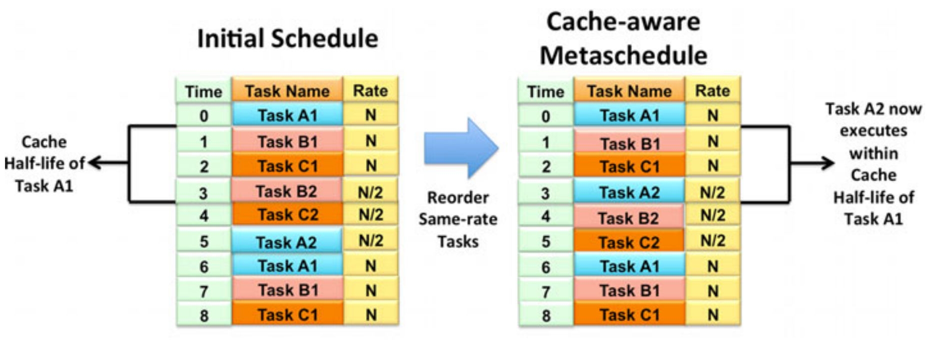
Figure 4 shows it is hard to decide which schedule will yield the highest cache hit rate. Constructing a metric for estimating temporal locality of a task execution schedules could provide integrated application developers with a mechanism for comparing multiple execution schedules and choosing which one would most yield the highest cache hit rate. It is hard to estimate temporal locality, however, due to several factors, such as the presence and degree of data sharing between tasks.

4.3 Using Cache-Half Life to Drive Cache-Aware Metascheduling

While metascheduling can be used to produce new execution schedules that continue to meet scheduling constraints, many of these schedules will not improve, and may even reduce the cache-hit rate and increase execution time of the system. We define a new heuristic, referred to as the cache half-life that can be used to drive the metascheduling process.

Cache Half-Life. We now explain the key factors that impact cache hit rate in integrated architectures and multi-tenant systems. A beneficial effect occurs when task T1 executes before task T2 and loads data needed by T2 into the cache. The beneficial effect can occur if T1 and T2 execute sequentially or if any intermediate task executions do not clear out the data that T1 places into the cache that is used by T2. The *cache half-life* is this window of time between which T1 and T2 can execute before the shared data is evicted from the cache by data used for intermediate task executions. While this model is simpler than the actual complex cache data replacement behavior, it is effective enough to give a realistic representation of cache performance [22].

For example, assume there are 5 applications, each consisting of 2 tasks, with each task consuming 20 kilobytes of memory in a 64k cache. The hardware uses a *Least Recently Used* (LRU) replacement policy, which replaces the cache line that remained the longest without being read when new data is written to the cache. The cache half-life formulation will differ for other cache replacement policies.


Initial Schedule
Cache-aware Metaschedule
Cache Half-life of Task A1
Reorder Same-rate Tasks
Task A2 now executes within Cache Half-life of Task A1

Initial Schedule

Time	Task Name	Rate
0	Task A1	N
1	Task B1	N
2	Task C1	N
3	Task B2	N/2
4	Task C2	N/2
5	Task A2	N/2
6	Task A1	N
7	Task B1	N
8	Task C1	N

Cache-aware Metaschedule

Time	Task Name	Rate
0	Task A1	N
1	Task B1	N
2	Task C1	N
3	Task A2	N/2
4	Task B2	N/2
5	Task C2	N/2
6	Task A1	N
7	Task B1	N
8	Task C1	N

Fig. 5. Using Cache Half-life to Drive Cache-aware Metascheduling

Executing the tasks will require writing up to 200 kilobytes to cache. Since the cache can only store 64 kilobytes of data, all data from all applications cannot persist in the cache simultaneously. Assuming the cache is initially empty, it would take a minimum of 4 task executions writing 20 kilobytes each before any data written by the first task potentially becomes invalidated. This system would therefore have a cache half-life of 4.

Our cache-aware metascheduling algorithm uses the cache half-life to increase the cache hit-rate and reduce system execution time. We attempt to maximize the number of tasks of the same application that execute before the cache half-life of the initial execution task expires.

For example, as shown in Figure 5 task A1 of Application ’A’ executes at timestamp 0. The cache half-life of task A1 is 3 timestamps. As a result, for at least timestamps 2-4 data from Application ’A’ will persist in the cache. Any tasks that share data with task A1 could use the data stored in the cache rather than accessing the data from main memory, resulting in a cache hit and reducing system execution time.

In this example, moving Task A2 from timestamp 5 to timestamp 3 will give Task A2 the opportunity to take advantage of the data cached by Task A1, resulting in a potentially higher cache hit-rate without violating scheduling constraints. Conversely, moving Task A2 to timestamp 4 will not increase the cache hit-rate as most or all of the data written by Task A1 to the cache will have been overwritten by this point due to tasks of other applications executing.

5 Empirical Results

This section analyzes the results of a performance analysis of integrated applications in multiple systems with different execution schedules generated through metascheduling. These systems also differ in the amount of memory shared between tasks. We investigate the impact of cache-aware metascheduling on L1 cache misses and runtime reductions for each system.

5.1 Overview of the Hardware and Software Testbed

To examine the impact of cache-aware metascheduling on integrated application performance, we collaborated with members of the Lockheed Martin Corporation to generated multiple systems that mimic the scale, execution schedule and data sharing of modern flight avionics systems. We specified the number of integrated applications, number of tasks per application, the distribution of task priority, and the maximum amount of memory shared between each task for each system. Together these integrated applications comprise a representative avionics system. We also developed a Java-based code generator to synthesize C++ system code that possessed these characteristics.

Figure 6 shows how the generated systems included a priority-based scheduler and multiple sample integrated applications that consisted of a variable number of periodic avionic tasks. Rate monotonic scheduling was used to create a deterministic priority based schedule for the generated tasks that adheres to rate

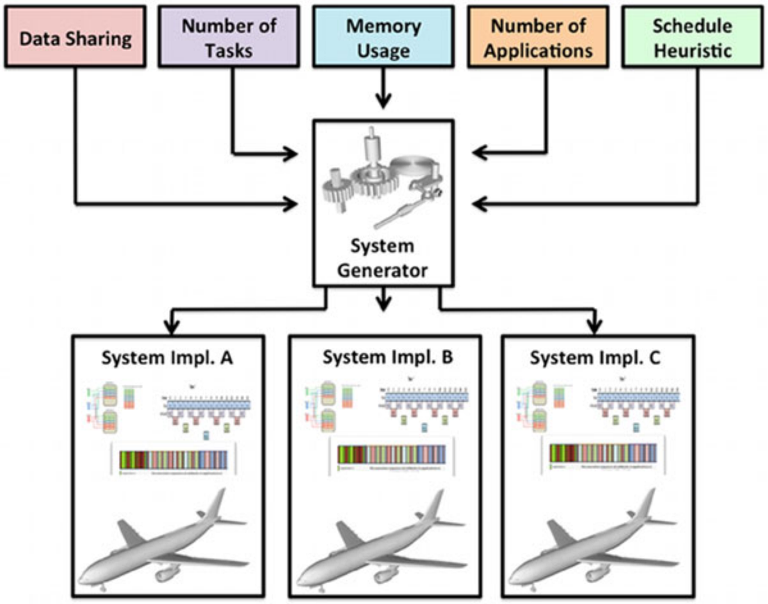


Fig. 6. System Creation Process

monotonic scheduling requirements. The systems then were compiled and executed on a Dell Latitude D820 with a 2.16Ghz Intel Core 2 processor with 2 x 32kb L1 instruction caches, 2 x 32 kb write-back data caches, a 4 MB L2 cache and 4GB of RAM running Windows Vista.

For each experiment, every system was executed 50 times to obtain an average runtime. The cache performance of these executions were profiled using the Intel VTune Amplifier XE 2011. VTune is a profiling tool that is capable of calculating the total number of times an instruction is executed by a processor.

For example, to determine the L1 cache misses of System A, we compiled and then executed it with VTune configured to return the total times that the instruction MEM_LOAD_REQUIRED.L1D_MISS is called. The data sharing and memory usage of these integrated applications, as well as the metacheduling strategy, are all parameterized and varied to generate a range of test systems. We use these simulated systems to validate cache-aware metascheduling by showing that taking into account data sharing when selecting a metaschedule performance in terms of execution time and cache misses.

System Size vs. Cache Size. The amount of memory required for the system has a major impact on the caching efficiency of the system. For example, consider a system that requires 6 kilobytes of memory executing on a processor with an L1 cache of 60 kilobytes. Assuming this is the sole executing system, all of data can be stored in the cache, leaving 90% of the cache free. The cache effects would therefore be the same for any system that does not require memory that exceeds the available memory in the cache.

Cache-aware metascheduling currently does not take into account available cache size since this may vary drastically from platform to platform. For our experiments, we require that memory requirements of all generated software systems exceed the memory available in the cache. Otherwise, the cache could store all of the data used by all applications simultaneously, removing any contention for cache storage space, which is unrealistic in industry systems.

Data Sharing Characteristics. The data shared between applications and shared between tasks of the same integrated application can greatly impact the cache effectiveness of a system. For example, the more data shared between two applications, the more likely the data in the cache can be utilized by tasks of the applications, resulting in reduced cache misses and faster system runtime. The system described in Section 2 prohibits data sharing between tasks of different integrated applications.

All systems profiled in this section are also restricted to sharing data between tasks of the same application. Integrated applications that exchange a great deal of common message data, however, are likely to share memory. To account for this sharing, our future work is examining cache-aware metascehduling strategies that account for these architectures.

Task Execution Schedule. The execution schedule of the software tasks of the system can potentially affect system performance. For example, assume there are two integrated applications named App1 and App2 that do not share data.

Each application contains 1,000 task methods, with tasks of the same application sharing a large amount of data. The execution of a single task stores enough memory to completely overwrite any data in the cache, resulting in a cache half-life of 1.

When a task from App1 executes it completely fills the cache with data that is only used by App1. If the same or another task from App1 executes next, data could reside in the cache that could potentially result in a cache hit. Since no data is shared with App2, however, executing a task from App2 could not result in a cache hit and would overwrite all data used by App1 in the class. We predict that multiple execution schedules therefore effect performance differently in terms of cache hit-rate and execution time.

5.2 Experiments: Determining the Impact of Cache-Aware Metascheduling on Cache Hit-Rate and Runtime Reductions

Experiment design. The execution schedule of tasks can potentially impact both the runtime and number of cache misses of a system. We manipulated the execution order of a single software system with 20% shared data probability between 5 applications consisting of 10 tasks each to create 2 new execution schedules. First, rate monotonic scheduling was use to create the baseline schedule. This cache-aware metascheduling was then applied to reorder same rate tasks to increase the temporal proximity between executions of tasks that share data to the Optimized schedule.

Experiment 1: Using Cache-Aware Metascheduling to Reduce Cache Misses. This experiment measures the impact of applying cache-aware

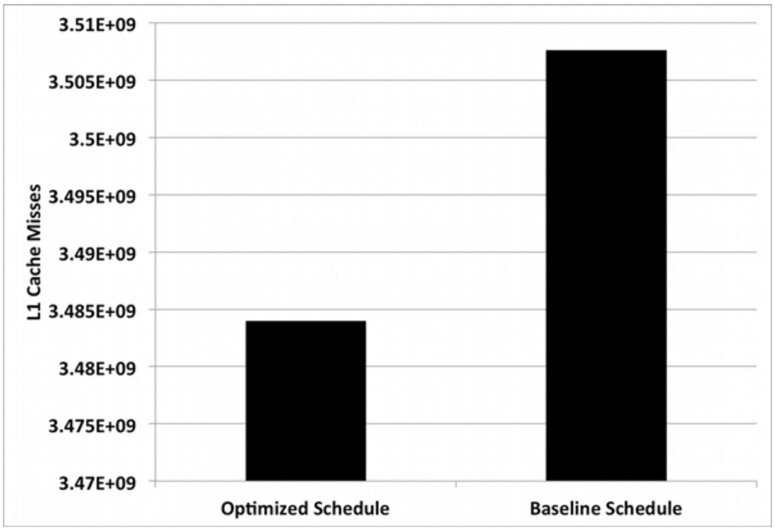


Fig. 7. Execution Schedules vs L1 Cache Misses

metascheduling on the total L1D cache misses generated by an execution schedule.

Hypothesis: Increasing temporal locality through cache-aware metascheduling will result in less cache misses. Altering the task execution schedule can raise or lower the temporal locality of a sequence of data accesses. This change in temporal locality could potential affect the cache hit-rate resulting from executing a specific schedule. One way to potentially raise temporal locality is to increase the instances in which a task executes before the cache half-life of a previous task with which it shares memory expires. We hypothesize that increasing temporal locality through cache-aware metascheduling will result in less cache misses.

Experiment 1 Results. We hypothesized that using cache-aware metascheduling to increase temporal locality would reduce the number of cache misses. Figure 7 shows the L1 cache misses for both execution schedules. The baseline execution schedule resulted in $3.5076x10^9$ L1 cache misses while the Optimized execution schedule generated $3.484x10^9$ cache misses. Therefore, this data validates our hypothesis that cache miss rates can be reduced by using cache-aware metascheduling to increase temporal locality.

Experiment 2: Reducing Execution Time with Cache-Aware Metascheduling. This experiment measures and compares the total execution time of a system execution schedule generated with rate monotonic scheduling and the schedule resulting from applying cache-aware metascheduling.

Hypothesis: Using cache-aware metascheduling to increase temporal locality of schedule will reduce execution time. While Experiment 1 showed that applying cache-aware metascheduling can reduce cache misses, the impact of cache-aware metascheduling on system execution time remains unclear. We hypothesize that the schedule generated with cache-aware metascheduling will execute faster than the schedule generated with normal rate monotonic scheduling.

Experiment 2 Results. Figure 8 shows the average runtimes for the different execution schedules. As shown in this figure, the task execution order can have a large impact on runtime. The baseline execution schedule executed in 3,374 milliseconds. The Optimized execution schedule completed in 3,299 milliseconds, which was an 2.22% reduction in execution time from the baseline execution schedule. These results demonstrate that applying cache-aware metascheduling can reduce the total execution time of a schedule.

Experiment 3: Impact of Data Sharing on Cache-Aware Metascheduling Effectiveness. This experiment measures the impact of data sharing on execution time reductions due to cache-aware metascheduling.

Hypothesis: Applying cache-aware metascheduling will reduce execution time for all levels of data sharing. Figure 8 shows the execution

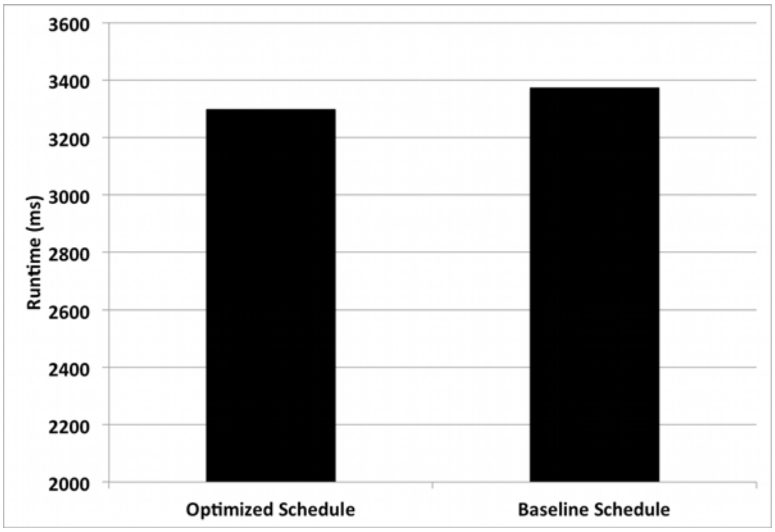


Fig. 8. Runtimes of Various Execution Schedules

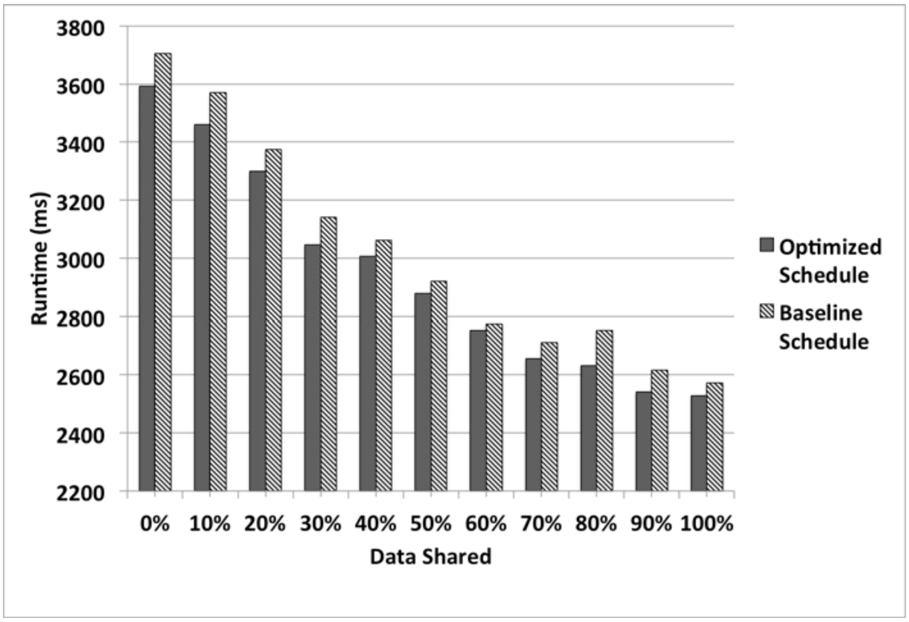


Fig. 9. Runtimes of Multiple Levels of Data Sharing

time of two execution schedules at only 20% data sharing. Data sharing of industry systems, however, may vary to a large extent. Therefore, we created 10 other systems with different data sharing characteristics. We hypothesize that cache-aware metascheduling will lead execution time reductions regardless of the amount of data shared between tasks.

Experiment 3 Results. The execution time for the baseline and Optimized schedules is shown in Figure 9. The Optimized schedule consistently executed faster than the baseline schedule with an average execution time reduction of 2.54% without requiring alteration to application source-code and without violating real-time constraints. Moreover, this reduction required no purchasing nor implementing of any additional hardware or software or obtaining any low-level knowledge of the system. These results demonstrate that cache-aware metascheduling can be applied to reduce the execution time of an array of systems, such as integrated avionics systems, regardless of cost constraints, restricted access to software source code, real-time constraints, or instruction level-knowledge of the underlying architecture.

6 Related Work

This section compares the cache-aware metascheduling and its use for cache optimization with other techniques for optimizing cache hits and system performance, including (1) software cache optimization techniques and (2) hardware cache optimization techniques.

Software cache optimization techniques. Many techniques change the order in which data is accessed to increase the effectiveness of processor caches by altering software at the source code level. These optimizations, known as data access optimizations [13], focus on changing the manner in which loops are executed. One technique, known as loop interchange [30], can be used to reorder multiple loops to maximize the data access of common elements in respect to time, referred to as *temporal locality* [2,31,30,24].

Another technique, known as loop fusion [25], is often applied to further increase cache effectiveness. Loop fusion maximizes temporal locality by merging multiple loops into a single loop and altering data access order [25,12,6]. Yet another technique for improving software cache effectiveness is to utilize *prefetch* instructions, which retrieves data from memory into the cache before the data is requested by the application [13]. Prefetch instructions inserted manually into software at the source code level can significantly reduce memory latency and/or cache miss rate [7,9].

While these techniques can increase the effectiveness of software utilizing processor caches, they all require source code optimizations. Many systems, such as the avionic system case study described in Section 2, are safety critical and must undergo expensive certification and rigorous development techniques. Any alteration to these applications can introduce unforeseen side effects and invalidate the safety certification. Moreover, developers may not have source code proprietary applications that are purchased. These restrictions prohibit the use of any code-level modifications, such as those used in loop fusion and loop interchange, as well as manually adding prefetch instructions.

These techniques, however, demonstrate the effects of increasing temporal locality on cache effectiveness and performance. cache-aware metascheduling can be used as a heuristic to change the execution order of the software tasks to increase cache effectiveness and performance by ordering the tasks in such a way that temporal locality is increased. The fundamental difference, however, between using cache-aware metascheduling for cache optimization and these methods is that no modifications are required to the underlying software that is executing on the system, thereby achieving performance gains without requiring source code access or additional application re-certification.

Hardware cache optimization techniques. Several techniques alter systems at the hardware level to increase the effectiveness of processor caches. One technique is to alter the *cache replacement policy* processors use to determine which line of cache is replaced when new data is written to the cache. Several policies exist, such as Least Recently Used (LRU), Least Frequently Used (LRU), First In First Out (FIFO), and random replacement [11].

The cache replacement policy can substantially influence DRE system performance. For example, LRU is effective for systems in which the same data will likely be accessed again before enough data has been written to the cache to completely overwrite the cache. Performance gains will be minimal, however, if enough new data is written to the cache such that previously cached data is always overwritten before it can be accessed [26]. In these cases, a random replacement policy may yield the most cache effectiveness [26].

Moreover, certain policies are shown to work better for different cache levels [1], with LRU performing well for L1 cache levels, but not as well for large data sets that may completely exhaust the cache. Unfortunately, it is hard—and often impossible for users—to alter the cache policy of existing hardware. Cache replacement policies should therefore be considered when choosing hardware to maximize the effects of cache optimizations made at the software or execution schedule level.

Cache-aware metascheduling does not alter the cache replacement policy of hardware since altering the hardware could invalidate previous safety certifications, similar to altering software at the source code level. Moreover, cache-aware metascheduling can be used a heuristic to increase temporal locality by altering the task execution order schedule. While many replacement policies exist, the metascheduling strategies we apply assumes an LRU replacement policy. Our future work is examining the impact of cache replacement policy on the performance gains of schedules altered via cache-aware metascheduling.

7 Concluding Remarks

Processor data caching can substantially increase performance of systems (such as integrated applications and other multi-tenant environments) in which SLAs provide response time assurance and QoS policies that restrict resource sharing. It is hard, however, to create valid task execution schedules that increase cache effects and satisfy timing constraints. Metascheduling can be used to generate multiple valid execution schedules with various levels of temporal locality and different cache hit rates.

This paper presents a cache-aware metascheduling to increase the performance gains due to processor caching of integrated applications. We empirically evaluated four task execution schedules generated with cache-aware metascheduling in terms of L1 cache misses and execution time. We learned the following lessons from increasing cache hit-rate with cache-aware metascheduling:

- **Cache-aware metascheduling increases cache hit rate of integrated applications.** Using cache-aware metascheduling led to runtime reductions of as much as 5% without requiring code-level modifications, violating scheduling constraints or implementing any additional hardware, middleware, or software, and thus can be applied to broad range of systems.
- **Relatively minor system knowledge yields effective metascheduling strategies for increasing cache performance.** Developing cache-aware metascheduling strategies does not require an expert understanding of the underlying software. Reasonable estimates of data sharing and knowledge of the executing software tasks are all that is required to determine schedules that yield effective reductions in computation time.
- **Algorithmic techniques to maximize cache-hit rate improvements due to cache-aware metascheduling should be developed.** The task execution schedule was shown to have a large impact on system performance. Our future work is examining algorithmic techniques for optimizing cache-aware metascheduling to determine the optimal execution order for tasks in specific systems (such as multi-tenant environments) to maximize cache hit rate.
- **Cache-aware metascheduling should be applied to cloud-based multi-tenant environments.** Given the similarities (such as response time requirements and data sharing restrictions) between integration applications and multi-tenant environments, we expect cache-aware metascheduling to also increase the efficiency of multi-tenant cloud environments. In future work, we will apply cache-aware metascheduling to multi-tenant clouds to determine what degree of performance enhancements can be achieved.

The source code simulating the integrated avionics system discussed in Section 5 can be downloaded at `ascent-design-studio.googlecode.com`.

References

1. Abandah, G., Abdelkarim, A.: A Study on Cache Replacement Policies (2009)
2. Allen, J., Kennedy, K.: Automatic loop interchange. In: Proceedings of the 1984 SIGPLAN Symposium on Compiler Construction, p. 246. ACM (1984)
3. Asaduzzaman, A., Mahgoub, I.: Cache Optimization for Embedded Systems Running H. 264/AVC Video Decoder. In: IEEE International Conference on Computer Systems and Applications, 2006, pp. 665–672. IEEE (2006)
4. Atlas, A., Bestavros, A.: Statistical rate monotonic scheduling. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, 1998, pp. 123–132. IEEE (1998)
5. Bahar, R., Albera, G., Manne, S.: Power and performance tradeoffs using various caching strategies. In: Proceedings of the International Symposium on Low Power Electronics and Design, 1998, pp. 64–69. IEEE (2005)
6. Beyls, K., DâĂŹHollander, E.: Reuse distance as a metric for cache behavior. In: Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems, vol. 14, pp. 350–360. Citeseer (2001)
7. Chen, T., Baer, J.: Reducing memory latency via non-blocking and prefetching caches. ACM SIGPLAN Notices 27(9), 51–61 (1992)
8. Dhall, S., Liu, C.: On a real-time scheduling problem. Operations Research 26(1), 127–140 (1978)
9. Fu, J., Patel, J., Janssens, B.: Stride directed prefetching in scalar processors. In: Proceedings of the 25th Annual International Symposium on Microarchitecture, pp. 102–110. IEEE Computer Society Press (1992)
10. Ghosh, S., Melhem, R., Mossé, D., Sarma, J.: Fault-tolerant rate-monotonic scheduling. Real-Time Systems 15(2), 149–181 (1998)
11. Guo, F., Solihin, Y.: An analytical model for cache replacement policy performance. In: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, pp. 228–239. ACM (2006)
12. Kennedy, K., McKinley, K.: Maximizing Loop Parallelism and Improving Data Locality Via Loop Fusion and Distribution. In: Banerjee, U., Gelernter, D., Nicolau, A., Padua, D.A. (eds.) LCPC 1993. LNCS, vol. 768, pp. 301–320. Springer, Heidelberg (1994)
13. Kowarschik, M., Weiß, C.: An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In: Meyer, U., Sanders, P., Sibeyn, J.F. (eds.) Algorithms for Memory Hierarchies. LNCS, vol. 2625, pp. 213–232. Springer, Heidelberg (2003)
14. Lee, Y., Kim, D., Younis, M., Zhou, J., McElroy, J.: Resource scheduling in dependable integrated modular avionics. In: Proceedings International Conference on Dependable Systems and Networks, DSN 2000, pp. 14–23. IEEE (2000)
15. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of Real Time Systems Symposium, 1989, pp. 166–171. IEEE (1987)
16. Manjikian, N., Abdelrahman, T.: Array data layout for the reduction of cache conflicts. In: Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems, pp. 1–8. Citeseer (1995)
17. Nayfeh, B., Olukotun, K.: Exploring the design space for a shared-cache multiprocessor. In: Proceedings of the 21st Annual International Symposium on Computer Architecture, p. 175. IEEE Computer Society Press (1994)
18. Orozco, J., Cayssials, R., Santos, J., Ferro, E.: 802.4 rate monotonic scheduling in hard real-time environments: Setting the medium access control parameters. Information Processing Letters 62(1), 47–55 (1997)

19. Panda, P., Nakamura, H., Dutt, N., Nicolau, A.: Augmenting loop tiling with data alignment for improved cache performance. IEEE Transactions on Computers 48(2), 142–149 (2002)
20. Pingali, S., Kurose, J., Towsley, D.: On Comparing the Number of Preemptions under Earliest Deadline and Rate Monotonic Scheduling Algorithms (2007)
21. Reineke, J., Grund, D., Berg, C., Wilhelm, R.: Timing predictability of cache replacement policies. Real-Time Systems 37(2), 99–122 (2007)
22. Robinson, J., Devarakonda, M.: Data cache management using frequency-based replacement. ACM SIGMETRICS Performance Evaluation Review 18(1), 134–142 (1990)
23. Rodríguez-Dapena, P.: Software safety certification: a multidomain problem. IEEE Software 16(4), 31–38 (1999)
24. Shiue, W., Chakrabarti, C.: Memory design and exploration for low power, embedded systems. The Journal of VLSI Signal Processing 29(3), 167–178 (2001)
25. Singhai, S., McKinley, K.: A parametrized loop fusion algorithm for improving parallelism and cache locality. The Computer Journal 40(6), 340 (1997)
26. Smith, J., Goodman, J.: Instruction cache replacement policies and organizations. IEEE Transactions on Computers, 234–241 (1985)
27. Sprangle, E., Carmean, D.: Increasing processor performance by implementing deeper pipelines. In: Proceedings of the 29th Annual International Symposium on Computer Architecture, 2002, pp. 25–34. IEEE (2002)
28. Stewart, D., Barr, M.: Rate monotonic scheduling. In: Embedded Systems Programming, pp. 79–80 (2002)
29. Wang, Z., Guo, C., Gao, B., Sun, W., Zhang, Z., An, W.: A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In: IEEE International Conference on e-Business Engineering, pp. 94–101. IEEE (2008)
30. Wolf, M., Maydan, D., Chen, D.: Combining loop transformations considering caches and scheduling. In: Micro, p. 274. IEEE Computer Society (1996)
31. Yi, Q., Kennedy, K.: Improving memory hierarchy performance through combined loop interchange and multi-level fusion. International Journal of High Performance Computing Applications 18(2), 237 (2004)

Dynamic Migration of Processing Elements for Optimized Query Execution in Event-Based Systems

Waldemar Hummer, Philipp Leitner, Benjamin Satzger, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology, Austria
{lastname}@infosys.tuwien.ac.at

Abstract. This paper proposes a method for optimized placement of query processing elements in a distributed stream processing platform consisting of several computing nodes. We focus on the case that multiple users run different continuous Complex Event Processing (CEP) queries over various event streams. In times of increasing event frequency it may be required to migrate parts of the query processing elements to a new node. Our approach achieves a tradeoff between three dimensions: balancing the load among nodes, avoiding duplicate buffering of events, and minimizing the data transfer between nodes. Thereby, we also take one-time costs for migration of event buffers into account. We provide a detailed problem description, present a solution based on metaheuristic optimization, and evaluate different aspects of the problem in a Cloud Computing environment.

Keywords: event-based systems, continuous queries, migrating query processing elements, placement of event subscriptions, WS-Aggregation.

1 Introduction

In recent years, academia and industry have increasingly focused on event-based systems (EBS) and Complex Event Processing (CEP) [11] for Internet-scale data processing and publish-subscribe content delivery. The massive and continuous information flow of today requires techniques to efficiently handle large amounts of data, e.g., in areas such as financial computing, online analytical processing (OLAP), wireless and pervasive computing, or sensor networks [22]. In most of these application areas, filtering and combining related information from different event sources is crucial for potentially generating added value on top of the underlying (raw) data. Platforms that are specialized in continuously querying data from event streams face difficult challenges, particularly with respect to performance and robustness. Evidently, continuous queries that consider a sliding window of past events (e.g., moving average of historical stock prices in a financial computing application) require some sort of buffering to keep the relevant events in memory. State-of-the-art query engines are able to optimize this buffer size and to drop events from the buffer which are no more needed (e.g., [17]).

However, a topic that is less covered in literature is how to optimize resource usage in a system with multiple continuous queries executing concurrently.

In our previous work we have presented *WS-Aggregation* [13,14], a distributed platform for aggregation of event-based Web services and Web data. The platform allows multiple users to perform continuous queries over event emitting data sources. WS-Aggregation employs a collaborative computing model in which incoming user requests are split into parts, which are then assigned to one or more aggregator nodes. For instance, when a query involves input data from two or more data sources, each of the inputs may be handled by a different aggregator. Throughout various experiments we observed that query distribution and placement of processing elements has a considerable effect on the performance of the framework. For the remainder of the paper, an *event subscription* determines the node that receives and processes the events of an event stream.

In this paper we study how the placement of processing elements affects the performance of single queries and the overall system. To that end, we take several aspects into account. Firstly, computing nodes have resource limits, and in times of peak loads we need to be able to adapt and reorganize the system. Secondly, complex queries over event streams require buffering of a certain amount of past events, and the required memory should be kept at a minimum. Finally, if the tasks assigned to collaborating nodes contain inter-dependencies, possibly a lot of network communication overhead takes place between the aggregator nodes. We propose an approach which considers all of the above mentioned points and seeks to optimize the system configuration. This work is highly important for the runtime performance of event-based systems that deal with load balancing and dynamic migration of event subscriptions, as exemplified using WS-Aggregation.

In the remainder of this paper, we first discuss related work in Section 2. In Section 3, we present the model for event-based continuous queries in WS-Aggregation. Section 4 discusses different strategies for optimal placement of processing elements in distributed event processing platforms and formulates the tradeoff between conflicting goals as a combinatorial optimization problem. Some implementation details are discussed in Section 6, and the overall approach is evaluated in Section 7. Section 8 concludes the paper with a future work outlook.

2 Related Work

Due to the large number of its application areas, event processing has attracted the interest of both industry and research [19,27]. Important topics in CEP include pattern matching over event streams [1], aggregation of events [20] or event specification [10]. In this paper, we focus on optimizing the distributed execution of continuous queries over event streams. Hence, we concentrate on some related work in this area in the remainder of this section.

Optimized placement of query processing elements and operators has previously been studied in the area of distributed stream processing systems. Pietzuch et al. [23] present an approach for network-aware operator placement on geographically dispersed machines. Bonfils and Bonnet [7] discuss exploration and adaptation techniques for optimized placement of operator nodes in sensor networks. Our work is also related to query plan creation and multi query optimization, which are core fields in database research. In traditional centralized databases, permutations of join-orders in the query tree are considered in order to compute an optimal execution plan for a single query [15]. Roy et al. [24] present an extension to the *AND-OR DAG* (Directed Acyclic Graph) representation, which models alternative execution plans for multi-query scenarios. Based on the AND-OR DAG, a thorough analysis of different algorithms for multi-query optimizing has been carried out. Zhu et al. [33] study exchangeable query plans and investigate ways to migrate between (sub-)plans.

Seshadri et al. [25,26] have identified the problem that evaluating continuous queries at a single central node is often infeasible. Our approach builds on their solution which involves a cost-benefit utility model that expresses the total costs as a combination of communication and processing costs. Although the approaches target a similar goal, we see some key differences between their and our work. Firstly, their approach builds on hierarchical network partitions/clusters, whereas WS-Aggregation is loosely coupled and collaborations are initiated in an ad-hoc fashion. Secondly, their work does not tackle runtime migration of query plans and deployments, which is a core focus in this paper. In fact, WS-Aggregation implements the Monitor-Analyze-Plan-Execute (MAPE) loop known from Autonomic Computing [16]. In that sense, the purpose of our optimization algorithm is not to determine an optimal query deployment up front, but to apply reconfigurations as the system involves. Chen et al. [9] describe a way to offer continuous stream analytics as a cloud service using multiple engines for providing scalability. Each engine is responsible for parts of the input stream. The partitioning is based on the contents of the data, e.g., each engine could be responsible for data generated in a certain geographical region.

Several previous publications have discussed issues and solutions related to active queries for internet-scale content delivery. For instance, Li et al. [18] presented the OpenCQ framework for continuous querying of data sources. In OpenCQ a continuous query is a query enriched with a trigger condition and a stop condition. Similarly, the NiagaraCQ system [8] implements internet-scale continuous event processing. Wu et al. [32] present another approach to dealing with high loads in event streams, tailored to the domain of real-time processing of RFID data. Numerous contributions in the field of query processing over data streams have been produced as part of the Stanford Stream Data Manager (STREAM) project [21]. The most important ones range from a specialized query language, to resource allocation in limited environments, to scheduling algorithms for reducing inter-operator queuing. Their work largely focuses on how to approximate query answers when high system load prohibits exact query execution. Query approximation and load shedding under insufficient available resources is also discussed in [3]. Our approach does not support approximation, but exploits the advantages of Cloud Computing to allocate new resources for dynamic migration of query processing elements.

Furthermore, database research has uncovered that special types of queries deserve special treatment and can be further optimized, such as k-nearest neighbor queries [6] or queries over streams that adhere to certain patterns or constraints [4]. WS-Aggregation also considers a special form of 3-way distributed query optimization, which we have presented in earlier work [13].

3 Event-Based Continuous Queries in WS-Aggregation

In the following we establish the model for distributed processing of event-based continuous queries that is applied in WS-Aggregation. The model serves as the basis for the concepts discussed in the remainder of the paper. WS-Aggregation is a distributed platform for large-scale aggregation of heterogeneous internet-based data sources, which supports push-style updates using the notion of continuous event queries on data sources. More information can be found in [13].

Table 1. Description of Symbols and Variables in Event-Based Query Model

Symbol	**Description**
$A = \{a_1, a_2, \ldots, a_n\}$	Set of deployed aggregator nodes.
$Q = \{q_1, q_2, \ldots, q_m\}$	Queries that are handled by the platform at some point in time.
$I = \{i_1, i_2, \ldots, i_k\}$	Set of all inputs over all queries.
$\mathit{inputs} : Q \rightarrow \mathcal{P}(I)$	Function that returns all inputs of a query.
$\mathit{deps} : Q \rightarrow \mathcal{P}(I \times I)$	Function that returns all data dependencies of a query.
$S = \{s_1, s_2, \ldots, s_l\}$	Data sources that emit events over which queries are executed.
$\mathit{source} : I \rightarrow S$	Function to determine the data source targeted by an input.
$\mathit{query} : I \rightarrow Q$	Function to determine the query an input belongs to.
$\mathit{buf} : A \rightarrow \mathcal{P}(S)$	Function to determine which data sources an aggregator buffers.

Table 1 summarizes the symbols and variables that are used in the formalization. In our model, a number of aggregator nodes (A) are collectively responsible to execute multiple continuous user queries (Q). Each query processes one or more inputs (I) from external data sources (S). The function *inputs* maps queries to inputs ($\mathcal{P}(I)$ denotes the power set of I), and the function *source* returns the data source targeted by an input. The actual processing logic of the query is application specific and not directly relevant for our purpose. However, we consider that a query q may contain data dependencies among any two of its inputs $i_x, i_y \in \mathit{inputs}(q), i_x \neq i_y$. A dependency $(i_x, i_y) \in \mathit{deps}(q)$ means that i_y can only be processed after certain data from i_x have been received, because the data are required either 1) by the request to initiate the event stream from the data source underlying i_y, or 2) by a *preprocessing* query that prepares (e.g., groups, filters, aggregates) the incoming events for i_y. Such dependencies are often seen in continuous queries over multiple data streams [5], where subscriptions are dynamically created (or destroyed) when a specific pattern or result is produced by the currently active streams. An example could be a sensor emitting temperature data in a smart home environment, which only gets activated as soon as another sensor emits an event that a person has entered the room.

Although we use the terms service and data source interchangeably, strictly speaking the notion of data source is narrower, because every entry in S is identified by a pair $(epr, filter)$, where *epr* is the *Endpoint Reference* [28] (location) of the service and the $filter$ expression determines which types of events should be returned. That is, different data sources may be accessed under the same service endpoint. The $filter$ may be empty, in which case events of all types are returned.

The reason for abstracting inputs from data sources is that different queries may require different data from one and the same source. As an example, assume a data source which every second emits an event with the market price of two stocks, and two queries which compute the *Pearson* correlation as well as the *Spearman* correlation of the historical prices. This means that each of the inputs needs to be processed (computed) separately, but the same underlying event buffer can be used for both inputs. We use the function *buf* to determine the data sources from which an aggregator "currently" (at some point in time) receives and buffers events.

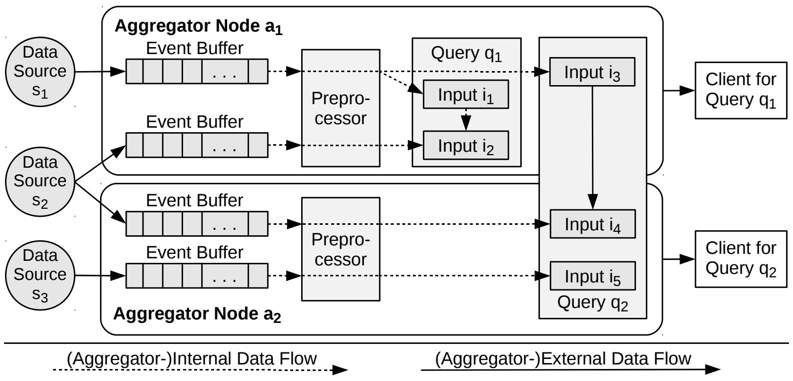


Fig. 1. Illustrative Instantiation of the Model for Distributed Event-Based Queries

The key aspects of the processing model are illustrated in Figure 1, which depicts two aggregator nodes (a_1,a_2) executing two queries (q_1,q_2) consisting of five inputs (i_1,...,i_5) in total. The query execution happens in two steps: firstly, the incoming events are buffered and preprocessed to become the actual inputs (e.g., average value of previous stock prices), and secondly the inputs are joined and combined as defined in the query specification. Dashed lines in the figure indicate aggregator-internal data flow, whereas solid lines stand for data exchanged with external machines. Aggregator a_1 orchestrates the execution of query q_1 and notifies the client of new results. We hence denote a_1 as the *master* aggregator for q_1 (analogously, a_2 is the master of q_2). The data source s_3 provides data for one single input (i_5), whereas inputs i_1/i_3 and i_2/i_4 are based on the events from s_1 and s_2, respectively. We observe that the events from s_2 are buffered both on a_1 and on a_2, which we denote buffer *duplication*. In Figure 1, an arrow pointing from an input i_x to i_y indicates a data dependency, i.e., that i_x provides some data which are required by i_y. In the case of i_1 and i_2, this passing of data happens locally, whereas i_3 and i_4 are handled by different aggregators and hence data are transmitted over the network. We see that assigning i_3 to node a_1 has the advantage that the events from s_1 are buffered only once (for both i_1 and i_3), but is disadvantageous with respect to network traffic between the two aggregators a_1 and a_2. Conversely, s_2 is buffered on both aggregators, reducing the network traffic but requiring more memory. Section 4 deals with this tradeoff in more detail and further refines the optimization problem that we strive to solve.

4 Problem Formulation

In this section we provide a detailed definition for the problem of finding an optimal placement of processing elements in distributed event processing platforms. The basis for optimization is the current assignment of inputs to aggregators at some point in time, $cur : I \rightarrow \mathcal{P}(A)$, where $\mathcal{P}(A)$ denotes the powerset of A. We define that $cur(i) = \emptyset$ *iff* input i has not (yet) been assigned to any aggregator node. For now, we assume that each input (as soon as it has been assigned) is only handled by one aggregator, hence $|cur(i)| \leq 1, \forall i \in I$, but in Section 4.3 we will discuss the case that inputs are redundantly assigned to multiple aggregators for fail-safety. The desired result is a new assignment $new : I \rightarrow \mathcal{P}(A)$ in which all inputs are assigned to some aggregator, $|new(i)| = 1, \forall i \in I$. The difference between cur and new constitutes all inputs that need to be migrated from one aggregator to another, denoted as $M := \{i \in I \mid cur(i) \neq \emptyset \wedge cur(i) \neq new(i)\}$.

Migrating a query input may require to migrate/duplicate the event buffer of the underlying data source, if such a buffer does not yet exist on the target aggregator. The technical procedure of migrating event buffers and subscriptions is detailed in Section 6.1. The (computational) cost associated with this operation is proportional to the size of the buffer in bytes, expressed as $size : S \times (A \cup \{\emptyset\}) \rightarrow \mathbb{N}$. For instance, the buffer size for a source s on an aggregator a is referenced as $size(s, a)$. If the aggregator is undefined ($\emptyset$), then the buffer size function returns zero: $size(s, \emptyset) = 0, \forall s \in S$. The costs for migration of an input i from its current aggregator to a new node (function $migr$) only apply when the data source of i is not yet buffered on the new node, as expressed in Equation 1.

$$migr(i) := \begin{cases} size(source(i), cur(i)), & \text{if } source(i) \notin buf(new(i)) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

In order to decide on actions for load balancing, we need to introduce some notion to express the current load of an aggregator node. In earlier work [14] we observed that the main influencing factor for the aggregators' workload in

WS-Aggregation is the number of inputs and the data transfer rate of the underlying event streams. The transfer rate of data streams is therefore continuously measured and averaged over a given time interval (e.g., 1 minute). Every aggregator provides a metadata interface which can be used to retrieve this monitoring information as a function $rate : (S \cup I) \rightarrow \mathbb{R}$, measured in kilobytes per second (kB/s). The $rate(s)$ of a data stream $s \in S$ is the transfer rate of external events arriving at the platform, and $rate(i)$ for an input $i \in I$ is the internal rate of events after the stream has passed the preprocessor. Based on the data transfer rate, we define the *load* function for an aggregator $a \in A$ as $load(a) := \sum_{s \in buf(a)} \sum_{i \in I_s} rate(s) \cdot c(i)$, where I_s denotes the set of all inputs targeting s, i.e., $I_s := \{i \in I \mid source(i) = s\}$, and $c : I \rightarrow \mathbb{R}$ is an indicator for the computational overhead of the preprocessing operation that transforms the data source s into the input i. The computational overhead depends on the processing logic and can be determined by monitoring. If no information on the running time of a processing step is available then $c(i)$ defaults to 1. For simplification, the assumption here is that n data streams with a *rate* of m kB/s generate the same load as a single data stream with a *rate* of $n * m$ kB/s. We denote the minimum load of all aggregators as $minload := min(\bigcup_{a \in A} load(a))$, and the difference between *minload* and the load of an aggregator a as $ldiff(a) := load(a) - minload$.

To obtain a notion of the data flow, in particular the network traffic caused by external data flow between aggregator nodes (see Figure 1), Equation 2 defines the $flow$ between two inputs $i_1, i_2 \in I$. If the inputs are not dependent from each other or if both inputs are handled by the same aggregator, the $flow$ is 0. Otherwise, $flow$ amounts to the data transfer rate ($rate(i_1)$), measured in kB/s.

$$flow(i_1, i_2) := \begin{cases} rate(i_1), & \text{if } (i_1, i_2) \in deps(query(i_1)) \wedge new(i_1) \neq new(i_2) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Finally, Equation 3 introduces *dupl* to express buffer duplication. The idea is that each data source $s \in S$ needs to be buffered by at least one aggregator, but additional aggregators may also buffer events from the same source (see Figure 1). The function $dupl(s)$ hence subtracts 1 from the total number of aggregators buffering events from s.

$$dupl(s) := |\{a \in A \mid s \in buf(a)\}| - 1 \tag{3}$$

4.1 Optimization Target

We now combine the information given so far in a single target function to obtain a measure for the costs of the current system configuration and the potential benefits of moving to a new configuration. Overall, we strive to achieve a tradeoff between three dimensions: balancing the load among aggregator nodes (L), avoiding duplicate buffering of events (D), while at the same time minimizing the data transfer between nodes (T). The goal of L is to keep each node responsive and to account for fluctuations in the frequency and size of incoming event data. The D dimension attempts to minimize the globally consumed memory, and T aims at a reduction of the time and resources used for marshalling/transmitting/unmarshalling of data.

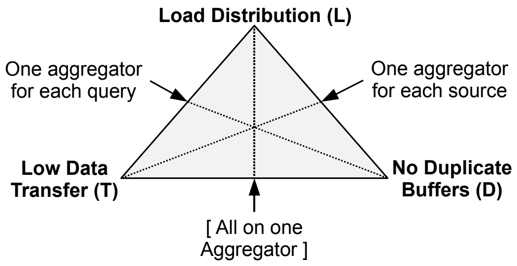

Load Distribution (L)
One aggregator for each query
One aggregator for each source
Low Data Transfer (T)
No Duplicate Buffers (D)
[All on one Aggregator]

Fig. 2. Relationship between Optimization Targets

Figure 2 illustrates the tradeoff relationship as a "Magic Triangle": each pair of goals can be fulfilled separately, but the three goals cannot fully be satisfied in combination. For instance, a way to achieve a balanced load for all aggregators (L) in combination with no duplicate data source buffers (D) is to assign each source to a single aggregator. However, if a single query contains several interdependent data sources on different aggregators (which is likely to occur in this case), the aggregators possibly need to frequently transmit data. Conversely, to achieve load distribution (L) together with low data transfer (T), each query with all its inputs could be assigned to a single aggregator, but we observe that duplicate buffers come into existence if any two queries on different aggregators use the same underlying data source. Finally, to achieve both T and D at the same time, all requests could be assigned to one single aggregator. As indicated by the brackets in Figure 2, this possibility is generally excluded since we are striving for a distributed and scalable solution.

The function F' in Equation 4 contains the three components that are to be minimized. We observe that the three parts have different value ranges. Therefore, the target function includes user-defined weights (w_L,w_T,w_D) to offset the differences of the value ranges and to specify which of the parts should have more impact on the target function.

$$F' := w_L * \sum_{a \in A} ldiff(a) + w_T * \sum_{i_1, i_2 \in I} flow(i_1, i_2) + w_D * \sum_{s \in S} dupl(s) \rightarrow min! \quad (4)$$

We observe that the optimization target in Equation 4 only considers how good a new system configuration (i.e., assignment of inputs to aggregators) is, but not how (computationally) expensive it is to reach the new setup. To account for the costs of migrating query inputs, we make use of the *migr* function defined earlier in Section 4 and use a weight parameter w_M to determine its influence. The final target function F is printed in Equation 5. Note that the additional one-time costs for migration in F are conceptually different from the cost components in F' which apply continuously during the lifetime of the queries.

$$F := F' + w_M * \sum_{i \in M} migr(i) \rightarrow min! \qquad (5)$$

4.2 Elastic Scaling Using Cloud Computing

The premise for being able to change the current system configuration (moving from *cur* to *new*) as defined in the optimization target in Section 4.1 is that there are enough resources globally available to execute the migration tasks. To ensure that the resources are sufficient, we take advantage of *Cloud Computing* [2] techniques to elastically scale the platform up and down. To that end, each aggregator exposes metadata about the current stress level of the machine it is running on, and new machines are requested if all nodes are fully loaded. Conversely, if the nodes operate below a certain stress threshold, the queries can be rearranged to release machines back to the Cloud. For more details about elastic scaling in WS-Aggregation we refer to [13].

The notion of stress level is quite broad - it may include CPU and memory usage, list of open files and sockets, length of request queues, number of threads and other metrics. For simplification, we assume that the individual parts of the stress level function are added up and normalized, resulting in a function $stress : A \rightarrow [0, 1]$. Every aggregator provides a metadata interface which can be used to retrieve monitoring information and performance characteristics of the underlying physical machine. We use the upper bound of the stress level (value 1) to express that an aggregator is currently working at its limit and cannot be assigned additional tasks.

During the optimization procedure, the nodes' stress levels are continuously monitored. To determine whether a reconfiguration can be applied, it must be ensured that $\forall_{a \in A} : (stress(a) > \lambda) \implies \forall_{i \in I}(cur(i) = a \vee new(i) \neq a)$, for a configurable stress level λ (e.g., $\lambda = 0.9$). This criterion allows inputs to be removed from machines with high stress level, but prohibits the assignment of new query inputs. If the algorithm fails to find a valid solution under the given constraints, a new machine is requested from the Cloud environment and the optimization is restarted.

4.3 Extension: Robustness by Redundancy

So far, this paper has considered the case that each query input is handled by a single node. While this may be sufficient for most applications, in a safety-critical system it may be required to process inputs redundantly in order to mitigate the impact of machine outages. Our query optimization model therefore maps inputs to sets of aggregators ($cur : I \rightarrow \mathcal{P}(A)$), as defined in Section 4. As part of the specification of a query q, users define the required level of redundancy, $red(q) \in \{1, 2, \dots, |A|\}$. The platform then duplicates the instantiation of the query, ensuring that each input is assigned to multiple aggregators, $\forall i \in inputs(q) : |cur(i)| \geq red(q)$. If, for any input $i \in I$, one of the aggregators $a \in cur(i)$ goes down, the event buffer migration technique allows to select a replacement aggregator for a and to copy the state from one of the replicated "backup" nodes $cur(i) \backslash \{a\}$.

5 Optimization Algorithm

The problem of finding an optimal input-to-aggregator assignment introduced in Section 4 is a hard computational problem, and the search space under the given constraints is prohibitively large (for a large number of inputs and many aggregators) and prohibits to compute exact solutions in feasible time. A formal proof of the problem's intractability is out of the scope of this paper, but we observe the combinatorial explosion as the algorithm needs to evaluate $\mathcal{O}(|A|^{|I| * red_{max}})$ solutions ($red_{max} := max(\bigcup_{q \in Q} red(q))$ denotes the maximum level of redundancy), each of which may potentially be optimal with respect to the target function F. In particular, pruning the solution space is hard to apply because during the search no solution can be easily identified as being suboptimal no matter what other solutions are derived from it. We therefore apply a metaheuristic and use Variable Neighborhood Search (VNS) [12] to approximate a near-optimal solution. The basic principle of VNS is to keep track of the best recorded solution x and to iterate over a predefined set of neighborhood structures which generate new solutions that are similar to x (for more details, see [12]). VNS has been successfully applied in a vast field of problem domains; one example is the multiplayer scheduling problem with communication delays [12], which has strong similarities to our problem. Figure 3 illustrates the encoding of a solution with 3 queries, 10 inputs and a maximum redundancy level of $red_{max} = 2$.

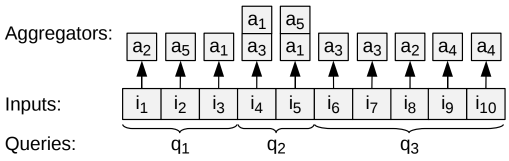

Aggregators:
a2 a5 a1 a3 a1 a1 a5 a3 a3 a2 a4 a4
Inputs:
i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
Queries:
q1 q2 q3

Fig. 3. Example of Solution Encoding in Optimization Algorithm with $red_{max} = 2$

5.1 Search Neighborhoods

The definition of neighborhoods in the VNS algorithm allows to guide the search through the space of possible solutions. In the following list of neighborhoods (NH), $temp : I \rightarrow A$ denotes the input-to-aggregator assignment of a temporary solution which is evaluated by the algorithm, and $temp' : I \rightarrow A$ denotes a solution which has been derived from $temp$ as specified by the NH.

- **avoid duplicate buffers NH:** This NH takes a random data source $s \in S$, determines all its inputs $I_s := \{i \in I \mid source(i) = s\}$ and the aggregators responsible for them, $A_s := \bigcup_{i \in I_s} temp(i)$. The NH then generates $|A_s|$ new solutions, in which all inputs in I_s are assigned to one of the responsible aggregators: $|\bigcup_{i \in I_s} temp'(i)| = 1 \wedge \forall i \in I_s : temp'(i) \in A_s$. When this neighborhood gets applied, the newly generated solutions will by tendency have less duplicate buffers.
- **bundle dependent inputs NH:** This NH selects a random query $q \in Q$ and generates new solutions in which all interdependent inputs of q are placed on the same aggregator node. More specifically, for each newly generated solution $temp'$ the following holds: $\forall (i_1, i_2) \in deps(q) : temp'(i_1) = temp'(i_2)$. Note that also transitive dependencies are affected by this criterion. The effect of this neighborhood is a reduced data traffic between aggregators.
- **equal data load per aggregator NH:** This NH selects the two aggregators $a_{max}, a_{min} \in A$ with $load(a_{max}) = max(\bigcup_{a \in A} load(a))$ and $load(a_{min}) = min(\bigcup_{a \in A} load(a))$, and generates a new solution by moving the input with the smallest data rate from a_{max} to a_{min}. More formally, let $I_{max} := \{i \in I \mid temp(i) = a_{max}\}$ denote the set of inputs that are assigned to aggregator a_{max} in the $temp$ solution, then the following holds in every solution derived from it: $temp'(\arg\min_{i \in I_{max}} rate(i)) = a_{min}$.
- **random aggregator swap NH:** This NH simply selects a random subset of inputs $I_x \subseteq I$ and assigns a new aggregator to each of these inputs, $\forall i \in I_x : temp'(i) \neq temp(i)$. The purpose of this NH is to perform jumps in the search space to escape from local optima.

VNS continuously considers neighborhood solutions to improve the current best solution until a termination criterion is reached. The criterion is either based on running time or solution quality. Furthermore, the algorithm only considers valid solutions with respect to the hard constraints (e.g., minimum level of redundancy as defined in Section 4.3). If a better solution than the current setting is found, the required reconfiguration steps are executed. The system thereby stays responsive and continues to execute the affected event-based queries.

6 Implementation

In the following we first briefly discuss how continuous queries are expressed and executed in WS-Aggregation, and then focus on implementation aspects concerning the migration of event buffers and subscriptions.

WS-Aggregation is implemented in Java using Web services [31] technology, and largely builds on WS-Eventing [29] as a standardized means to manage subscriptions for event notification messages. The platform is designed for loose coupling – aggregators may dynamically join and leave the system, and collaborative query execution across multiple aggregators is initiated in an ad-hoc manner. The endpoint references of currently available aggregator nodes are deployed in a service registry. WS-Aggregation employs a specialized query language named *WAQL* (Web services Aggregation Query Language), which is built on *XQuery* 3.0 [30] and adds some convenience extensions, e.g., to express data dependencies between query inputs. For the actual XQuery processing, we use the light-weight and high-performance *MXQuery* engine (http://mxquery.org/). More details can be found in [13,14].

6.1 Migration of Event Buffers and Subscriptions

One of the technical challenges in our prototype is the implementation of event buffer migration, which becomes necessary when the result of the optimization (see Section 4) mandates that certain query inputs be moved between aggregators. The challenge is that transmitting the contents of a buffer over the network is a time-consuming operation, and new events for this buffer may arrive while the transmission is still active. At this point, it must be ensured that the arriving events are temporarily buffered and later forwarded to the target aggregator node. Therefore, transactionally migrating an event buffer while keeping the buffer state consistent at both the sending and the receiving node is a non-trivial task.

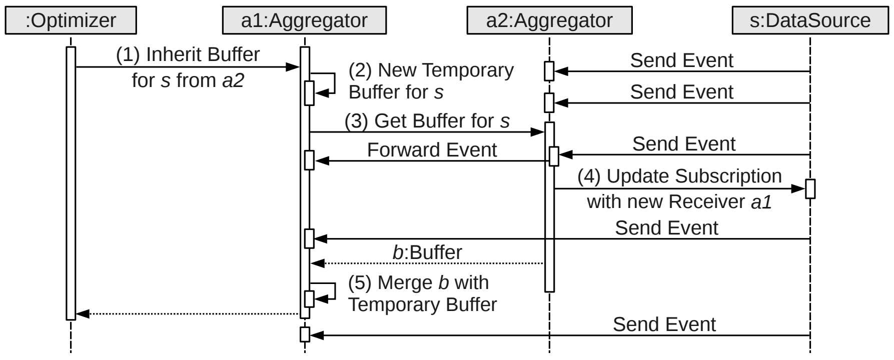

:Optimizer
a1:Aggregator
a2:Aggregator
s:DataSource
(1) Inherit Buffer for s from a2
(2) New Temporary Buffer for s
Send Event
Send Event
(3) Get Buffer for s
Send Event
Forward Event
(4) Update Subscription with new Receiver a1
Send Event
b:Buffer
(5) Merge b with Temporary Buffer
Send Event

Fig. 4. Procedure for Migrating Buffer and Event Subscription between Aggregators

Figure 4 contains a UML sequence diagram which highlights the key aspects of our solution. It involves the optimizer component which instructs an aggregator $a1$ to *inherit* (become the new owner of) the event subscription for data source s together with the previously buffered events from aggregator $a2$ (point (1) in the figure). Data source d continuously sends events to the currently active subscriber. Before requesting transmission of the buffer contents from $a2$ (3), $a1$ creates a temporary buffer (2). Depending on the buffer size, the transmission may consume a considerable amount of time, and the events arriving at $a2$ are now forwarded to $a1$ and stored in the temporary buffer. The next step is to update the event subscription with the new receiver $a1$ (4). Depending on the capabilities of the data source (e.g., a WS-Eventing service), this can either be achieved by a single *renew* operation, or by a combination of an *unsubscribe* and a *subscribe* invocation. However, the prerequisite for keeping the event data consistent is that this operation executes atomically, i.e., at no point in time both $a1$ and $a2$ may receive the events. Finally, after the transmission has finished, the received buffer b is merged with the temporary buffer (5). If the execution fails at some point, e.g., due to connectivity problems, a rollback procedure is initiated and the process can be repeated.

7 Evaluation

To evaluate the performance effects that can be achieved with the proposed approach, we have set up an experimental evaluation in a private Cloud Computing environment with multiple virtual machines (VM), managed by an installation of *Eucalyptus*[1]. Each VM is equipped with 2GB memory and 1 virtual CPU core with 2.33 GHz (comparable to the *small* instance type in Amazon EC2[2]). Our experiments focus on three aspects: firstly, the time required to migrate event buffers and subscriptions between aggregators (Section 7.1); secondly, evolution of the network topology for different optimization parameters (Section 7.2); thirdly, performance characteristics of optimization weights (Section 7.3).

7.1 Migration of Event Buffers and Subscriptions

We first evaluate the effort required to switch from the current configuration to a new (optimized) configuration. For each input $i \in I$ there are three possibilities:

1. If $new(i) = cur(i)$ then there is nothing to do.
2. If $new(i) \neq cur(i) \wedge source(i) \in buf(new(i))$ then the new aggregator $a = new(i)$ needs to be instructed to handle input i, but no migration is required because the target buffer already exists on a.
3. If $new(i) \neq cur(i) \wedge source(i) \notin buf(new(i))$ then we need to perform full migration (or duplication) of the event buffer and subscription.

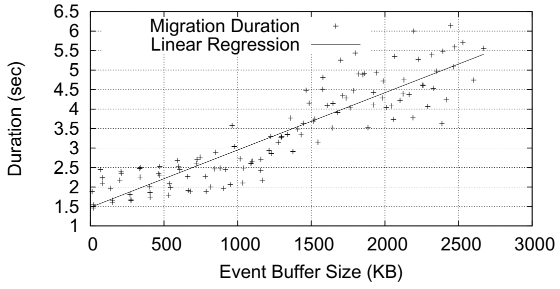


Fig. 5. Duration for Migrating Event Subscriptions for Different Buffer Sizes

Obviously, the most time-consuming and resource-intensive possibility in the list above is point 3. To measure the actually required time, we have executed various buffer migrations with different buffer sizes. Each data point in the scatter

[1] http://www.eucalyptus.com/
[2] http://aws.amazon.com/ec2

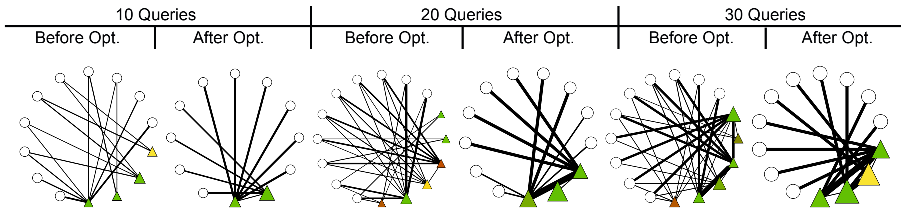
plot in Figure 5 represents a combination of buffer size and migration duration. The duration measures the gross time needed for the old aggregator a_1 to contact the new aggregator a_2, transmitting the buffer, feeding the buffer contents into the query engine on a_2, and freeing the resources on a_1. A linear regression curve is also plotted, which shows an approximate trendline (variance of residuals was 0.2735). Note that the numbers in the figure represent the net buffer size, that is, the actual accumulated size of the events as they were transported over the network (serialized as XML). The gross buffer size, which we evaluate in Section 7.3, is the amount of Java heap space that is consumed by the objects representing the buffer, plus any auxiliary objects (e.g., indexes for fast access).

7.2 Evolution of Network Topology

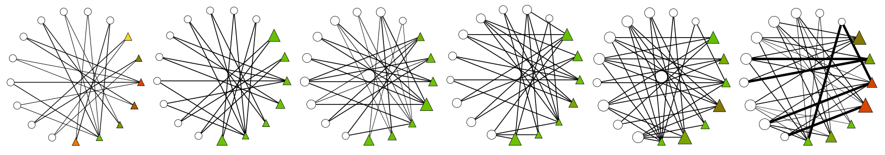
The effect of applying the optimization is that the network topology (i.e., connections between aggregators and data sources) evolves according to the parameter weights. In our second experiment, we deployed 10 data sources (each emitting 1 event per second with an XML payload size of 250 bytes) and 7 aggregator nodes, and started 30 eventing queries in 3 consecutive steps (in each step, 10 queries are added). Each query instantiation has the following processing logic:

- Each query q consists of 3 inputs (i_q^1,i_q^2,i_q^3). The inputs' underlying data sources are selected in *round robin* order. That is, starting with the fourth query, some inputs target the same data source (because in total 10 data sources are available) and the buffer can therefore be shared.
- The events arriving from the data sources are preprocessed in a way that each group of 10 events is aggregated. The contents of these 10 events collectively form the input that becomes available to the query.
- Since we are interested in inter-aggregator traffic, each instance of the test query contains a data dependency between the inputs i_q^1 and i_q^2. This means that, if these two inputs are handled by different nodes, the results from i_q^1 are forwarded over the network to the node responsible for i_q^2.
- Finally, the query simply combines the preprocessed inputs into a single document, and the client continuously receives the new data.

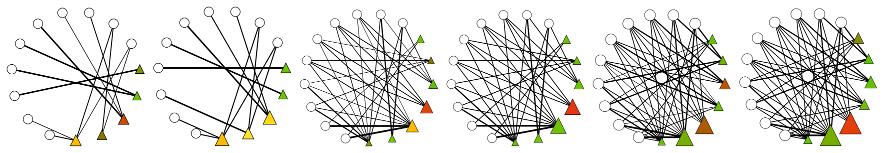
Figure 6 graphically illustrates how the network topology evolves over time for different parameter settings. Each of the subfigures ((a),(b),(c)) contains six snapshots of the system configuration: for each of the 3 steps in the execution (10/20/30 queries), we record a snapshot of the system configuration *before* and *after* the optimization has been applied. In each step, the optimization algorithm runs for 30 seconds, and the best found solution is applied. Data sources are depicted as circles, aggregators are represented by triangles, and the nodes with data flow are connected by a line. The size of the nodes and lines determines the load: the bigger a circle, the more event subscriptions are executed on this data source; the bigger a triangle, the more data this aggregator is buffering; the thicker a line, the more data is transferred over the link. Furthermore, the aggregators' colors determine the stress level (green-yellow-red for low-medium-high).



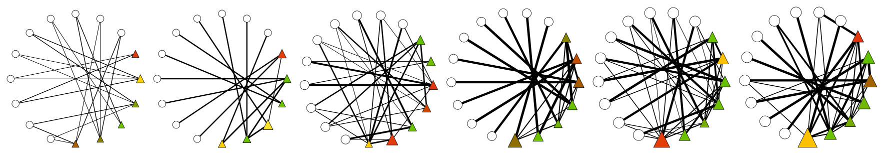
(a) $w_D = 1$, $w_L = 0$, $w_T = 0$



(b) $w_D = 0$, $w_L = 1$, $w_T = 0$



(c) $w_D = 0$, $w_L = 0$, $w_T = 1$



(d) $w_D = 1$, $w_L = 1$, $w_T = 1$; $w_M = 1$

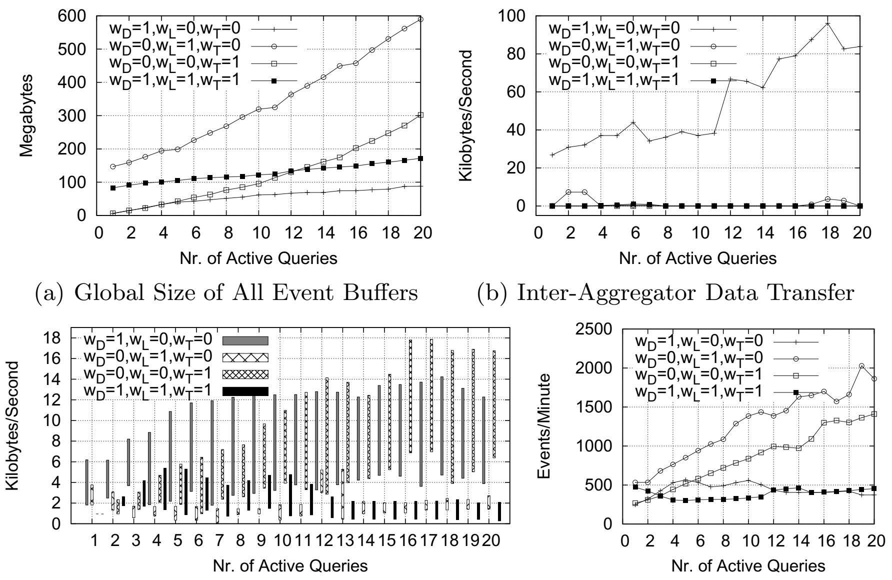
Fig. 6. Effect of Optimization With Different Weights

We can see clearly that different optimization weights result in very distinct topological patterns. A characteristic outcome of emphasizing the w_D parameter (Figure 6(a)) is that few aggregators handle many event subscriptions and are hence loaded with a high data transfer rate. If the goal of preventing duplicate buffering is fully achieved, then there are at most $|S|$ active aggregators (and possibly less, as in Figure 6(a)), however, there is usually some inter-aggregator traffic required. In Figure 6(b) only the weight w_L is activated, which results in a more dense network graph. The inputs are spread over the aggregators, and in many cases multiple aggregators are subscribed with the same event source. Also in the case where w_T is set to 1, the resulting network graph becomes very dense. We observe that in Figure 6(c) there are no inter-aggregator connections, i.e., this setting tends to turn the network topology into a bipartite graph with the data sources in one set and the aggregator nodes in the second set. Finally, in Figure 6(c) all weights, including the penalty weight for migration (w_M) are set to 1. Note that the weights are subject to further customization, because setting equal weights favors parameters that have higher absolute values. In our future work, we plan to evaluate the effect of automatically setting the weights and normalizing the units of the optimization dimensions (D,L,T,M).

7.3 Performance Characteristics of Optimization Parameters

We now use the same experiment setup as in Section 7.2 and evaluate in more detail how the platform's performance characteristics evolve over time when optimization is applied. Again, 10 data sources and 7 aggregator nodes were deployed and queries were successively added to the platform. This time we took a snapshot 30 seconds after each query has been added for execution. The 4 subplots in Figure 7 illustrate the test results as a trendline over the number of active queries (x axis). To slightly flatten the curves, each experiment has been executed in 5 iterations and the numbers in the figures are mean values. The gross heap size of event buffer objects (Figure 7(a)) is determined using the Java instrumentation toolkit (*java.lang.instrument*) by recursively following all object references.

Figure 7(a) shows that the global memory usage is particularly high (up to 600MB for 20 queries) for $w_L = 1$ and also for $w_T = 1$. Figure 7(b) depicts the inter-aggregator transfer, which in our experiments was quite high for $w_D = 1$, and near zero for the other configurations. The box plots in Figure 7(c) show the minimum and maximum event rates over all aggregators. We see that the absolute values and the range difference are high for $w_D = 1$ and $w_T = 1$, but, as expected, considerably lower for the load difference minimizing setting $w_L = 1$. Finally, the combined event frequency of all aggregators is plotted in Figure 7(d).



(a) Global Size of All Event Buffers

(b) Inter-Aggregator Data Transfer

(c) Minimum/Maximum Event Data Rates

(d) Global Event Frequency

Fig. 7. Performance Characteristics in Different Settings

8 Conclusions

The placement of event processing elements plays a key role for the performance of query processing in distributed event-based systems. We have proposed an approach that performs dynamic migration of event buffers and subscriptions to optimize the global resource usage within such platforms. The core idea is that event buffers can be reused if multiple query inputs operate on the same data stream. We identified a non-trivial tradeoff that can be expressed as a "magic triangle" with three optimization dimensions: balanced load distribution among the processing nodes, minimal network traffic, and avoidance of event buffer duplication. Variable Neighborhood Search (VNS) has proven effective for exploring the search space and generating possible solutions.

We have exemplified our solution on the basis of several experiments carried out with the WS-Aggregation framework. The platform integrates well with the Cloud Computing paradigm and allows for elastic scaling based on the current system load and event data frequency. Our evaluation has illustrated how different optimization parameters can be applied to influence the evolution of the network topology over time. Furthermore, we have evaluated how different performance characteristics evolve in different settings. The experience we have gained in the various experiments conducted has shown that the (short-term) costs of migration or duplication are often outweighed by the (long-term) benefits gained in performance and robustness. As part of our ongoing work we are experimenting with very high-frequency event streams that cannot be handled by a single node. We envision an extension of the current query processing model to allow splitting up such streams to multiple aggregators. Furthermore, we are investigating Machine Learning techniques to automatically derive reasonable optimization parameters for the target function based on prior knowledge.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 257483 (Indenica).

References

1. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient Pattern Matching Over Event Streams. In: SIGMOD Int. Conference on Management of Data (2008)
2. Armbrust, M., et al.: Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, University of California at Berkeley (2009)
3. Ayad, A., Naughton, J.: Static optimization of conjunctive queries with sliding win- dows over infinite streams. In: SIGMOD Int. Conf. on Management of Data (2004)
4. Babu, S., Srivastava, U., Widom, J.: Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. ACM Transactions on Database Systems 29, 545–580 (2004)
5. Babu, S., Widom, J.: Continuous queries over data streams. In: ACM SIGMOD International Conference on Management of Data, vol. 30, pp. 109–120 (2001)
6. Böhm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: Int. Conf. on Data Engineering, pp. 156–165 (2007)
7. Bonfils, B.J., Bonnet, P.: Adaptive and decentralized operator placement for in-network query processing. Telecommunication Systems 26, 389–409 (2004)

8. Chen, J., DeWitt, D., Tian, F., Wang, Y.: NiagaraCQ: a scalable continuous query system for Internet databases. In: ACM SIGMOD International Conference on Management of Data, pp. 379–390 (2000)
9. Chen, Q., Hsu, M.: Data stream analytics as cloud service for mobile applications. In: Int. Symp. on Distributed Objects, Middleware, and Applications, DOA (2010)
10. Cugola, G., Margara, A.: TESLA: a Formally Defined Event Specification Language. In: International Conference on Distributed Event-Based Systems (2010)
11. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications (2010)
12. Hansen, P., Mladenović, N.: Handbook of metaheuristics. Springer, Heidelberg (2003)
13. Hummer, W., Leitner, P., Dustdar, S.: WS-Aggregation: Distributed Aggregation of Web Services Data. In: ACM Symposium on Applied Computing (2011)
14. Hummer, W., Satzger, B., Leitner, P., Inzinger, C., Dustdar, S.: Distributed Continuous Queries Over Web Service Event Streams. In: 7th IEEE International Conference on Next Generation Web Services Practices (2011)
15. Ioannidis, Y.E.: Query optimization. ACM Computing Surveys 28, 121–123 (1996)
16. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36(1) (2003)
17. Li, X., Agrawal, G.: Efficient evaluation of XQuery over streaming data. In: International Conference on Very Large Data Bases, pp. 265–276 (2005)
18. Liu, L., Pu, C., Tang, W.: Continual queries for Internet scale event-driven information delivery. IEEE Trans. on Knowledge and Data Engineering 11(4) (1999)
19. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman (2001)
20. Maybury, M.T.: Generating Summaries From Event Data. International Journal on Information Processing and Management 31, 735–751 (1995)
21. Motwani, R., et al.: Query processing, approximation, and resource management in a data stream management system. In: Conference on Innovative Data Systems Research, CIDR (2003)
22. Mühl, G., Fiege, L., Pietzuch, P.: Distributed event-based systems. Springer, Heidelberg (2006)
23. Pietzuch, P., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., Seltzer, M.: Network-aware operator placement for stream-processing systems. In: International Conference on Data Engineering, ICDE (2006)
24. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: ACM SIGMOD International Conference on Management of Data, pp. 249–260 (2000)
25. Seshadri, S., Kumar, V., Cooper, B.: Optimizing multiple queries in distributed data stream systems. In: Int. Conference on Data Engineering, Workshops (2006)
26. Seshadri, S., Kumar, V., Cooper, B., Liu, L.: Optimizing multiple distributed stream queries using hierarchical network partitions. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–10 (2007)
27. Vitria: Complex Event Processing for Operational Intelligence (2010), http://www.club-bpm.com/Documentos/DocProd00015.pdf
28. W3C: Web Services Addressing, http://www.w3.org/Submission/WS-Addressing/
29. W3C: Web Services Eventing, http://www.w3.org/Submission/WS-Eventing/
30. W3C: XQuery 3.0: An XML Query Language, http://www.w3.org/TR/xquery-30/
31. W3C: Web Services Activity (2002), http://www.w3.org/2002/ws/
32. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD International Conference on Management of Data (2006)
33. Zhu, Y., Rundensteiner, E., Heineman, G.: Dynamic plan migration for continuous queries over data streams. In: SIGMOD Int. Conf. on Management of Data (2004)

A Survey on SLA and Performance Measurement in Cloud Computing

Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang

Curtin University, Australia
Mohammed.Alhamad@postgrad.curtin.edu.au,
{Tharam.Dillon,Elizabeth.Chang}@cbs.curtin.edu.au

Abstract. Cloud computing has changed the strategy used for providing distributed services to many business and government agents. Cloud computing delivers scalable and on-demand services to most users in different domains. However, this new technology has also created many challenges for service providers and customers, especially for those users who already own complicated legacy systems. This paper reviews the challenges related to the concepts of trust, SLA management, and cloud computing. We begin with a survey of cloud computing architecture. Then, we discuss existing frameworks of service level agreements in different domains such as web services and grid computing. In the last section, we discuss the advantages and limitations of current performance measurement models for SOA, distributed systems, grid computing, and cloud services. Finally, we summarize and conclude our work.

Keywords: SLA, Measurement, Cloud computing.

1 Introduction

Cloud computing has been the focus of active and extensive research since late 2007. Before the term ‘cloud’ was coined, there was grid technology. Now, the hot topic of research is cloud and more proposed frameworks and models of various solutions for the new technology have started to be applied to the cloud architecture. In this section, we survey the literature in order to determine the most appropriate definition of “cloud computing”. Also, we review the different architectural frameworks and the common challenges that may present major problems for providers and customers who are interested in understanding this type of distributed computing.

2 Definition

Experts and developers who investigate issues and standards related to cloud computing do not necessarily have the same technology background. In research projects, professionals from grid technology, SOA, business, and other domains of technology and management have proposed several concepts to define cloud computing. These definitions of cloud computing still need to be presented in a common standard to cover most technology and aspects of cloud computing. In the context of networking and communication, the term “cloud” is a metaphor for the common internet concept [1]. The cloud symbol is also used to present the meaning of network connection and the way that the cloud technology is provided by internet infrastructure. “Computing” in the context of the cloud domain refers to the technology and applications that are implemented in the cloud data centers [2]. In [3], Vaquero et al. comment on the lack of a common definition of cloud computing.

In this paper, we adopted and considered the definition provided by U.S. NIST (National Institute of Standards and Technology) [4], according to which “Cloud computing is a model for enabling convenient, on demand network access to a share pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management afford or service provider interaction” [4].

Shortcomings of the proposed definitions of cloud computing are as follows

1. None of the definitions consider cloud computing from the technical and business perspectives. This would cause confusion to decision makers in large organizations, especially when they want to define the parameters of a costing model of cloud services.
2. Existing cloud definitions do not specify the onus of responsibility in cases of poor QoS delivery.
3. Most of the proposed definitions consider specific types of cloud services, whereas a comprehensive definition of cloud should clearly define all classes of cloud services.
4. The proposed definitions do not consider a definition of cloud users.

3 Service Level Agreements

A service level agreement is a document that includes a description of the agreed service, service level parameters, guarantees, and actions for all cases of violation. The SLA is very important as a contract between consumer and provider. The main idea of SLAs is to give a clear definition of the formal agreements about service terms like performance, availability and billing. It is important that the SLA include the obligations and the actions that will be taken in the event of any violation, with clearly expressed and shared semantics between each party involved in the online contract.

This section discusses works related to SLAs in three domains of distributed services. Firstly, we discuss the proposed SLAs structure for web services. Secondly, the frameworks of SLAs designed to grid computing are reviewed; thirdly, we discuss the main works that specifically focus on cloud computing. Finally, we include in this section the main shortcomings of these SLA frameworks.

A) SLAs for Web Services

Several specifications for defining SLAs have been proposed for web services. WSLA language [5] introduces a mechanism to help users of web services to configure and control their resources in order to meet the service level. Also, the service users can monitor SLA parameters at run time and report any violation of the service. WSLA was developed to describe services under three categories: 1) Parties: in this section, information about service consumers, service providers, and agents are described. 2) SLA parameters: in this section the main parameters which are measurable parameters are presented in two types of metrics. The first is resource metrics, a type of metrics used to describe a service provider's resources as row information. The second one is composite metrics. This metrics is used to calculate the combination of information about a service provider's resources. The final section of the WSAL specification is Service Level Objective (SLO). This section is used to specify the obligations and all actions when service consumers or service providers do not comply with the guarantees of services. The WSLA provides an adequate level of online monitoring and contracting, but does not clearly specify when and how a level of service can be considered a violation. WSOL [6] is a service level specification designed mainly to specify different objectives of web services. Defining concepts of service management, cost and other objectives of services can be presented in WSOL. However, WSOL cannot adequately meet the objectives of the new paradigm of cloud computing.

WS-Agreement [7] is created by an Open Grid Forum (OGF) in order to create an official contract between service consumers and service providers. This contract should specify the guarantees, the obligations and penalties in the case of violations. Also, the functional requirements and other specifications of services can be included in the SLA. The WS-Agreement has three main sections: name, context, and terms. A unique ID and optional names of services are included in the name section. The information about service consumer and service provider, domain of service, and other specifications of service are presented in the context section. Terms of services and guarantees are described in greater detail in the terms section. These types of online agreements were developed for use with general services. For cloud computing, service consumers need more specific solutions for SLAs in order to reflect the main parameters of the visualization environment; at the same time, these SLA solutions should be dynamically integrated with the business rules of cloud consumers.

The primary shortcomings of these approaches is that they do not provide for dynamic negotiation, and various types of cloud consumers need a different structure for the implementation of SLAs to integrate their own business rules with the guarantees that are presented in the targeted SLA.

B) SLAs for Grid Computing

In the context of grid computing, there are a number of proposed specifications which have been developed especially to improve security and trust for grid services. In [8], an SLA-based knowledge domain has been proposed by Sahai to represent the measurable metrics for business relationships between all parties involved in the transaction of grid services. Also, the author proposed a framework to evaluate the management proprieties of grid services in the lifecycle. In this work, business metrics and a management evaluation framework are combined to produce an estimated cost model for grid services. In our research, we extend this approach in order to build a general costing model based on the technical and business metrics of the cloud domain. The framework proposed in this work lacks a dynamic monitoring technique to help service customers know who takes responsibility when a service level is not provided as specified in SLA documents. Leff [9] conducted a study of the main requirements to define and implement SLAs for the grid community. The author provides an ontology and a detailed definition of grid computing. Then, a scientific discussion is presented about the requirements that can help developers and decision makers to deploy trusted SLAs in a grid community. A basic prototype was implemented in order to validate the use of SLAs as a reliable technique when the grid service provider and customer need to build a trusting relationship. The implementation of the framework in this study does not consider important aspects of security and trust management in grid computing. Keung [10] proposed an SLA-based performance prediction tool to analyse the performance of grid services. Keung uses two sources of information as the main inputs for the proposed model. The source code information and hardware modelling are used to predict the value of performance metrics for grid services. The model proposed by Keung can be used in other types of distributed computing. But in the cloud environment, this model cannot be integrated with a dynamic price model of cloud services. It needs to be improved by using different metrics for cost parameters to reflect the actual price of cloud services. The system proposed by Padget in [11] considers the response time of applications in the grid systems. The main advantage of the proposed system is that it can predict the CPU time for any node in the grid network before conducting the execution. When Padget tested the adaptation SLA model using a real experiment on the grid, the prediction system produced values for response time close to the values obtained when users executed the same application on the grid. Noticing the delay recorded for the large size of executed files, the author claims that the reason for this delay is the external infrastructure such as internet connections. The author also discusses the impact of the time delay caused by external parties to the reputation of service providers when using SLA management systems. Although the author provides a good method for calculating the response time for grid resources, other metrics such as security and management metrics, are absent in this work.

C) SLAs for Cloud Computing

The context of this research is the management of service level agreements in cloud communities. In the sections above, we presented the frameworks and models in the current literature that are designed mainly for managing SLAs in traditional distributed systems. In this section, SLAs and approaches to agreement negotiations in the cloud community are presented.

Valdimir [12] describes the quality of services related to cloud services and different approaches applied to map SLA to the QoS. Services ontology for cloud computing is presented in order to define service capabilities and the cost of service for building a general SLAs framework. The proposed framework does not consider all types of cloud services; it is general and was tested on the Amazone EC2 only. It also needs to consider other types of cloud providers such as PaaS, DaaS, and SaaS. Our framework in this research considers this issue in the validation phase of the research. The framework developed by Hsien [13] focuses on software as a service model of delivery in cloud computing. More details are provided on how the services can be integrated to support the concept of stability of cloud community especially for SaaS.

Shortcomings of the Proposals for SLAs in the Context of Distributed Services

The frameworks and structures that were discussed in previous sections have the following problems:

1. The existing frameworks focus more on the technical attributes than on the security and management aspects of services.
2. The proposed structures of SLAs in the above domains do not include a clear definition of the relationship between levels of violation and the cost of services.
3. Most of the above studies do not integrate a framework of trust management of the service provider with the collected data from monitoring systems of SLAs.
4. The concepts and definitions of service objectives and service descriptions included in SLAs are not easy to understand, especially for business decision makers.
5. The proposed works for cloud environments focus more on the evaluation of virtualization machines on local servers than on existing cloud service providers.
6. Most of the proposed structures of SLAs are defined by technical experts.

4 Performance Measurements Models

Cloud providers have been increased to deliver different models of services. These services are provided at different levels of quality of services. Cloud customers need to have a reliable mechanism to measure the trust level of a given service provider. Trust models can be implemented with various measurement models of services. As a part of this research, we investigate the use of a measurement approach in order to develop a general trust model for cloud community. In this section, the measurement model of SOA, distributed, and grid services will be reviewed.

A) SOA Performance Models

Kounev et al. in [14] propose an analytical approach to modelling performance problems in SOA-based applications. The authors discuss the different realistic J2EE applications for large systems of SOA architecture. A validated approach has been tested for capacity planning of the organizations that use distributed services as an outsourcing infrastructure. The advantage of the proposed method is its ability to predict the number of application servers based on the collected information of SLA metrics. Walter et al. [15] implemented a simulation tool to analyse the performance of composite services. Authors used an online book store as a case study to simulate experiment scenarios. They focus on measuring communication latency and transaction completion time. Real data sets were compared with the simulation results. The authors state that the simulation tool presents results that approximate those of the real data. This type of simulation can be extended and applied to other distributed services. For cloud computing, more efforts is required to make this technique compatible with existing interfaces of cloud providers. Rud et al. in [16] use the WS-BPEL composition approach to evaluate the performance of utilization and throughput of SOA-based systems in large organizations. They developed the proposed methodology using a mathematical model in order to improve the processes of service level agreements in the SOA environment. The main focus of Rud's method is on the management aspects of services. However, this approach does not consider performance issues of response time, data storage, and other metrics of technical infrastructure. For the optimization of total execution time and minimization of business processes cost, Menasce in [17] provides an optimized methodology based on the comparison of performance metrics of SOA-based services. In this study, Menasce developed the proposed method to estimate the cost level of all services which are registered in the SOA directory under medium sized organizations. Then, the cost metric is compared to the real performance of services. The parameters of the performance metrics can be selected by service customers. So, the proposed model can be used for different types of services. Although, the proposed method produces a high level of reliability and usability, issues such as risk management, and trust mechanisms of the relationship between service providers and service customers are not discussed in more details.

B) Distributed Systems Performance Models

Kalepu et al. [18] propose a QoS-based attribute model to define the non-functional metrics of distributed services. Availability, reliability, throughput, and cost attributes are used in their work to define performance of resources of a given service provider. Two approaches of resources are used to calculate the final value of reputation. The first resource is the local rating record. Ratings of services which are invoked by local customers are stored in this record. In the second resource, global ratings of all services that are executed on resources of a given service provider are stored. Although, Kalepu et al. discuss the need to use SLA parameters to calculate the value of performance metrics, they do not explain how these parameters can be linked to the local global resources of a rating system. In [19], Yeom et al. provide a monitoring methodology of the performance parameters of service. The proposed methodology uses the broker monitoring systems to evaluate the performance of resources of a service provider. Collected data of performance metrics are not maintained on the service consumer database. This method incurs low cost in terms of implementing measurement architecture but more risk in terms of privacy, availability of data, and security. Such risks are not easy to control, especially in the case of multi tenant distributed systems. Kim et al. in [20] analyse the quality factors of performance level of services and propose a methodology to assign priorities message processing of distributed web services based on the quality factors of services. This assigning aspect of their framework is a dynamic process in different service domains. They claim that their framework satisfies the agreement regarding service level in web services. The validation methodology of the proposed work lacks a clear definition of the evaluation criteria and a description of the way in which the experiment was conducted to produce the claimed results. The work proposed by Guster et al. in [21] provides an evaluation methodology for distributed parallel processing. In the proposed method, authors use a parallel virtual machine (PVM) and real hosting servers to compare the results of their experiments. The efficiency of the evaluation method performed better in PVM for the processing time. In the real server environment, the experiments presented better performance in terms of communication time. The evaluation of this work does not include the implementation processes and the experiment results are not clearly explained.

C) Cloud Computing Performance Models

Several studies already exist on the scalability of virtual machines. Most of these studies considered the measurement of performance metrics on the local machines. The background loads of tested machines are controlled to compare the results of performance with a different scale of loads. Evangelinos and Hill [22] evaluated the performance of Amazon EC2 to host High Performance Computing (HPC). They use a 32-bit architecture for only two types of Amazon instances. In our study, we run various experiments on most types of Amazon EC2 instances. These instances are: small, large, extra large, high CPU, medium, and high CPU extra large instance. Jureta, and Herssens [23] propose a model called QVDP which has three functions: specifying the quality level, determining the dependency value, and ranking the quality priority. These functions consider the quality of services from the customers' perspective. However, the performance issues related to cloud resources are not discussed and details are missing regarding the correlation of the quality model with the costing model of services. Cherkasova and Gardner [24] use a performance benchmark to analyse the scalability of disk storage and CPU capacity with Xen Virtual Machine Monitors. They measure the performance parameters of visualization infrastructure that are already deployed in most data centres. But they do not measure the scalability of cloud providers using the visualization resources. However, our proposed work profiles the performance of virtualization resources that are already running on the infrastructure of existing cloud providers.

The Shortcomings of the Proposed Works for Above Performance models

1. The above proposed models for evaluating the virtualization services focus on how to measure the performance of virtual machines using local experiments. However, the techniques used for measuring the actual resources of cloud providers need further refinement in order to ensure some level of trust between service providers and the customers.
2. Most of the proposed works on performance evaluation do not allow service customers to specify the parameters of performance metrics. In cloud computing, service customers need a more flexible and dynamic approach to modify the parameters of performance metrics in order to solve the problem of dynamic changes of service requirements and business models of customers.
3. The experiments using the above proposed models do not specify the benchmarks for the performance evaluation.
4. In cloud computing architecture, the relationship between performance monitoring and costing metric is very important. The proposed models do not link the results of performance monitoring with the actual cost metric of services. So, service customers are not able to build a trust relationship with service providers without having a real cost model of services

5 Conclusions

The above discussions have highlighted many issues both in the development of SLAs and Performance Models for Cloud Computing which constitute rich fields for future research.

References

[1] Katzan Jr., H.: On An Ontological View of Cloud Computing. Journal of Service Science (JSS) 3 (2011)
[2] Wyld, D.C.: Moving to the cloud: An intro. to cloud computing in government. IBM Center for the Bus. of Government (2009)
[3] Vaquero, L.M., et al.: A break in the clouds: towards a cloud defin. ACM SIGCOMM Comp. Comm. Rev. 39, 50–55 (2008)
[4] Mell, P., Grance, T.: Draft nist working definition of cloud computing (referenced on June 3, 2009)
[5] Ludwig, H., et al.: Web service level agreement (WSLA) language specification. IBM Corporation (2003)
[6] Tosic, V.: WSOL Version 1.2: Carleton Univ., Dept. of Systems and Comp. Eng. (2004)
[7] Andrieux, A., et al.: Web services agree. spec. (WS-Agreement) (2004)
[8] Sahai, A., et al.: Specifying and monitoring guarantees in commercial grids through SLA (2003)
[9] Leff, A., et al.: Service-level agreements and commercial grids. IEEE Internet Computing 7, 44–50 (2003)
[10] Keung, H.N.L.C., et al.: Self-adaptive and self-optimising resource monitoring for dynamic grid environ., pp. 689–693 (2004)

[11] Padgett, J., et al.: Predictive adaptation for service level agreements on the grid. International Journal of Simulation: Systems, Science and Technology 7, 29–42 (2006)
[12] Stantchev, V., et al.: Neg. and enforcing qos and slas in grid and cloud comp. Adv. in Grid and Perv. Comp., 25–35 (2009)
[13] Wen, C.H., et al.: A SLA-based dynamically integrating services Saas framework, pp. 306–311
[14] Kounev, S., Buchmann, A.: Performance modeling and evaluation of large-scale J2EE applications, pp. 273–284 (2003)
[15] Walter, A., Potter, D.: Compos., Performance Analy. and Simulation of Web Services (2007)
[16] Rud, D., et al.: Performance modeling of ws-bpel-based web service compositions, pp. 140–147 (2006)
[17] Menascé, D.A., et al.: A heuristic approach to optimal service selection in service oriented architectures, pp. 13–24 (2008)
[18] Kalepu, S., et al.: Verity: a QoS metric for selecting Web services and providers, pp. 131–139 (2003)
[19] Yeom, G., Min, D.: Design and implementation of web services qos broker (2005)
[20] Kim, D., et al.: Improving Web services performance using priority allocation method (2005)
[21] Guster, D., et al.: Computing and netw. Perform. of a distr. parallel processing environ. using MPI and PVM commun. methods. J. Computing Sci. in Colleges 18, 246–253 (2003)
[22] Evangelinos, C., et al.: Cloud Comput. for paral. Sci. HPC Applic. Feasib. of run. Coup. Atmos. Ocean Climate Models on Amazon's EC2. Ratio 2, 2.34 (2008)
[23] Jureta, I., et al.: A comprehensive quality model for service-oriented systems. Software Qual. Journal 17, 65–98 (2009)
[24] Cherkasova, L., Gardner, R.: Measur. CPU overhead for I/O processing in the Xen virtual machine monitor, p. 24 (2005)

Experiences with Service-Oriented Middleware for Dynamic Instrumentation of Enterprise DRE Systems

James H. Hill[1] and Douglas C. Schmidt[2]

[1] Indiana University/Purdue University at Indianapolis
Indianapolis, IN, USA
`hillj@cs.iupui.edu`

[2] Carnegie Mellon University
Pittsburgh, PA USA
`dschmidt@sei.cmu.edu`

Abstract. This paper describes our experiences applying a test and evaluation (T&E) service-oriented middleware framework called the *Open-source Architecture for Software Instrumentation Systems (OASIS)* to the Unified SHIP platform, which is a representative system for next-generation shipboard computing systems. The OASIS service-oriented middleware framework discussed in this paper enables instrumenting distributed software systems, such as enterprise distributed real-time and embedded (DRE) systems, to collect and extract metrics without *a priori* knowledge of the metrics collected. The flexibility of OASIS's metametrics-driven approach to instrumentation and data collection increased developer and tester knowledge and analytical capabilities of end-to-end QoS in shipboard computing systems. This paper also discusses our strategy for deploying OASIS in a cloud environment.

1 Introduction

Shipboard computing systems are a class of enterprise distributed real-time and embedded (DRE) systems with stringent quality-of-service (QoS) requirements (such as latency, response time, and scalability) that must be met in addition to functional requirements [24]. To ensure QoS requirements of such DRE systems, developers must analyze and optimize end-to-end performance throughout the software lifecycle. Ideally, this test and evaluation (T&E) [8] process should start in the architectural design phase of shipboard computing, as opposed to waiting until final system integration later in the lifecycle when it is more expensive to fix problems.

T&E of shipboard computing system QoS requirements typically employs software instrumentation techniques [22,3,18,24] that collect metrics of interest (*e.g.*, CPU utilization, memory usage, response of received events, and heartbeat of an application) while the system executes in its target environment. Performance analysis tools then evaluate the collected metrics and inform system developers and testers whether the system meets its QoS requirements. These tools can also identify bottlenecks in system and application components that exhibit high and/or unpredictable resource usage [21,11].

Although software instrumentation facilitates T&E of shipboard computing system QoS requirements, conventional techniques for collecting metrics are typically highly-coupled to the system's implementation [27,24,8]. For example, shipboard computing developers often decide during the system design phase what metrics to collect for T&E, as shown in Figure 1. Developers then incorporate into the system's design the necessary probes to collect these metrics from the distributed environment.

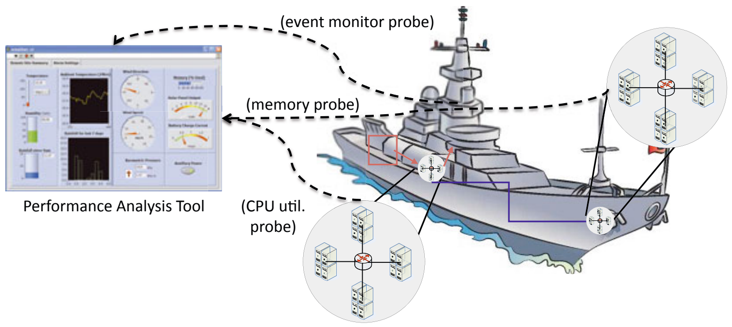


Fig. 1. Conventional Way to Instrument Shipboard Computing Systems

The drawback with a tightly-coupled approach is that shipboard computing developers must either (1) redesign the system to incorporate the new/different metrics or (2) use *ad hoc* techniques, such as augmenting existing code with the necessary interfaces without understanding its impact to the overall system's design and maintainability, to collect such metrics. Developers therefore need better techniques to simplify instrumenting shipboard computing systems for collecting and extracting metrics—especially when the desired metrics are not known *a priori*.

The *Embedded Instrumentation Systems Architecture* (EISA) [26] initiative defines a service-oriented *metadata-driven method* for heterogeneous data collection and aggregation in a synchronized and time-correlated fashion [26], as opposed to an *interface-centric method* [15] used in conventional DRE systems. Instead of integrating many interfaces and methods to extract and collect metrics into the system's design, EISA treats all metrics as arbitrary data that flows over a common reusable channel and discoverable via metametrics.[1] EISA thus helps reduce the coupling between system design and instrumentation logic incurred with the conventional T&E techniques described above [23], as shown in Figure 2.

Initial implementations of the EISA standard focused mainly on hardware instrumentation. To apply the EISA standard in the software domain, we developed the *Open-source Architecture for Software Instrumentation of Systems*

[1] Metametrics are metadata that describe metrics collected at runtime without knowing its structure and quantity *a priori*.

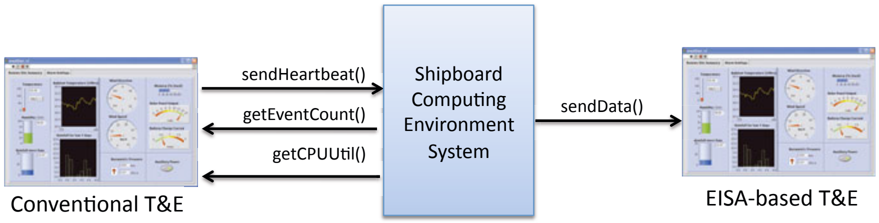


Fig. 2. Conventional Approach vs. EISA's Approach to T&E

(OASIS). OASIS is a service-oriented middleware framework that enables lightweight dynamic instrumentation. This paper discusses our insights and lessons learned while developing and applying OASIS to a representative shipboard computing project. This paper extends prior research efforts on OASIS [7] as follows:

– It discusses of design choices made while designing and implementing OASIS service-oriented middleware and framework;
– It analysis of current limitations of the OASIS service-oriented architecture, as well as insights on how such limitations can be addressed; and
– It discusses a strategy for using OASIS in cloud computing environments.

Our experiences gained from developing and applying OASIS to shipboard computing show that EISA's metadata-driven approach to instrumentation and data collection provides flexibility that can increase enterprise DRE system developers and tester's knowledge base and analytical capabilities of end-to-end QoS. OASIS also provides a solid foundation for addressing open problems associated with instrumenting enterprise DRE systems.

Paper organization. The remainder of this paper is organized as follows: Section 2 provides an overview the representative shipboard computing system we use as a case study for our work, and of OASIS focusing on key instrumentation challenges; Section 3 describes how OASIS addresses these challenges; Section 4 discusses OASIS's applicability to cloud computing environments; Section 5 compares OASIS with related work; and Section 6 presents concluding remarks.

2 Case Study: The Unified SHIP Platform

EISA-based tools have primarily been used to instrument DRE system hardware components (*e.g.*, sensor hardware components) [26]. These systems, however, are composed of both hardware and software components. Ideally, end-to-end QoS evaluation of shipboard computing systems should employ performance analysis of both hardware and software components.

To help evaluate EISA in a representative enterprise DRE system, we created the *Unified Software/Hardware Instrumentation Proof-of-concept (Unified SHIP)* platform, which provides a representative environment for investigating technical challenges of next-generation shipboard computing systems. The Unified SHIP platform contains software components (*i.e.*, the rectangles in Figure 3) implemented using the Component Integrated ACE ORB (`www.dre.vanderbilt.edu/CIAO`), which is a C++ implementation of the Lightweight CORBA Component Model [13]. Likewise, performance analysis tools are implemented using a variety of programming languages, such as C++, C#, and Java.

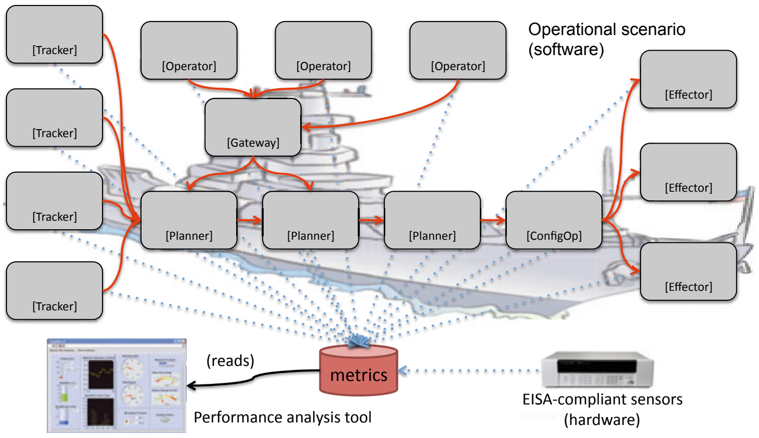


Fig. 3. Overview of the Unified SHIP Platform

Figure 3 also shows how the Unified SHIP platform consists of EISA-compliant sensor hardware components and a collection of software components that perform the following operational capabilities for shipboard computing systems: 4 components are trackers that monitor events in the operational environment, 3 components are planners that process data from the sensor components, 1 component performs configuration of the effectors, 3 components are effectors that react to commands from the configuration component, 3 components allow operators to send commands to the planner components, and 1 component is a gateway that authenticates login credentials from the operator components.

The directed line between each component in Figure 3 represents inter-component communication, such as sending an event between two different components. The hardware components of the Unified SHIP platform run in a conventional data center environment consisting of networked computers that run either Linux, Solaris, or Windows operating systems. The software components are thus bound to a particular set of hardware components.

Existing techniques for instrumenting shipboard computing systems assume software instrumentation concerns (*e.g.*, what metrics to collect and how to extract metrics from the system) are incorporated into the system's design. Since the Unified SHIP platform consists of hardware and software components at various degrees of maturity and deployment, it is hard to use existing instrumentation techniques to collect and extract metrics for QoS evaluation during early phases of the software lifecycle. In particular, developers and testers of the Unified SHIP platform faced the following challenges:

- **Challenge 1: Describing metametrics in a platform- and language-independent manner.** The heterogeity of the Unified SHIP platform's software and hardware components makes it undesirable to tightly couple performance analysis tools to the target platform and language of software and hardware components to collect and analyze metrics. Platform- and language-independent techniques and tools are therefore needed that will enable description of metrics collected from hardware and software components.
- **Challenge 2: Collecting metrics without a priori knowledge of its structure and quantity.** Metrics collected via instrumentation in the Unified SHIP platform come from heterogenous sources, which make it tedious and error-prone for system developers and testers to tightly couple the systems implementation to understand each metric and technology a priori. Techniques are therefore needed that will enable the collection of metrics from the Unified SHIP platform for QoS evaluation without a priori knowledge of which metrics are collected.

The remainder of this paper discusses how different design choices in OASIS enabled us to address these two challenges in context of the Unified SHIP platform.

3 Experiences Applying OASIS to the Unified SHIP Platform

This section codifies our experiences applying OASIS to the Unified SHIP Platform introduced in Section 2. For each experience discussed below we first introduce the topic and then provide a detailed account of our experience—both positive and negative when applicable.

3.1 Overview of OASIS

OASIS is dynamic instrumentation service-oriented middleware for DRE systems that uses a metametics-driven design integrated with loosely coupled data collection facilities. Metametrics are defined as software probes, which are autonomous agents that collect both system and application-level metrics. Listing 1.1 highlights an example software probe—written in OASIS's *Probe Definition Language (PDL)*—that collects memory statistics.

```
[uuid(ed970279-247d-42ca-aeaa-bef0239ca3b3); version(1.0)]
probe MemoryProbe {
  uint64 avail_physical_memory;
  uint64 total_physical_memory;
  uint64 avail_virtual_memory;
  uint64 total_virtual_memory;
  uint64 cache;
  uint64 commit_limit;
  uint64 commit_total;
};
```

Listing 1.1. Definition of a Memory Probe in OASIS

OASIS's PDL compiler uses such definitions to generate a stub, skeleton, and base implementation for the target programming language, and a *XML Schema Definition* (XSD) file that details the structure of a memory probe's data.

```
class MEMORYPROBE_STUB_Export MemoryProbe :
public virtual ::OASIS::Software_Probe {
public:
  /// The software probe's XML Schema Definition.
  static const char * __schema__;

  /// The metadata for the software probe.
  static const ::OASIS::Software_Probe_Metadata __metadata__;

  MemoryProbe (void);
  virtual ~MemoryProbe (void);

  /// Package the software probe's data.
  virtual int package_data (::OASIS::Software_Probe_Data_Packager & p);

  /// Recall the software probe's data.
  virtual int recall (const char * data, size_t length, int byte_order);

protected:
  int package_data (::OASIS::Software_Probe_Data_Packager & p, bool owner);
  int recall (ACE_InputCDR & input, bool owner);

public:
  // Setter and getter methods for avail_physical_memory
  void avail_physical_memory (ACE_UINT64 avail_physical_memory);
  ACE_UINT64 avail_physical_memory (void) const;

  // Setter and getter methods for total_physical_memory
  void total_physical_memory (ACE_UINT64 total_physical_memory);
  ACE_UINT64 total_physical_memory (void) const;

  // remaining setter/getter methods for probe.

protected:
  ACE_UINT64 avail_physical_memory_;
  ACE_UINT64 total_physical_memory_;

  // remaining member variables..
};
```

Listing 1.2. Code Snippet of the Memory Software Probe's Stub

Listing 1.2 shows a portion of the stub file generated from the memory probe PDL file in Listing 1.1. Likewise, Listing 1.3 shows a snippet of the memory software probe's skeleton, which is automatically generated from the PDL file.

```
class MEMORYPROBE_STUB_Export MemoryProbe_Impl :
public ::OASIS::Software_Probe_Impl_T <MemoryProbe> {
public:
 /// Default constructor
 MemoryProbe_Impl (void);

 /// Destructor
 virtual ~MemoryProbe_Impl (void);

 /// Flush the contents.
 virtual int flush (void);
};
```

Listing 1.3. Code Snippet of the Memory Software Probe's Skeleton

Finally, Figure 4 shows an overview of the process for generating source files from the a PDL file. The stub is used in the *Performance Analysis Tool* (shown as PAT in Figure 4) to recall data, the skeleton and base implementation are used in the instrumented application (App. in Figure 4) to collect metrics, and the XSD file is used for dynamic discovery of metrics.

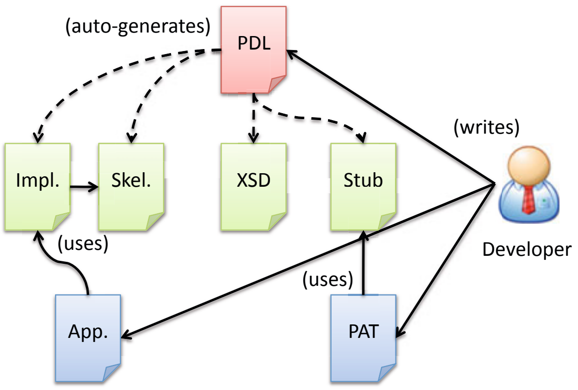


Fig. 4. Overview of Files Generated from a PDL Probe by OASIS

Figure 5 shows a high-level diagram of OASIS architecture and data collection facilities. As shown in this figure, this portion of OASIS consists of the following entities:

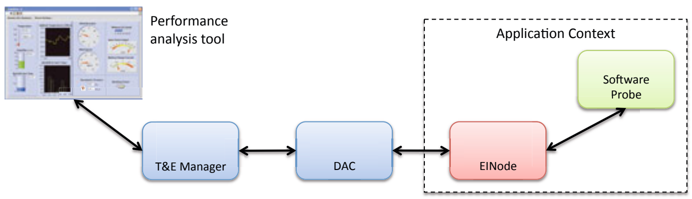


Fig. 5. Architectural Overview of the OASIS Middleware

– **Embedded instrumentation node (EINode)**, which is responsible for receiving metrics from software probes. OASIS has one EINode per application-context, which is a domain of commonly related data. Examples of an application-context include a single component, an executable, or a single host in the target environment. The application-context for an EINode, however, is locality constrained to ensure data transmission from a software probe to an EINode need not cross network boundaries, only process boundaries. Moreover, the EINode controls the flow of data it receives from software probes and submits to the data and acquisition controller described next. Each EINode is distinguished by a unique user-defined UUID and corresponding human-readable name.

- **Data acquisition and controller (DAC)**, which is a service that receives data from an EINode and archives it for acquisition by performance analysis tools, such as querying the performance of the latest state of component collected by a application-level software probe. The DAC is a persistent database with a consistent location in the target environment that can be located via a naming service. This design decouples an EINode from a DAC and enables an EINode to dynamically discover at creation time which DAC it will submit data. Moreover, if a DAC fails during at runtime the EINode can (re)discover a new DAC to submit data. The DAC registers itself when the test and evaluation manager (see below) when it is created and is identifiable by a unique user-defined UUID and corresponding human-readable name.
- **Test and Evaluation (T&E) manager** , which is a service that acts as the main entry point for user applications (see below) into OASIS. The T&E manager gathers data from each DAC that has registered with it. The T&E manager also enables user applications to send signals to each software probe in the system at runtime to alter its behavior, *e.g.*, by decreasing/increasing the hertz of the heartbeat software probe in the Unified SHIP platform scenario. This dynamic behavior is possible because the T&E manager is aware of all its DACs in the system, the DACs are aware of all its EINodes, and the EINodes are aware of all their registered software probes.
- **Performance analysis tools**, such as distributed resource managers and real-time monitoring and display consoles from the Unified SHIP platform, that interact with OASIS by requesting metrics collected from different software probes via the T&E manager. Tools can also send signals/commands to software probes to alter their behavior at runtime. This design enables system developers and testers and performance analysis tools to control the effects of software instrumentation at runtime and minimize the affects on overall system performance.

Figure 6 shows the integration of OASIS with the Unified SHIP platform. Each hardware and software component is associated with an EINode that contains a set of software probes (or instruments in the case of hardware components [23]) that collect and submit metrics for extraction from the system. When an EINode receives metrics from a software probe (or instrument), it sends it to a DAC for storage and on-demand retrieval. Performance analysis tools then request collected metrics via the T&E manager, which locates the appropriate metrics in a DAC.

Using this architecture, it is possible for the OASIS middleware framework to collect and analyze metrics without *a priori* knowledge of either the structure and complexity. The remainder of this section discusses how different design choices have impacted our experience using OASIS on the Unified SHIP Platform.

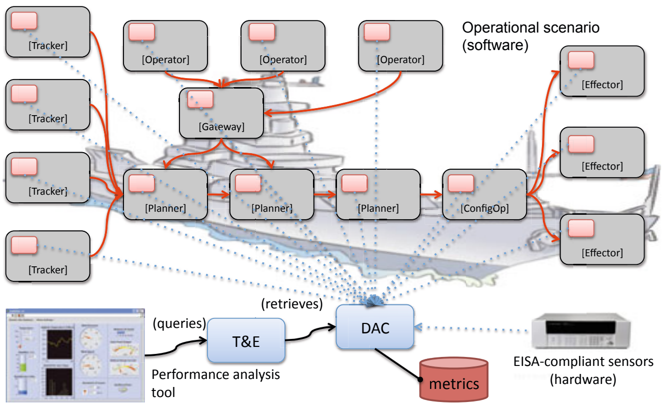


Fig. 6. Integration of OASIS with the Unified SHIP Platform

Experience 1: On Separating Metrics from Metametrics

In OASIS, metrics are separated from metametrics (*i.e.*, information that describes the metric's structure and types). The metametics are defined using XML Schema Definition (XSD) (see Listing 1.4 for an example), whereas metrics are packaged as blobs of data. As shown in Figure 7, the software probes package the data, prepend a header, and pass the metrics to the EINode. The EINode then prepends its header information and forwards it to the DAC. During this packaging process, however, no metametrics are stored with the actual metrics. Instead, the metametrics are sent to the DAC for storage when an EINode registers itself with a DAC.

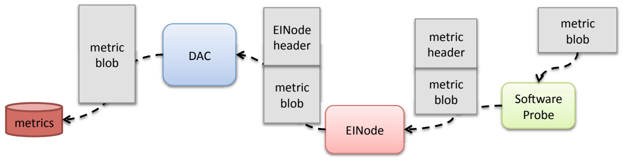


Fig. 7. The Metric Collection and Packaging Process in OASIS

Based on our experience applying OASIS to the Unified SHIP platform, we learned that separating metrics from metametics has the following advantages:

A1. Portability across different architectures. For example, the Unified SHIP platform consists of many different middleware technologies, such as the Common Object Request Broker Architecture (CORBA) [15, 16, 17], the Data Distribution Services [14], and Microsoft .NET [12]. None of these technologies, however, provide a straightforward or standard method for discovering metametrics that is portable across programming languages, middleware technologies, and platforms.

Moreover, the technologies used in the Unified SHIP platform assume that communication occurs between two strongly-typed endpoints. For example, in CORBA the client and server use strongly-typed interfaces that know what data types are sent across the network. The `CORBA::Any` element type is used in CORBA to send data without *a priori* knowledge. This element type knows the data type (*e.g.*, `tk_long`, `tk_short`, and `tk_boolean`). It does not, however, know the structure of complex types (*e.g.*, `tk_struct`), which makes it hard for the DAC to store metrics in its database.

For example, there is no standard method for discovering a metrics structure or serializing it to a blob of data using the generic `CORBA::Any` type. In some programming languages, such as Java and C#, it is possible to use reflection to support this requirement. This approach is only possible, however, because metametrics are built into the programming language. The serialization problem can also be solved by forcing the DAC to know each kind of metrics collected by a software probe. When a new metric arrives at the DAC, the DAC locates a corresponding software probe stub that can serialize data contained in the generic type. This approach, however, requires the DAC to know *a priori* all the software probes used in the Unified SHIP platform, which is not possible since developers can add new probes as they identify more data types to instrument and collect.

A2. Self-containment for offline analysis. Another advantage of separating metrics from metametrics is self-contained storage for offline analysis of data since the DAC stores both metametrics and metrics for a given execution of the Unified SHIP platform in a single database. This database can then be archived and recalled later to compare results of different test executions of the Unified SHIP platform. Moreover, developers can create new analysis tools at later dates to analyze different aspects of the data.

When applying OASIS to the Unified SHIP platform we have not yet found any disadvantages to separating metrics and metametrics. Its self-contained and standard method for storing and recalling metrics is platform-, language-, and technology-independent.

Experience 2: On Using XML Schema Definition to Describe Metametrics

Metametrics in OASIS are defined using XSD files (as shown in Listing 1.4).

When an EINode registers itself with the DAC, this information is sent to the DAC. The use of XSD to describe metametrics has the following advantage:

A3. GUI support. The main motivation for using XSD files to define metametics in OASIS is that there are existing tools that can create a *graphical user interface* (GUI) from a XSD file [19], which made it easier for Unified SHIP platform developers to visualize collected metrics as new software probes were added to the system. XSD is a verbose language since it is based on XML, *e.g.*, the metametrics in Listing 1.4 is approximately 300 bytes of data just to describe the metric's type name and its structure.

Using XSD to describe metametrics, however, has the following disadvantage:

```xml
<?xml version='1.0' ?>
<xsd:schema>
 <xsd:element name='probeMetadata' type='stateType' />
 <xsd:complexType name='stateType'>
  <xsd:sequence>
   <xsd:element name='component' type='xsd:string' />
   <xsd:element name='state' type='xsd:integer' />
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Listing 1.4. An Example XML Schema Definition that Describes Component State Metrics Collected by a Software Probe

D1. High processing overhead. Processing XSD files, which are XML files, can have high overhead and impact real-time performance. In OASIS, however, we do not process XSD files in real-time. Instead, they are processed at initialization time or when new metric types are discovered. Based on our experience with the Unified SHIP platform, the rate of discovering new metrics is not frequent enough to warrant using a less verbose method for defining metametrics—even when implementing generic performance analysis tools.

Experience 3: On Software Probe Definition and Stucture

Software probes in OASIS are defined using PDL. Developers define the metrics collected by a software probe, as shown in Listing 1.1 in the overview of OASIS. The OASIS compiler then generates the appropriate stubs and skeletons for using the software probe for instrumentation. The current implementation of OASIS does not support hierarchical software probe definitions, which means that each software probe definition is its own entity. This design choice, however, presented the following disadvantage:

D2. Lack of hierarchy increases instrumentation complexity. Based on our experience applying OASIS to the Unified SHIP platform, the lack of hierarchical software probe definitions increases the complexity of instrumenting such systems since developers must either (1) define a software probe such that it is too broad in scope, (2) define a software probe that is too narrow in scope, or (3) create separate software probes that collect similar information with slight differences.

The problem with broad software probes is that they collect more information than needed, *i.e.*, have fields with no data on different platforms. Likewise, narrow software probes must sacrifice data in certain situations, such as not collecting a specific metric on Linux since there is no equivalent metrics on Windows.

For example, in the Unified SHIP platform, software components execute on either a Windows or Linux platform. If developers want to collect memory metrics from either platform they would have to decide either to implement a broad or narrow software probe since each platform provides different information about memory usage, as shown in Table 1. If a broad software probe were implemented the Unified SHIP platform developers would have to ensure

Table 1. Comparison of Memory Metrics Collected on Linux vs. Windows

Linux (/proc/meminfo)	**Windows (MPI)**	**Description**
MemTotal	PhysicalTotal	Total amount of memory (avail. + used)
MemFree	PhysicalAvail	Total amount of memory free
Buffers		Amount of physical RAM used for file buffers
Cached	SystemCache	Amount of physical RAM used as cache memory
SwapCache		Amount of Swap used as cache memory
InActive		Total amt of buffer or page cache memory avail.
Active		Total amt of buffer or page cache memory
HighTotal		Total amt of memory in the high region
LowTotal		Total amt of non-highmem memory
LowFree		Amount of free memory of the low memory region
	KernelTotal	Sum of memory in paged and nonpaged kernel pools
	KernelPaged	Memory currently in paged kernel pool, in pages
	KernelNonpaged	Memory currently in nonpaged kernel pool, in pages
	PageSize	Size of a page, in bytes
SwapTotal		Total amt of physical swap memory
SwapFree		Total amt of swap memory free
Dirty		Total amt of memory waiting to be written back to the disk
WriteBack		Total amt of memory being written back to the disk
	CommitPeak	Max number of pages simultaneously in committed state
CommittedLimit	CommitLimit	Max memory available without extending paging files
Committed_AS	CommitTotal	Number of pages currently committed by the system
VmallocTotal	TotalVirtual	Total size of vmalloc memory area
VmallocUsed	TotalVirtual - AvailVirtual	Amount of virtual memory used
VmallocTotal - VmallocUsed	AvailVirtual	Amount of virtual memory available for allocation
VmallocChunk		Largest contiguous block of virtual memory that is free

that all metrics in Table 1 were covered. If a narrow software probe were implemented, conversely, they would only cover 8 common memory metrics (*i.e.*, MemTotal, MemFree, Cached, CommittedLimit, Committed_AS, VmallocTotal, VmallocUsed, and AvailVirtual), which also fails to account for mapping similar metrics to a common name and unit in the software probe's implementation.

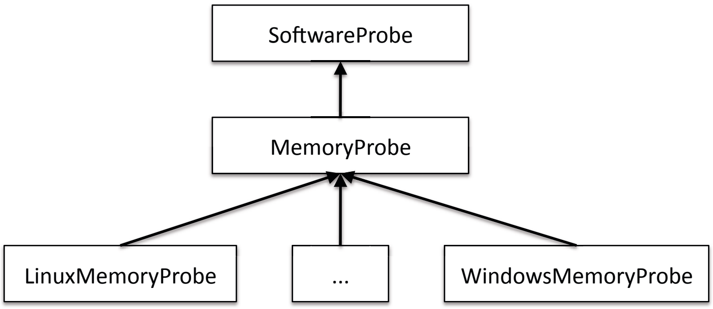


Fig. 8. An Example of Hierarchically Defining the Memory Software Probe in OASIS

Ideally, it should be possible for Unified SHIP platform developers to define hierarchical software probes to show relations between them. For example, Unified SHIP platform developers should be able to define a `MemoryProbe` that contains all metrics common across all platforms, as shown in Figure 8.

Each specific platform-specific memory probe (*e.g.*, `LinuxMemoryProbe` and `WindowsMemoryProbe`) then extends the `MemoryProbe` definition, as needed.

Based on our needs, we have realized that supporting hierarchical software probe definitions, however, has the following advantages:

A4. Metric reuse. When software probes are defined hierarchically in object-oriented programming languages, such as C++, C#, and Java, similar software probes can reuse metric definitions. Unified SHIP Platform developers therefore need not make critical decisions as to whether they should implement broad or narrow software probes.

A5. Platform-specific vs. general-purpose performance analysis tools. OASIS allows performance analysis tools to request real-time updates when new data arrives. The hierarchical software probe definitions give performance analysis tools greater flexibility when registering for real-time updates. For example, they can request general memory probe data, *i.e.*, data collected by a probe of type `MemoryProbe`, or specific memory probe data, *i.e.*, either `WindowsMemoryProbe` or `LinuxMemoryProbe` data. The Unified SHIP platform developers can therefore implement general-purpose performance analysis tools or platform-specific performance analysis tools.

Experience 4: Observing Other Roles of the T&E Manager

The T&E Manager is a service that acts as the main entry point into the OASIS architecture for performance analysis tools, as described in Section 2. This manager assists with gathering and correlating data requested by performance analysis tools. It also routes commands to software probes—via the DAC and EINode—to enable dynamic runtime behavior modifications, such as reducing its data collection frequency. Based on our experience applying OASIS to the Unified SHIP Platform, the T&E Manager has the following advantages:

A6. Domain-specific naming service. Based on our experience applying OASIS to the Unified SHIP platform, the T&E Manager is also a domain-specific naming service that keeps track of available DACs since the T&E manager must know all DACs available in test execution. Otherwise, it is hard for performance analysis tools to send commands to software probes. In addition, it is hard for performance analysis tools to register for real-time updates, which must be done by first locating an appropriate DACs via the T&E manager.

A7. Gateway and policy manager. Another role of the T&E Manager that we learned is that it can be a gateway/policy manager. In the Unified SHIP platform, some metrics collected by software probes should not be available to all performance analysis tools. For example, software metrics that would be considered sensitive metrics should not be available to performance analysis tools that do not have the correct privileges. The T&E Manager can therefore enforce such policies. Realizing this role of the T&E Manager also requires security enhancements at the DAC since metrics are stored in a database therefore for offline processing.

There is, however, the disadvantage to observing other roles of the T&E manager:

D3. The "god" T&E manager. If done incorrectly, the T&E manager could become a "god' T&E manager.[2] This superordination occurs when all roles of the T&E manager are condensed into a single entity, instead of decomposing it into distinct entities. We can overcome this design challenge via the Component Configurator [20] pattern, where each role is realized as a dynamically loadable component. The T&E manager then loads different components/roles as needed, ensuring the T&E manager is as lightweight as possible.

4 Towards Using OASIS in Cloud Computing Environments

This section evaluates OASIS's ability to support dynamic software instrumentation of applications in a cloud computing environments, such as Emulab [28], Amazon EC2 [25], and Microsoft Azure [4].

4.1 Motivation and Challenges of Instrumentation in Cloud Computing Environments

The previous sections provided an overview of OASIS and experiences gained from applying OASIS to the Unified SHIP platform. This platform is based on a

[2] This name is derived from the "god" class [21] software performance antipattern where a single class contains all functionality, instead of modularizing it into a family of related classes.

conventional data center architecture where software components are bound to their hardware components, as discussed in Section 2. While conventional data center architectures are common for today's enterprise DRE systems, we expect many types of these systems will migrate to cloud computing environments as its technology matures.

Several implementation and research challenges related to performance, scalability, data security, and data integrity will arise when deploying OASIS in a cloud environment. Key research challenges include transparently instrumenting an application in the on-demand without impacting its quality, evaluating different distributed middleware technologies and architectures for collecting and extracting metrics from cloud applications, investigating policy management mechanisms that control how collected metrics are accessed and used in analysis (both offline and online), merging and understanding data collected from many different sources, and enabling QoS-aware applications through real-time feedback.

4.2 A Strategy for Bringing OASIS to the Cloud

To enable OASIS to support deployment in a cloud, we are using an Emulab [28] environment (`www.isislab.vanderbilt.edu`) that enables us to construct clouds having different network characteristics for realistic experimentation purposes. Figure 9 shows how we are extending OASIS to operate in a cloud environment. As shown in this figure, software applications can run within a standard cloud computing environment. Within the cloud computing environment, portions of the OASIS service-oriented middleware and framework will already be executing. In particular, there are a number of DACs responsible for collecting information from different hosts and software applications. The location of the DAC responsible for collecting information, however, is unknown to software applications.

Each installment of an application in the cloud provides a T&E Manager. Only this Manager knows the location of the DACs, which are configured at

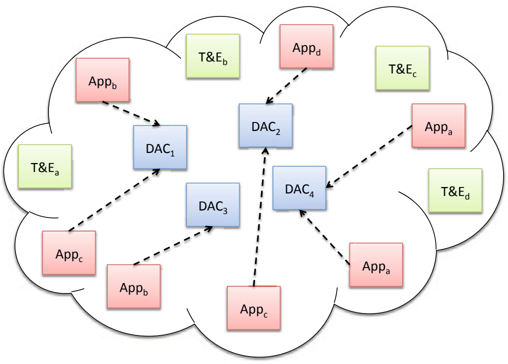


Fig. 9. Deploying OASIS in a Cloud Computing Environment

installation time. The T&E Manager accesses the different software probes executing on each host via the DAC. Applications can write custom PATs, which can be deployed into the cloud (if necessary). There is also a standard PAT that provides a view of all software probes (both active and inactive) in the software installment—similar to a web portal. This design enables users to manually control the behavior of software probes to (1) collect needed information and (2) minimize the impact on applications being instrumented.

The deployment of OASIS into the cloud computing environment is still a work-in-progress. When complete, it will provide a foundation for addressing the key research challenges outlined above, which must be resolved without any application awareness. As explained above, applications are not physically aware of the location for any OASIS entities operating in the cloud. The planned deployment of OASIS is therefore congruent with the transparency requirement for addressing key research challenges in cloud computing environments.

5 Related Work

This section compares our work on OASIS with related work.

Dynamic binary instrumentation (DBI) frameworks. Pin [10] and DynamoRIO [2] are examples of DBI frameworks. Unlike OASIS, both Pin and DynamoRIO do not require modification of existing source code to enable instrumentation. Instead, software developers use Pin to execute the application, and during the process Pin inserts points of instrumentation based on C/C++ user-created instrumentation tools—similar to performance analysis tools in OASIS. Although DBI frameworks address different problems, we believe they can work together in that software probes can be implemented as third-party analysis tools for DBI frameworks. This combination would allow OASIS to collect instrumentation information from the DBI framework that instruments a DRE system in real-time without modifying any of the existing source code—as done traditionally with OASIS.

DTrace [3] is another DBI framework. Unlike Pin and DynamoRIO, DTrace provides a scripting language for writing performance analysis tools. DTrace also has the ability to write custom software probes, which can be easily integrated into DTrace's collection facilities. DTrace's software probe design is therefore similar to OASIS in that it is extensible without *a priori* knowledge. It differs in that software metrics cannot be extracted from the host machine where software instrumentation is taking place.

Distributed data collection. Distributed Data Collector (DDC) [6] is a framework for collecting resource metrics, such as CPU and disk usage, from Windows personal computers (PCs). In DDC, software probe metrics are collected from PCs and stored in a central location. Unlike OASIS, each software probe's metrics are stored its own file, which is then parsed by analysis tools. OASIS improves upon this design by storing all metrics in a single database, instead of separate files. Likewise, OASIS's data collection framework is platform-, language-, and architecture-independent (*i.e.*, not bound to only Windows PCs and Windows-specific software probes).

General-purpose middleware solutions can be used for distributed data collection. For example, the DDS is an event-based middleware specification that treats data as first-class entities. This concept is similar to OASIS in that events are similar to software probe metrics. The main difference is that DDS is a strongly-typed middleware solution in that both endpoints know the data type *a priori*. Moreover, there is not standard way to serialize the data in a DDS event. This therefore makes it hard to store metrics in the DAC's database.

6 Concluding Remarks

The test and evaluation (T&E) of enterprise DRE system QoS during early phases of the lifecycle helps increase confidence that the system being developed will meet it functional and QoS requirements. Conventional T&E instrumentation mechanisms, however, are tightly coupled with the system's design and implementation. This paper therefore described how design choices in OASIS's implementation of the EISA standard helped reduced these coupling concerns. In addition, it also highlighted several observations about different design choices that are being addressed in the OASIS service-oriented middleware framework.

Based on our experience with OASIS, we found the following open research challenges, which extend the research directions presented in [7], remain when instrumenting enterprise DRE systems:

Complex event processing. Complex event processing [5] involves processing many different events and streams of data, across all layers of a domain, identifying meaningful events, and determining their impact on a given concern, such as performance, functionality, and scalability. Each software probe in OASIS can be viewed as stream of data and the DAC can be viewed as a data source with historical records. Likewise, performance analysis tools can register for real-time delivery of software probe data.

Our future research is therefore focusing on implementing complex event processing support in OASIS. Adding this support will be hard because the traditional use of complex event processing engines involves viewing results via a graphical user interface, which is considered one form of a performance analysis tool in OASIS. In reality, many different performance analysis tools (such as real-time monitoring and feedback performance analysis tools) should be able to leverage complex event processing support. Likewise, complex event processing has not been applied to general-purpose instrumentation middleware for enterprise DRE systems.

Data fusion and data integration. Data fusion [1] is the process of combining data from multiple sources for inference. The motivation for data fusion is that multiple data sources will be more accurate that a single data source. Although data fusion can be used to support complex event processing, it is a separate research area. Data integration [9], however, is the process of combining data from different sources to provide a unified view.

When we examine the OASIS middleware framework—and each of its entities that play a major role in collecting and storing data (*e.g.*, software probe, EINode, and DAC)—it is clear that data fusion and data integration techniques can be applied readily. The challenge, however, is understanding how both data fusion and data integration can be integrated with the real-time aspects of OASIS. Our future research is therefore focusing on addressing these challenges in OASIS to provide a general-purpose middleware solution for data fusion and data integration in OASIS.

Cloud computing environments. Cloud computing environments will eventually become the norm for enterprise DRE systems, as they are in the process of becoming for standard enterprise applications. Unfortunately, cloud computing environments do not provide dynamic instrumentation capabilities. Instead, cloud computing environment users only see metrics (such as CPU, disk, and network usage) that the cloud computing environment can collect. We outline several key research challenges in Section 4 for migrating OASIS to a cloud computing environment, which will enhance the instrumentation capabilities of cloud computing environment. Our future work will investigate these research challenges.

As we apply OASIS to other application domains, such as resource-constrained embedded systems, mobile devices, and cloud computing environments, we will continue identifying new research challenges. Since OASIS is an open-source middleware framework, it provides an effective foundation for ensuring that solutions to these open research challenges will be available to the T&E community.

OASIS is currently integrated into CUTS and is freely available for download in open-source format from `cuts.cs.iupui.edu`.

References

1. Bleiholder, J., Naumann, F.: Data Fusion. ACM Computing Surveys 41, 1:1–1:41 (2009), `http://doi.acm.org/10.1145/1456650.1456651`
2. Bruening, D., Garnett, T., Amarasinghe, S.: An Infrastructure for Adaptive Dynamic Optimization. In: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization, CGO 2003, pp. 265–275. IEEE Computer Society, Washington, DC, USA (2003), `http://portal.acm.org/citation.cfm?id=776261.776290`
3. Cantrill, B., Shapiro, M.W., Leventhal, A.H.: Dynamic Instrumentation of Production Systems. In: Proceedings of the General Track: 2004 USENIX Annual Technical Conference, pp. 15–28 (June 2004)
4. Chappell, D.: Introducing the windows azure platform (2009) (retrieved May 30, 2010)
5. Dekkers, P.: Complex Event Processing. Master's thesis, Radboud University Nijmegen, Nijmegen, Netherlands (October 2007)
6. Domingues, P., Marques, P., Silva, L.: Distributed Data Collection through Remote Probing in Windows Environments. In: 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2005, pp. 59–65. IEEE (2005)

7. Hill, J.H., Sutherland, H., Staudinger, P., Silveria, T., Schmidt, D.C., Slaby, J.M., Visnevski, N.: OASIS: An Architecture for Dynamic Instrumentation of Enterprise Distributed Real-time and Embedded Systems. International Journal of Computer Systems Science and Engineering, Special Issue: Real-time Systems (April 2011)
8. Hudgins, G., Poch, K., Secondine, J.: The Test and Training Enabling Architecture (TENA) Enabling Technology For The Joint Mission Environment Test Capability (JMETC) and Other Emerging Range Systems. In: Proceeding of U.S. Air Force T&E Days (2009)
9. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems PODS 2002, pp. 233–246. ACM, New York (2002), http://doi.acm.org/10.1145/543613.543644
10. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. SIGPLAN Notes 40, 190–200 (2005)
11. Menasce, D.A., Dowdy, L.W., Almeida, V.A.F.: Performance by Design: Computer Capacity Planning By Example. Prentice Hall PTR, Upper Saddle River (2004)
12. Microsoft Corporation: Microsoft.NET Framework 3.0 Community (2007), http://www.netfx3.com
13. Object Management Group: Light Weight CORBA Component Model Revised Submission, OMG Document realtime/03-05-05 edn. (May 2003)
14. Object Management Group: Data Distribution Service for Real-time Systems Specification, 1.2 edn. (January 2007)
15. Object Management Group: The Common Object Request Broker: Architecture and Specification Version 3.1, Part 1: CORBA Interfaces, OMG Document formal/2008-01-04 edn. (January 2008)
16. Object Management Group: The Common Object Request Broker: Architecture and Specification Version 3.1, Part 2: CORBA Interoperability, OMG Document formal/2008-01-07 edn. (January 2008)
17. Object Management Group: The Common Object Request Broker: Architecture and Specification Version 3.1, Part 3: CORBA Component Model, OMG Document formal/2008-01-08 edn. (January 2008)
18. O'Hair, K.: The JVMPI Transition to JVMTI (2006), http://java.sun.com/developer/technicalArticles/Programming/jvmpitransition
19. Radha, V., Ramakrishna, S., kumar, N.P.: Generic XML Schema Definition (XSD) to GUI Translator. In: Chakraborty, G. (ed.) ICDCIT 2005. LNCS, vol. 3816, pp. 290–296. Springer, Heidelberg (2005)
20. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, vol. 2. Wiley & Sons, New York (2000)
21. Smith, C., Williams, L.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley Professional, Boston (2001)
22. Srivastava, A., Eustace, A.: ATOM: A System for Building Customized Program Analysis Tools. In: PLDI 1994: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation, pp. 196–205 (1994)
23. Stefani, A., Xenos, M.N.: Meta-metric Evaluation of E-Commerce-related Metrics. Electronic Notes in Theoretical Computer Science (ENTCS) 233, 59–72 (2009)
24. Tan, Z., Leal, W., Welch, L.: Verification of Instrumentation Techniques for Resource Management of Real-time Systems. J. Syst. Softw. 80(7), 1015–1022 (2007)

25. Varia, J.: Cloud architectures. White Paper of Amazon, jineshvaria. s3. amazonaws. com/public/cloudarchitectures-varia. pdf (2008)
26. Visnevski, N.: Embedded Instrumentation Systems Architecture. In: Proceedings of IEEE International Instrumentation and Measurement Technology Conference (May 2008)
27. Waddington, D.G., Roy, N., Schmidt, D.C.: Dynamic Analysis and Profiling of Multi-threaded Systems. In: Tiako, P.F. (ed.) Designing Software-Intensive Systems: Methods and Principles, Idea Group (2007)
28. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proc. of the Fifth Symposium on Operating Systems Design and Implementation, pp. 255–270. USENIX Association, Boston (2002)

Dynamic Event-Based Monitoring in a SOA Environment

Fabio Souza[1], Danilo Lopes[1], Kiev Gama[2], Nelson Rosa[1], and Ricardo Lima[1]

[1] Federal University of Pernambuco, Center of Informatics
{fns,dvl,nsr,rmfl}@cin.ufpe.br
[2] University of Grenoble, LIG laboratory, ADELE team
kiev.gama@imag.fr

Abstract. There is an increasing need to monitor quality attributes (e.g., performance and availability) in SOA environments. Existing approaches to monitor these attributes (commonly referred to as QoS attributes) do not allow reconfiguration while services are in execution. This paper presents a generic QoS-aware SOA mechanism able to monitor runtime quality attributes of services. The solution is dynamic, event-based, extensible, transparent and lightweight in such way that the performance impact on the application is minimal and the overall mechanism is easily reconfigurable. To validate our solution, we propose a typical SOA scenario and evaluate its impact on the performance of the service execution.

Keywords: SOA, QoS, monitoring, events.

1 Introduction

Service-Oriented Computing (SOC) [5] proposes the utilization of loosely coupled entities (services) as first-class elements. Services are published in a registry where they can be dynamically discovered, selected and bound by service consumers. Providers, consumers and registry compose the traditional "triangle" which is central to the architectural style referred to as Service-Oriented Architecture (SOA).

In SOA, applications are developed as service compositions in which services are selected based on contracts defining their functional interface and non-functional attributes. Different QoS attributes can be considered in SOA environments. According to their nature, these attributes can be classified as deterministic (e.g., price) or non-deterministic (e.g., response time, throughput, availability).

The dynamic and distributed nature of SOA environments and the intrinsic characteristics of non-deterministic QoS attributes suggest the need for a continuous monitoring system, which checks whether a service is providing the expected level of quality.

Despite the increasing demand for continuous monitoring systems, their development is complex. Key concerns are the number of quality attributes and the costs related to monitoring them. In fact, a monitoring system can hamper the overall performance of a system and negatively impact the availability of monitored services. Another aspect is how to introduce new and not foreseen quality attributes to be monitored, without interrupting service execution.

This paper proposes a dynamic, event-based, extensible, non-obtrusive and lightweight QoS monitoring system that supports continuous monitoring of non-deterministic attributes in SOA environments. The solution is dynamic because it can be enabled, disabled or reconfigured (e.g. changing the set of QoS attributes) at runtime without impacting the availability of services. It is event-based because its components interact by sending and receiving event notifications. It is extensible because the support for monitoring unforeseen attributes is provided using documented extension points. It is non-obtrusive because services are not aware of monitoring tasks. Finally, it is lightweight because the impact on performance is minimized by splitting data gathering and processing in asynchronous phases.

The rest of this paper is organized as follows: Section 2 presents the proposed mechanism in details, whilst Section 3 presents its experimental evaluation. Related work is presented in Section 4, followed by the conclusions and future work in Section 5.

2 Monitoring System

A monitoring system dynamically gathers, interprets and acts on data concerning an application while it executes. Monitoring systems can be used in different contexts such as security monitoring, correctness checking and debugging.

Monitoring systems are configured through some kind of requirement specification that defines the target entities and the properties (or attributes) they are expected to observe. To collect data, a monitoring system attaches some sensors, which are triggered by the occurrence of events (e.g. the arrival of a message). The collected data are sent to an analyzer, which will evaluate them and eventually notify some handlers. Event handlers are responsible for the monitor's reaction and can perform simple activities such as logging or take sophisticated decisions such as starting a reconfiguration process.

Figure 1 presents a high-level view of the proposed monitoring system's architecture, which is composed by three modules: *management agent*,*data gathering center* and *data processing center*. To promote loose-coupling, the communication between these modules is based on internal services and events.

The proposed monitoring system has been implemented atop of an OSGi environment. This technology was selected because it has some very distinctive characteristics: it provides an inherently dynamic and modular platform, allowing loose coupling between modules thanks to a Service-Oriented Architecture. The OSGi specification defines a framework that leverages Java's dynamic class loading feature for enhancing modularization and also introduces a runtime where modules can be installed, started, stopped, updated or uninstalled without stopping the application.

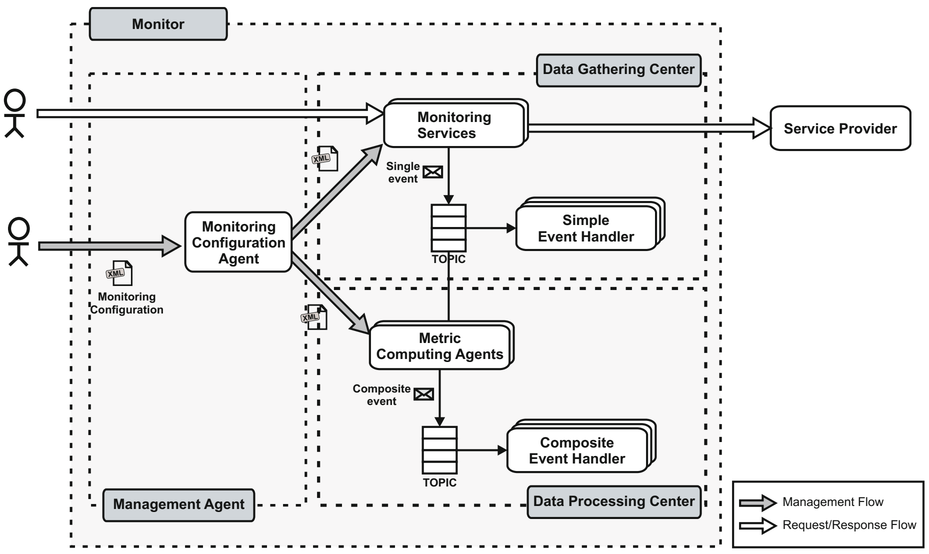


Fig. 1. High level representathion of the QoS monitoring system

2.1 Management Agent

The *management agent* is responsible for providing management interfaces that can be accessed through a local or remote management console. These interfaces support suspension and resumption of the monitoring system as well as the definition of the *monitoring configurations*, which define the target services and the quality attributes that are expected to be monitored.

When a *monitoring configuration* is defined, the *management agent* verifies whether there are monitoring services (see section 2.2) responsible for dealing with the corresponding quality attributes. If this condition is satisfied, the monitoring activity starts with the agent notifying the *data gathering* and the *Data Processing* centers. Otherwise, the *monitoring configuration* is marked as pending, being reanalyzed when the agent is notified about the registration of new monitoring services.

2.2 Data Gathering Center

The *data gathering center* uses the Invocation Interceptor pattern to intercept requests and replies exchanged between consumers and providers. It collects QoS-related data and wraps them in simple events that are asynchronously sent to the *data processing center*, reducing the impact of the monitoring activities since the aggregated metrics (e.g. average response time) are computed in parallel. This computed data can then be stored in a service registry enabling QoS-based service selection.

The *data gathering center* is composed of two kinds of components: monitoring services and simple event handlers. Essentially, a monitoring service is responsible for collecting data required to compute metrics related to a particular QoS attribute, e.g., response time. To perform this task, it uses sensors, which are implemented as QoS interceptors dynamically (at runtime) interposed between consumers and providers.

To connect the QoS interceptors to a service, we developed a *chainer interceptor*, which is added to the beginning of each service's chain when it is published (this is done through middleware configuration and is transparent to the developer). This interceptor modifies the chain by adding the QoS interceptors provided by the monitoring services. To select the QoS interceptors to be included, the chainer must know what QoS attributes are configured for the service. This information is in the *monitoring configurations* related to the service, so the chainer is notified each time one of these configurations is defined. In summary, based on the QoS attributes defined in the configurations, the chainer discovers the monitoring services that it has to interact with in order to obtain the QoS interceptors (sensors) that ought to be plugged in the chain.

In order to support an extensible collection of QoS attributes, our monitoring system should be able to dynamically support new QoS interceptors. As these interceptors are provided by the monitoring services, our monitoring system must dynamically discover when a new monitoring service is available. To cope with this requirement, we use a service registry that triggers an event each time a new monitoring service is published. When our *management agent* is notified about the occurrence of this event, it realizes that it must allow the definition of *monitoring configurations* that enable the monitoring the corresponding QoS attribute.

Another key component in our solution is the publisher, which should be added as one of the last interceptors in the chains. Its purpose is to forward the collected data to the components responsible for processing them (simple event handlers). The communication between publisher and handlers is asynchronous. In fact, we adopt an event-based solution built atop of a publish/subscribe communication model. Simple event handlers can be used, for example, to keep historical data. In our solution, the event handlers are services that are dynamically discovered. This design decision enables defining alternative ways to handle QoS data at runtime. In fact, our monitoring mechanism includes a default handler implementation that forwards monitoring events to the Data Processing Center where the aggregated QoS metrics are computed.

In summary, the architecture of the Data Gathering Center is quite flexible once it does not prescribe what QoS attributes can be monitored, or how the collected data should be processed. In fact, it allows the definition, design and deployment of new monitoring services which can be dynamically plugged into the proposed monitoring system, providing support for the monitoring of unforeseen QoS attributes without impacting running services. New simple event handlers can also be independently deployed, defining alternative ways to deal with the monitored data. In fact, this architecture supports the subscription of various simple event handlers simultaneously, each one of them manipulating the same data in a different manner.

2.3 Data Processing Center

To compute aggregated QoS metrics, different design alternatives can be considered. A usual solution is to store simple events containing QoS related data in a database and process them *a posteriori*. This approach is directly supported by our monitoring mechanism through the development of a simple event handler that stores simple events in a database. This database is processed by a background task that computes the metrics.

That approach has some important drawbacks. As mentioned before, supporting dynamic service composition requires runtime monitoring. At runtime a monitoring system can produce a huge volume of primitive events. Storing these events in a database forces frequent I/O operations and imposes unlimited storage demands. Besides, extracting valuable information from such volume of data is expensive and time-consuming. So, the capacity of taking real-time decisions based on events is clearly affected. To perform online monitoring, we usually need to process a continuous flow of events that is constantly changing, i.e., an event stream. In such situations, the use of an Event Stream Processing (ESP) engine is recommended.

In our context, the monitoring events generated by the *data gathering center* compose event streams that are forwarded to the *data processing center*. The core component of this center is the Esper engine, which computes aggregated metrics defined through queries provided by a set of metric computing agents. These agents are build dynamically (at runtime) when the *data processing center* is notified that there is a new monitoring configuration available. In fact, it is important to mention that, besides the data concerning the target service and the QoS attribute, a monitoring configuration defines which statistical operator should be used to calculate the aggregated QoS metric. It also contains information concerning the interval (in terms of time or number of events) that the agents will use to define their queries.

Aggregated metrics are wrapped in composite events which are forwarded to a collection of composite event handlers that have subscribed to receive metrics computation results. These handlers analyze the metrics using embedded rules and take the necessary actions. In order to dynamically support new ways of dealing with aggregated metrics, our composite event handlers are designed as services and are dynamically discovered from the service registry.

Different composite event handlers can be realized in a SOA environment. An example is a composite event handler that verifies SLA contracts and notifies parties when violations are detected [4]. Although our monitoring mechanism does not deal with SLA verification, it includes a handler that updates our QoS-aware service registry based on the aggregated metrics enabling the proposition of a QoS-aware composition layer.

3 Experimental Evaluation

To validate the monitoring system, some experiments were performed on a controlled environment. The validation scenario consists of service consumers periodically invoking a service provider to get the price of stocks. To evaluate the impact of the monitoring system, two performance metrics were considered, namely “throughput” and “response time”. Both metrics were measured from the consumer’s point of view. We use these metrics to evaluate the behavior of the system with and without monitoring.

The experiments were performed on an isolated network composed by 3 hosts: 1 Pentium dual-core with 3GB running a CXF DOSGi environment that supports the service provider; 1 Pentium dual-core with 2GB emulating the service consumers modeled as threads in a test plan; and 1 Pentium dual-core with 1 GB running an Apache Derby database (storage of the stock prices) and a Zookeeper-based service registry.

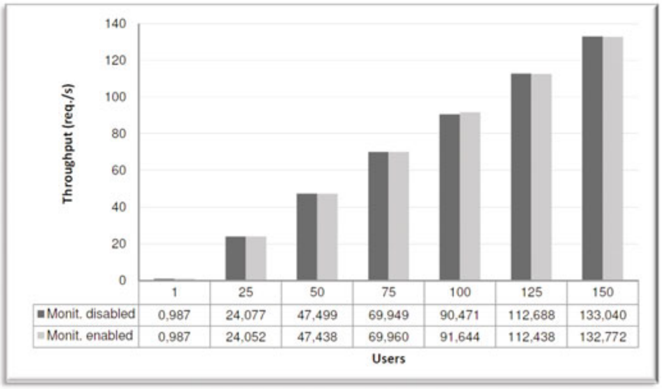
Different workloads were designed by varying the number of simultaneous consumers. Experiments containing 1, 25, 50, 75, 100, 125 and 150 consumers (submitting 1 request/second) were performed on two different configurations: 1) monitoring mechanism enabled; 2) monitoring mechanism disabled. Each user submits 1.000 requests.

For the purpose of the experiments, the data gathering center includes a single monitoring service measuring the time required to process each request at the service provider side. The monitoring mechanism wraps the measured time in a primitive monitoring event that is sent to a monitoring topic to which a single primitive event handler is subscribed. This handler forwards each received event to the data processing center.

To compute the aggregated metrics, the data processing center uses a collection of queries provided by agents defined through the monitoring configurations. In our experiments, they define queries that compute maximum, minimum, and average processing time over windows composed of collections of 500 events. These metrics are wrapped in derived events that are forwarded to derived event handlers. In our environment, just one handler is subscribed to receive the events. It reads the metrics and updates our QoS-aware service registry.

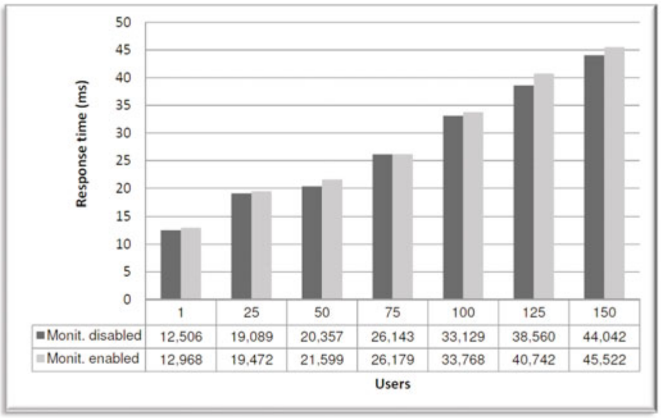
Figure 2 shows the throughput with and without enabling the monitoring mechanism. Despite the increasing number of consumers, these measures are very close. The greatest difference is around 1.3% (at 100 users). This is an indication that the monitoring infrastructure has a low impact on the performance.

The response time measures are presented in Figure 3. It shows that the measures with and without monitoring are close. In fact, the greatest difference is around 6% (at 50 simultaneous users). According to Figures 2 and 3, we can infer some information concerning the system’s operation status that is not related to monitoring itself. Figure 2 shows that the throughput varies linearly with the workload. Furthermore, regardless of the submitted load, the system’s throughput is reasonably close to its nominal value, meaning that the system can process requests at a rate close to that demanded by the users. As an example, a demand equivalent to 150 simultaneous users, each one operating at 1 request/s, is 150 requests/s. The measured throughput (with and without monitoring) is around 133 requests/s, i.e., around 89% of the users’ demand. Figure 3 shows that the response time also varies linearly with the workload.



	1	25	50	75	100	125	150
■ Monit. disabled	0,987	24,077	47,499	69,949	90,471	112,688	133,040
■ Monit. enabled	0,987	24,052	47,438	69,960	91,644	112,438	132,772

Fig. 2. Throughput on the consumers' side



	1	25	50	75	100	125	150
■ Monit. disabled	12,506	19,089	20,357	26,143	33,129	38,560	44,042
■ Monit. enabled	12,968	19,472	21,599	26,179	33,768	40,742	45,522

Fig. 3. Response time on the consumers' side

4 Related Work

Several researches in academia and industry discuss the importance of monitoring QoS attributes in SOA environments [5], although there is neither a standard QoS model for services nor a standard way to use QoS data. For instance, Baresi and Guinea [2] propose an extension to a BPEL engine using aspects, allowing the definition of self-supervising BPEL processes enhanced by supervision rules (e.g., declarative monitoring directives, recovery strategies). However, the rules are static and after compiled they cannot be changed dynamically.

A monitoring specification language called SECMOL [3] is used for specifying functional requirements and QoS constraints in BPEL processes. It allows splitting activities of data collection, aggregation and computation, but lacks the capacity of dealing with unforeseen QoS attributes at runtime.

A probing tool for client-side monitoring called QUATSCH [6] generates stubs, based on WSDL files, in which performance code is woven. These stubs are used to submit requests to the web services and collect QoS data, which are combined with information obtained through low level sniffing in order to infer server side performance data. This is a non-invasive approach that captures QoS as seen by the client. However, it is not "live data" and may not represent the current state of the service provider.

A QoS monitoring mechanism proposed in [4] collects data from both client and server sides. The client-side monitoring is based on QUATSCH [6], and server-side monitoring is based on Windows Performance Counters (WPC) and processes real service invocations. Besides monitoring, this paper proposes a SLA validation mechanism which is supported by an event stream processor. This solution, however, is not able to dynamically support unforeseen QoS requirements.

A discussion [7] of possible approaches to enable monitoring QoS on the client and server sides includes: performing low-level package sniffing, interposing a proxy between consumers and providers, and modifying a SOAP engine. They chose this last option in order to enable intercepting consumer and provider exchanges. They implemented a prototype using Apache Axis arguing that the impacts in terms of memory and time were not significant. However their solution is not easily extensible, and cannot cope with new QoS requirements without modifying its code.

Finally, a complete approach for enabling end-to-end support of non-functional properties in web services scenarios is presented in [1]. They define a representation of those properties, an architecture for measuring and updating QoS data, and an implementation based on Axis. Like our solution, they also use interceptors for supporting data collection. However, their interceptors cannot be changed dynamically and new metrics can only be supported through a redeploy.

5 Conclusion and Future Directions

QoS monitoring in Service-Oriented Architectures allows verifying quality attributes (e.g., availability, performance) of services at runtime. Monitoring these attributes is fundamental for implementing QoS-aware service compositions, as well as for validating Service-Level Agreements. Although there is no consensus on which and how QoS attributes should be monitored, there are different researches and industrial efforts that already collect and analyze QoS data in non-standardized ways. Existing mechanisms lack flexibility concerning on-the-fly reconfiguration of QoS monitoring and data analysis. In this paper, we proposed an extensible, dynamic, event-based, flexible, lightweight and transparent monitoring mechanism for continuous monitoring QoS observable attributes in a non-obtrusive way. This mechanism was implemented atop of CXF/DOSGi and validated using a typical SOA scenario.

For future work we plan to enhance our monitoring events with semantic information in order to enable the development of an autonomic management infrastructure and to use it for validating SLAs and for detecting SLA violations.

References

1. Agarwal, V., Jalote, P.: Enabling end-to-end support for non-functional properties in web services. In: Service-Oriented Computing and Applications (SOCA), pp. 1–8 (2009)
2. Baresi, L., Guinea, S.: Self-supervising BPEL processes. IEEE Transactions on Software Engineering 37(2), 247–263 (2011)
3. Guinea, S., Baresi, L., Spanoudakis, G., Nano, O.: Comprehensive monitoring of BPEL processes. IEEE Internet Computing PP(99), 1 (2009)
4. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Comprehensive qos monitoring of web services and event-based sla violation detection. In: Proc. of the 4th International Workshop on Middleware for Service Oriented Computing, pp. 1–6. ACM, New York (2009)
5. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer 40(11), 38–45 (2007)
6. Rosenberg, F., Platzer, C., Dustdar, S.: Bootstrapping performance and dependability attributes ofweb services. In: Proc. of the IEEE International Conference on Web Services, pp. 205–212. IEEE Computer Society, Washington, DC, USA (2006)
7. Thio, N., Karunasekera, S.: Automatic measurement of a qos metric for web service recommendation. In: Proc. of the 2005 Australian conference on Software Engineering, pp. 202–211. IEEE Computer Society, Washington, DC, USA (2005)

A SIP-Based Network QoS Provisioning Framework for Cloud-Hosted DDS Applications

Akram Hakiri[1], Aniruddha Gokhale[2], Douglas C. Schmidt[2], Berthou Pascal[1], Joe Hoffert[2], and Gayraud Thierry[1]

[1] CNRS; LAAS, 7, avenue du Colonel Roche,
Universit de Toulouse; UPS, INSA, INP, ISAE; LAAS;
F-31077 Toulouse, France
[2] Institute for Software Integrated Systems, Dept of EECS
Vanderbilt University, Nashville, TN 37212, USA
{Hakiri,Berthou,Gayraud}@laas.fr,
{a.gokhale,d.schmidt,jhoffert}@vanderbilt.edu

Abstract. The growing trend towards running publish/subscribe (pub/sub)-based distributed real-time and embedded (DRE) systems in cloud environments motivates the need to achieve end-to-end quality-of-service (QoS) over wide-area networks (WANs). The OMG Data Distribution Service (DDS) is a data-centric middleware that provides fast, scalable and predictable distribution of real-time critical data. The DDS standard, however, provides QoS control mechanisms that are confined only to the middleware residing at end-systems, which makes it hard to support DRE pub/sub systems over WANs. A promising solution to this problem is to integrate DDS with the Session Initiation Protocol (SIP), which is an IP-based signaling protocol that supports real-time applications involving voice, video, and multimedia sessions via the QoS mechanisms in IP networks.

This paper describes our approach to bridge the SIP protocol and DDS to realize DDS-based applications over QoS-enabled IP WANs by overcoming both inherent and accidental complexities in their integration. An exemplar of the proposed approach for IP DiffServ networks is described, which uses the Common Open Policy Server (COPS) protocol to assure QoS for cloud-hosted DRE pub/sub applications. To distinguish the DDS traffic from other best-effort traffic in the cloud environment, our approach uses the COPS-DRA protocol as a generic protocol for automatic service-level negotiation and the integration of this protocol in an overall QoS management architecture to manage service levels over multiple domains deploying different QoS technologies.

Keywords: Cloud, End-to-End QoS, DDS, SIP, COPS, DiffServ.

1 Introduction

Many publish/subscribe (pub/sub)-based distributed real-time and embedded (DRE) systems operate in heterogeneous environments that receive data from a large number of sensors and multimedia sources and stream it in real-time to remote entities. Examples of such DRE systems include unmanned air vehicles, video surveillance, on-demand video transmission, online stock trading, air traffic management, power grid control, shipboard computing environments, and weather monitoring. These types of DRE systems often typically optimize the performance and scalability of applications and provision/control key network resources [18].

The Object Management Group (OMG)'s *Data Distribution Service* (DDS) [14] is a data-centric pub/sub middleware that simplifies application development, deployment, and evolution for DRE systems. DDS delivers needed capabilities of DRE systems via powerful quality-of-service (QoS) policies that enable applications to control data delivery properties at each distributed node. In addition, DDS supports fast and predictable distribution of real-time critical data over heterogeneous networks.

Most applications of DDS in pub/sub DRE systems either use stable networks, such as local area networks, or networks whose scale is small and whose properties are controllable. With a growing trend [1] towards supporting pub/sub DRE systems in the cloud, however, it becomes necessary to realize the QoS mechanisms for pub/sub DRE systems in wide-area networks (WANs) that are characteristic of cloud environments. The DDS specification, however, does not currently support key QoS properties of pub/sub DRE systems over WANs because the DDS middleware resides on end-systems and thus defines only mechanisms that control end-system properties, such as OS-level parameters and tuning network parameters for the connecting link.

For example, DDS provides no mechanisms for provisioning/controlling end-to-end QoS over WANS. The lack of these mechanisms makes it hard to use DDS to assure end-to-end QoS for large-scale DRE systems running in clouds. Key challenges required to support cloud-based DDS applications include optimizing the performance and scalability of WAN deployment over fixed and wireless access technologies while also providing network-centric QoS provisioning.

A promising approach to address these challenges is to integrate DDS with the Session Initiation Protocol (SIP) [16] and Session Description Protocol (SDP) [17]. Together, SIP and SDP provide a powerful capability to convey new enhanced services. Examples of such services include disseminating information about end-systems, identifying the originator of a session, and identifying multimedia content in a session initiation request that might contain pictures, signals from sensors, or a personalized ring-tone.

Despite the strengths of SIP/SDP, they do not take into account the application needs of pub/sub DRE systems. In particular, the desired QoS policies of pub/sub DRE systems cannot be seamlessly supported today both by DDS middleware on end-systems and SIP/SDP mechanisms in WANs. To bridge the gap between DDS and SIP/SDP, therefore, this paper describes a SIP-based QoS management architecture using a proxy SIP QoS module, which implements both DDS QoS policies and standardized QoS mechanisms. Our approach defines a new SIP *Signaling Class of Service* (S-CoS) for transferring signaling messages for sessions whose characteristics are specified in extensions we defined for SDP messages. A key benefit of our approach is that it require no modifications to applications, which can continue to use standard DDS QoS policy and programming interfaces.

To demonstrate our approach, we have prototyped our QoS-enabled SIP proxy for cloud environments using DiffServ IP networks, which makes it possible for interworking DDS sessions with QoS-enabled IP networks by integrating it with DiffServ mechanisms such as the Common Open Policy Server (COPS) [10] protocol. COPS aims to exchange policy information between a policy server (Policy Decision Point or PDP) and its clients (Policy Enforcement Points or PEPs), using TCP transport protocol for reliable exchange of messages between PEPs and PDPs. In our case, therefore, COPS acts as the protocol for QoS requests and for admission control to achieve the desired QoS for DDS traffic over DiffServ IP networks.

The remainder of this paper is organized as follows: Section 2 describes how we integrated DDS with SIP/SDP to support pub/sub-based DRE systems in cloud environments; Section 3 showcases our integrated solution in a DiffServ network, focusing on the signaling procedure and QoS provisioning; Section 4 compares our research with related work; and Section 5 presents concluding remarks.

2 Supporting DDS in Cloud Environments Using SIP/SDP

This section describes the SIP/SDP-based framework we developed to support DDS session in cloud environments, which comprise WANs. To better understand our solution, we first explain how applications use DDS and provide an overview of the SIP/SDP protocols. We then describe the enhancements we made for SIP/SDP media description to support DDS sessions over WANs.

2.1 Overview of Underlying Technologies

This section summarizes the DDS, SIP, and SDP technologies.

DDS and DDS sessions. DDS is a middleware standard for distributed real-time application that simplifies application development, deployment and maintenance and provides fast, predictable distribution of real-time critical data over heterogeneous networks. The DDS architecture consists of two layers. The *Data-Centric Publish Subscribe* (DCPS) layer provides efficient, scalable, predictable, and resource-aware data distribution. The *Data Local Reconstruction Layer (DLRL)* provides an object-oriented facade atop the DCPS so that applications can access object fields rather than data and defines navigable associations between objects.

The DCPS DDS entities include *topics*, which describe the type of data to read or write, *data readers*, which subscribe to the values or instances of particular topics, and *data writers*, which publish values or instances for particular topics. Properties of these entities can be configured using combinations of the DDS QoS policies.

SIP and SDP. The Session Initiation Protocol (SIP) is an IP-based application-level signaling protocol defined by the IETF in RFC3261 [16]. It plays a major role in establishing, maintaining, modifying, and tearing-down multimedia sessions between two distributed end-points. SIP focuses on IP core networks for all services: mobility, scalability, multimedia services, high bit rate, and dissemination of call/service control and user data. From the service plane (application signaling) point of view, SIP simplifies the network operation and management by automating the QoS provisioning, and providing the level of QoS by facilitating admission control with the network QoS negotiation mechanisms.

Multimedia sessions can be described using the Session Description Protocol (SDP) [12]. Information in the session defines a syntax to characterize the multimedia session, such as the types of media in the session, the available support for each of the media types, the contact information (IP address (IP/port) where packets should be sent) and bandwidth requirements for receiving the session. SDP is therefore decomposed into three main descriptors: (1) *session-level descriptions*, which describe characteristics of the whole session; (2) *time descriptions*, which indicate time-related aspects of the session, and (3) *media descriptions*, which characterize the different media present in the session.

2.2 SIP/SDP Enhancements for Cloud-Based DDS

Section 1 alluded to the limitations in realizing cloud-based DDS applications. We present our SIP/SDP enhancements to support cloud-based DDS applications as shown in Figure 1. Cloud-based DDS applications are expected to be located at the access network. Specifically, SIP is used with the discovery protocol to discover end-systems while SDP is used to encapsulate DDS QoS policies.

We propose to let SIP messages carry DDS QoS policies, which in turn are used in reserving the required network resources. The signaling part of SIP is therefore independent of the session being established and of the mechanism used to describe it. This approach provides the way to distribute this information between potential participants in the session, *i.e.*, the QoS information exchange is made transparent to the DDS clients. The proxy SIP negotiates on behalf of

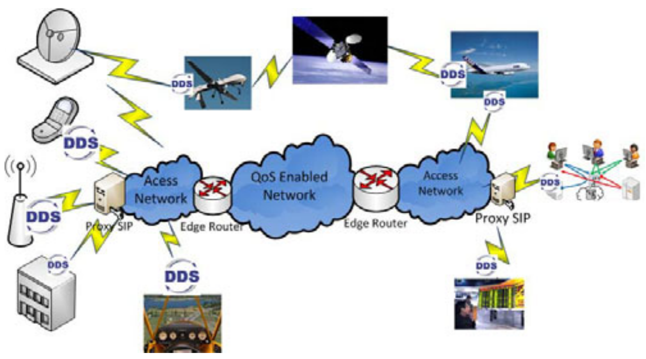


Fig. 1. A SIP-based, DDS-enabled QoS Support Architecture

the clients with the network QoS mechanisms. The DDS QoS policies are mapped into SIP/SDP messages, and carried within a new SDP media session description attribute we added to the SIP protocol stack, and used by the Proxy SIP server.

2.3 End-System Architecture

Figure 2 presents an architectural view of interfaces and functional entities that comprise our QoS support architecture, which is located both at the sender and the receiver sides.

To establish a communication between SIP applications (without DDS) and DDS applications attached to SIP entities, the following entities are required: (1) The SIP User Agent (UA) acts as an interface between the QoS-enhanced SIP Proxy (Q-SIP) and DDS application, (2) SIP Proxy which maintains media session between DDS pub/sub participants. The remainder of this section describes the specifics of the SIP Proxy and the new SDP attributes we added.

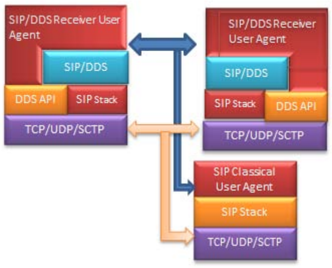


Fig. 2. DDS Mapping Interface with a SIP Stack

DDS application. DDS application are not integrated within the SIP-based environment, as shown in Figure 2. Indeed, because these applications contain a rich set of domain users, it was less convenient to implement them within the SIP interface. Moreover, for extensibility reasons, to make use of general SIP applications like IMS applications, we determined the application to remain independent of the SIP/DDS implementation.

SIP user agent (UA). The UA is not integrated within the application. The UA acts as an interface between the Proxy SIP and DDS application. This approach preserves interoperability properties since any SIP-based application can communicate with a DDS-based application.

Proxy server. The goal of the proxy server architecture is to support both wired and wireless communication between DDS-based pub/sub system and other wireless access points. Moreover, DDS application can access a network via a mobile client. The Proxy SIP is considered as a network element. Hence it is included in the network vision to allow the session management.

The proxy SIP is present in all phases of the SIP-based communication including registration (to access to the registrar server matching two SIP-UA requests), establishment (forcing all SIP and SDP messages to pass across it with the header "record route"), and it is able to understand the information about the session. The memory footprint of the proxy SIP implementation is low.

2.4 Proxy SIP Signaling Mechanism

The proxy SIP receives SIP requests/responses and forwards them through the network on behalf of the user agent. Proxies are essentially routers that are also capable of generating requests and responses. Two key types of SIP proxies are supported:

– **Inbound proxies**, which handle incoming requests for an administrative domain and also routes incoming requests to the appropriate UA within the domain it is responsible for.
– **Outbound proxies**, which help the UAs to route outgoing requests. UAs are usually configured to route all their requests to an outbound proxy, which will route the requests for them. Since outbound proxies include all the capabilities of inbound proxies, in the rest of this paper we focus only on outbound proxies.

2.5 SDP Extensions

We extended SDP messages to support the media description of the DDS session. The new SDP attributes deal with the session management and are used by the SIP User Agent (UA) that we collocate with both the sender and the receiver to indicate how the QoS may be achieved. An SDP message consists of a set of lines of text of the form:

$$< attribute >=< value >$$

The offer/answer [17] model for SDP assumes that endpoints somehow establish the QoS required for the media streams they establish. For DDS session, however, more than one QoS service is available, which requires the capability to negotiate which QoS mechanism to use for a particular media stream. We therefore created new DDS QoS attributes and syntax to incorporate the signaling procedure.

Table 1 and 2 describe the enhancements to the SIP/SDP header and the new attributes we added to support the specific DDS QoS requirements to be assured by cloud environments. The caller and callee terminals exchange SIP messages that include the resource reservation demands and the current status to support the QoS demands in each direction.

The "qos-dds" token identifies a QoS mechanism that is supported by the entity generating the session description. A token that appears in a "qos-dds-send" attribute identifies DDS QoS policies supported by the DataWriters to assist the resource reservation for traffic sent by the publisher generating SDP

Table 1. Description of the DDS QoS policy Attributes

attribute	Description
Deadline	DataReader expects a new sample updating the value of each instance at least once every deadline period. DataWriter indicates that the application commits to write new value for each instance managed by this DW at least once every deadline period. The Deadline is a duration "0 0".
Latency	The delay from data writing until its delivery is inserted in the receiver's application cache and the receiving application is notified of the fact. The Latency Budget is duration "0 0".
Reliability	Is the reliability of the service. Could be Reliable "R" or Best Effort "BE".
Priority	The Transport Priority is a hint to the infrastructure used to set the priority of the underlying transport used to send data in the DSCP field for DiffServ. This is presented as an integer value.

Table 2. New SDP Attributes for DDS-based QoS Support

attribute	value
"qos-dds"	Deadline Latency Reliability Priority

messages. A token that appears in a "qos-dds-recv" attribute identifies the DDS QoS policies that can be supported by the DataReader to reserve the resources for traffic coming from the DDS publisher.

For example, the SDP session description presented in Table 3 offers video communication requesting a total bandwidth of 64 kilobits per second as described in line "b" with *qos-dds-send* and *qos-dds-recv* attributes.

2.6 Semantics of SIP/SDP Extensions

Offer/answer behavior. When using *qos-dds-send* and *qos-dds-recv* attributes, an offer/answer negotiation is done between the publisher and the subscriber to allow endpoints to load a list of QoS mechanisms. The participants negotiate the direction in which the QoS mechanisms are exchanged with respect to both preconditions [2] and DDS changeable table parameters [14]. Participants may also use other QoS parameters (such as those described in [8]) to allow bandwidth, jitter, and delay parameters to be negotiated at the same time with per-flow resources reservation for IntServ/RSVP, and per-Class for DiffServ infrastructure.

Offer behavior. Publishers include *qos-dds-send* flow in the publication direction to inform subscribers about the QoS parameters supported by the sender. Similarly, a participant can use *qos-dds-recv* attributes to specify which kind of QoS policies can be supported at the receive direction.

Answer behavior. After receiving an offer from remote participant user agent with the *qos-dds-send* attributes (in the Invite message), the proxy SIP forwards them to the COPS-PEP for translation into network QoS query to reserve network resources in the edge router. In the receive direction, those attributes correspond to the QoS query a participant uses in a *qos-dds-recv* attributes in the answer. When a participant receives an offer with *qos-dds-recv* attributes, the

Table 3. Example of an **m** Line with "qos-dds" Attributes

```
v = 0
o = alice 2890844526 2890842807 IN IP4 1.2.3.4
s =
c = IN IP4 1.2.3.4
t = 0 0
m = video 51372 RTP/AVP 31
b = AS:64
a=qos-dds-send: 0 0 0 0 R 12
a=qos-dds-recv: 0 50 0 5 R 12
```

proxy SIP uses the corresponding QoS translation mechanisms it supports in the send direction, and then includes them in a *qos-dds-send* attributes.

In all the cases described above—and once the offer/answer exchange completes—both sender and receiver use the attributes embedded in the SDP messages to perform resource reservation. If a participant does not succeed in using the mechanism in *qos-dds* attributes, it should use default QoS settings defined in [14]. If a participant unsuccessfully tries all the common QoS mechanisms in the *qos-dds* attributes for a given direction, there may be network entities (such as Proxy SIP entities) that cannot process the message.

2.7 Integrating All the Pieces Together

Having described the architectural pieces of our solution above, we now describe how we integrated these pieces and show the interactions among the entities that help realize cloud-based DDS applications. The specific DDS–SIP/SDP interactions that take place in our solution are summarized below and illustrated in Figure 3.

- DDS entities interested in publishing data specify the DDS media description by sending a SIP control message to the core network. If the network core accepts the query for resources the required resources are allocated, at which point the application can securely and reliably exchange data. When a call setup is initiated, the caller application calls the SIP session setup through the proxy SIP. The proxy SIP encountering the caller message starts the QoS session to interact with the remote Proxy SIP and the QoS mechanisms in the edge router.
- When the host application specifies or modifies its QoS requirements, it sends an INVITE message to the proxy SIP which intercepts it and redirects to the destination for notification (offer/response contract). Subsequently, the receiver node adapts its DDS QoS policies with those notifications and sends a response to its proxy SIP, which notifies the COPS Policy Enforcement Point (PEP) with the new QoS requirements.
- In the discovery phase, DDS entities (DomainParticipant, DataWriters and DataReaders) in distributed nodes find out about each other using the default EndPoint Simple Discovery Protocol (ESDP) [9]. During this process

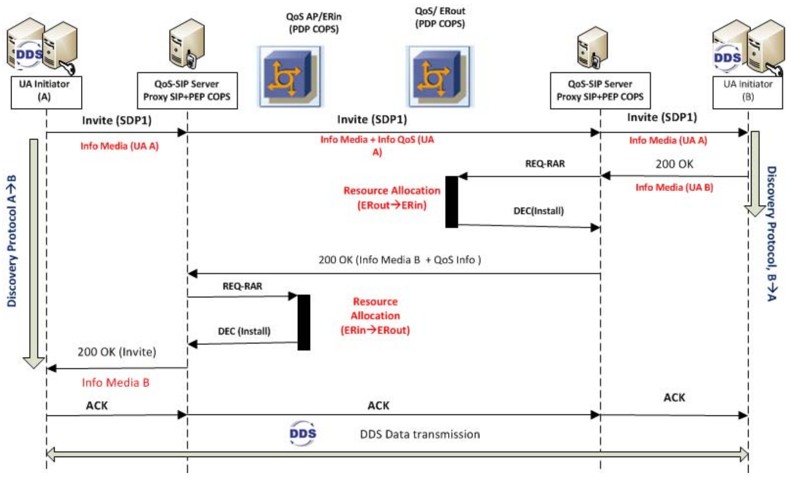


Fig. 3. Architecture of the Signaling Flow Description

the DomainParticipant details are communicated automatically to all other DomainParticipants in the same Domain by sending/receiving discovery messages—also known as participant DATA messages. The discovery phase integrates the SIP service to enable the communication between SIP and non SIP applications, that is, it assesses the interoperability between distributed applications that are intended to share data with DDS applications.

– The transport phase actually sends and receives messages over the underlying network. This phase integrates the network QoS services, *e.g.*, resource reservations.

3 Realizing Cloud-Based DDS on DiffServ IP Networks

This section describes a concrete realization of our cloud-based DDS solution using SIP/SDP for DiffServ IP networks.

3.1 Implementation Overview

Below we describe how we implemented key elements of our architecture shown in Figure 1.

SIP proxy. In our implementation, we modified the JAIN SIP Presence Proxy [6] to support the DDS QoS policies as described by the SIP standard: we added those policies as new attributes in the Jain SIP Stack [7], and modified the proxy to support the communication with COPS-DRA server (explained below) which we created for this purpose. In fact, the QoS requests are handled by the edge router, *Juniper Mi7 in our case*, which implements all mechanisms to perform the admission control decision with the help of the COPS-DRA server.

COPS-DRA. The COPS-DRA is used on the interface between the edge router and the Bandwidth Broker (BB). The BB is the COPS-PDP entity in charge of controlling the provisioning of resources for the network. It offers the capability of provisioning resources to the local DDS client and eases requesting the update of allocated resources.

Edge router. We used Juniper M7i, which is a multi-service edge router that incorporates network backbones needing reliable, secure and high-performance IP WAN connectivity, Internet access and services. It delivers rich Layer 3 services including granular per logical interface QoS, hardware-based IPv6, and multicast.

To describe the new offer/answer extension to support signaling for DiffServ infrastructure, we used the attributes shown in Table 4, which are defined in [8] and developed them using the same JAIN SIP Stack.

Table 4. SDP Attribute Values for Cloud-based DDS QoS in DiffServ Networks

```
attribute =/ qos-dds-send-attr
attribute =/ qos-dds-recv-attr
qos-dds-send-attr='qos-dds-send' ':'[[SP]qos-dds*(qos-dds))]
qos-dds-recv-attr='qos-dds-send' ':'[[SP]qos-dds*(qos-dds))]
qos-dds = '0 20' '0 10' 'R' '12'
qos-dds = '0 0' '0 50' 'U' '20'
qos-dds = '0 0' '0 0' 'U' '46'
```

3.2 Signaling Procedure

We now describe how the DDS and SIP/SDP mechanism are integrated in the context of DiffServ networks and demonstrate the QoS provisioning capabilities provided with the help of the COPS-DRA bandwidth broker (called also COPS Server) for pub/sub DDS sessions in cloud environments.

Contemporary DDS implementations contain a set of commonly used transport plug-ins (such as UDPv4 and UDPv6), as well as extensible transports (such as Secure WAN Transport). The DDS transport priority QoS describes the DSCP Field used by the DiffServ edge router classification. Packet classification features provide the capability to partition the DDS traffic into multiple priority levels. During packet classification, the DDS client performs a lookup and assigns a QoS label to the packet. The QoS label identifies all QoS actions to be performed on the packet and from which queue the packet is sent. The QoS label is based on the DSCP value in the packet and decides the queuing and scheduling actions to perform on the packet.

In our architecture, the DDS transport priority QoS policy is encoded within SIP/SDP messages, and partitioned into 4 Classes of Service (CoS) in the context of COPS-DRA-based network: Real Time (RT), Non Real Time 1 (NRT-1), Non Real Time 2 (NRT-2) and Standard (STD) as shown in Table 5.

Table 5. DDS Class of Service in the DSCP Field

PHB	CoS	dropping probability	DSCP	
			Binary	Decimal
EF	RT		101110	46
AF1	NRT1	Low	010100	20
AF2	NRT2	Medium	001100	12
BE	STD	High	000000	0

Each edge router applies the following DiffServ concept: packets belonging to real-time streams are handled in the router according to the same Per Hop Behavior (PHB) defined for the RT CoS, and packets belonging to near real-time streaming and High Throughput Data are handled in the router according to the PHBs defined for the NRT-1 CoS and NRT-2 CoS, respectively.

QoS policies mapped into SIP messages are used by the Proxy SIP. When the host changes its QoS requirements, therefore, the SIP message is intercepted by the proxy SIP to be redirected to the destination for notification (offer/response contract). The receiver node then adapts its DDS_QoS policies with those notifications and sends a response back to its proxy, which notifies the PEP with the new QoS requirements to adapt its class of service (CoS).

3.3 Session Management and QoS Setup for DDS on the Cloud Network

After the SIP UA is configured to make and receive calls, the processing starts with the registration phase in the "Registrar" (user registration within the SIP location database) to make all users accessible from all other SIP Servers: the Proxy SIP delivers SIP REGISTER request from UA to the "Registrar" to check the authentication, authorization and accounting using the authentication headers [16]. The scenario shown in Figure 4 presents a detailed description of the signaling procedure and all mechanisms included in the QoS reservation.

The signaling procedure shown in Figure 4 comprises the exchange of messages from DDS-SIP (*i.e.*, Start DDS session, Invite, 183 Progress, Prack, 200 OK Prack, 200 OK and Ack) and the messages from COPS protocols (*i.e.*, Decision, Report, Reserve and Response). This procedure is started when a caller (user A) sends a standard SIP *Invite* message and is completed upon receiving a corresponding standard SIP *ACK* message. The *200 OK* message that reaches the callee (user B) already informs it about successful connection establishment. Accordingly, the setup duration is given by the time elapsed between the moment of sending Invite and receiving *200 OK* message.

The following example shown in Table 6 describes an IP communication session described with SDP. This corresponds to a classical usage of SDP message to convey real-time media component including audio and video transported over RTP and including the PCM codec used for the audio (payload type=0), and also a media line for video with the H.261 codec (payload type=31).

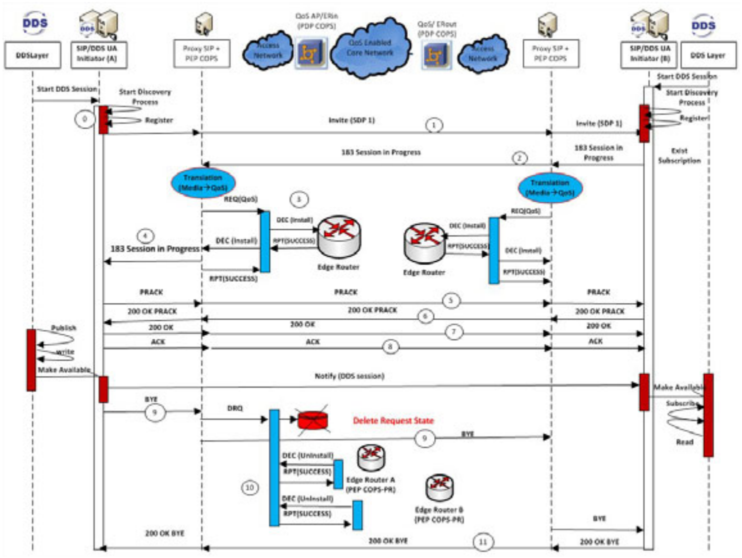


Fig. 4. QoS Support and Session Establishment for DiffServ

The bandwidth requirements are included in line 'b', the QoS requirements in the sending direction are included within the 'qos-dds-send', that is, the application requires low latency, minimum deadline, reliable transport (reliability in DDS means using NAck requests of the reliable multicast protocol) and its required PHB behavior (AF) is fixed to 12. With respect to Table 5, this application has medium dropping probability. The DiffServ mechanisms in the edge router (classification, policing, shaping, dropping) take those parameters from the COPS-DRA server and process packets with respect to this query.

Session setup. Participants exchange `INVITE` and `Session in Progress` messages to negotiate the media description and the QoS setting that the session should fulfill: An INVITE message is sent with all information about the

Table 6. Body SDP Carried Within the SIP Invite Message

v = 0
o = alice 2890844526 2890844526 IN IP4 host.ocean.com
s = -
c = IN IP4 ahkiri.laas.fr
t = 0 0
m = audio 49170 RTP/AVP 0
a = rtpmap:0 PCMU/8000
m = video 51372 RTP/AVP 31
a = rtpmap:31 H261/90000
b = AS:512
a=qos-dds-send: 0 0 0 0 R 12
a=qos-dds-recv: 0 50 0 5 R 12

participant (IP address, port number, QoS attributes, CoS description, media description) to establish the session between the publisher and the subscriber. The Proxy SIP "A" then adds the address of record to attract all other messages to pass through. Since the destination address is found in the local database, the message will be redirected to the destination, precisely to the remote SIP UA which starts the `Session in Progress` message with its media description. After the session description attributes, *e.g.*, QoS attributes, are known, the QoS negotiation for resource reservation begins.

The *Invite* message carries the caller URI within the SIP header and the session specification within the body SDP (session description, media description, source port, codec, DDS-QoS) as shown in Table 6. The proxy SIP intercepts the invite message, and on the basis of the information given in the body SDP decides whether to start a QoS session or not. If so, it inserts the required descriptors within the Invite message and forwards it to the callee.

The message can also be intercepted by any other SIP server encountered in its destination or any QoS-aware SIP proxy (since DDS QoS attributes are not recognized by classical SIP server, they are just ignored when it processes the Invite message). Hence, the Proxy SIP is present during all steps of SIP-based communication described below, including the registration (to access to the registrar server matching two SIP-UA requests) and the session establishment (forcing all SIP and SDP messages to pass across it with the header "record route").

Since the proxy SIP has all required information to request the QoS reservation to the edge router for the SIP UA (B), it sends a request to the COPS-DRA server (Bandwidth Broker or BB) to translate the media description fields into a QoS specification understandable by the edge router. BB is in charge of automating the QoS resource reservation process. It sends QoS requests to the router to install them (DEC or Decision, RPT or Report, REQ or Request).

At the same time, the QoS information is sent within the SIP `183 Session Progress` message to the other proxy SIP at the border of the access network of the SIP UA (A). Similarly, this information is stored by the SIP Proxy to maintain a trace of the current session, called `QoS State`. Consequently, it sends a QoS request to the COPS-PDP in its domain to claim the QoS reservation at the edge of the network. The PDP connected to the edge router has to automate the QoS reservation process based on the QoS information it has received from the proxy SIP.

At this point, the caller UA (A) sends a Prack (Reliability of Provisional Responses in SIP) message to confirm the `183 Session Progress` message has been received and waits for the `200 Ok Prack` message sent by the caller UA (B) to notify the session setup. Moreover, the caller proxy SIP inserts the `Caller ER` field into its message including the IP address of the caller endpoint. The caller proxy SIP uses this field to specify the remote address of the endpoint where the reservation request have to be send to QoS provider.

The caller proxy SIP adds its VIA field to the message intercepted from the sender UA (A) in which it adds some specific information (IP address, source port, caller address) to maintain trace of the QoS State. Finally, the session establishment is confirmed when both caller and caller Proxy SIP exchange `200 OK` and `ACK` SIP messages. The sender application performs the data delivery to the remote participants, since the signaling path is established as described herein, the data path is taken by the data packets according this signaling process.

QoS negotiation. The RT CoS covers the requirements to offer strict QoS assurance of low packet delay, low packet losses, and low jitter. As a consequence, we focus on admission control methods that support such assurance. The NRT-1 CoS covers the requirements to offer strict QoS assurance of low packet losses and medium packet delay. In NRT-2 CoS for inter-domain link, we follow the QoS requirements related to the High Throughput Data end-to-end CoS that are expressed by the assurance of minimum throughput R_{min}. We do not consider any QoS assurance support for STD CoS, though an amount of inter-domain link capacity and buffer must be dedicated to this CoS, as well.

Moreover, since DDS is based on Offer/Request contract between distributed entities, this model can be used to enhance adaptivity and robustness of communication. For example, consider user agent A in Figure 4 that is responsible for the resource allocation. In this case Proxy A follows the “183 session progress” and performs the translation of the “qos-dds” attributes included within the media information to QoS requirements.

The “qos-dds” attributes include the *DiffServ Code Point* (DSCP) priority that the DDS application adds to SIP/SDP signaling messages. The PEP uses this value for traffic classification in each border router. Moreover, QoS settings (such as latency budget, deadline in the “a” line in Table 3) are translated into bandwidth, latency, and jitter. The PEP then sends the “REQ” message to the PDP in its domain to perform the resource reservation using the message context with the appropriate identification and authorization of the participant. As a result, the PDP analyzes the request (consults its database for verification), performs the DiffServ resource allocation using COPS Provisioning mechanisms, and sends a decision “DEC” to the PEP including the DSCP value for the media stream which generates a report message “RPT” to indicate the success of the decision.

The “*183 session progress*” message is also sent to proxy A to indicate successful resource allocation in its direction. The DDS discovery protocol performs at both direction endpoints discovery from the information given by its near proxy and registrar to connect different DDS domains. The rest of the process is performed as described above and the DDS streams are exchanged between participants. After a pair of remote participants have discovered each other, they can move on to the Endpoint Discovery phase, which is how DataWriters and DataReaders find each other. During this phase DDS matches DataWriters and DataReaders by sending publication/subscription declarations in DATA messages including information (Globally Unique ID or GIU, QoS) about the application.

Terminating the session and releasing the resources. DiffServ entities should release the allocated resources after the session termination. The proxy SIP therefore performs the resource liberation from the PEP and the PDP. In particular, the "uninstall" message is sent to perform this operation and a negotiation process executed by PDP and PEP.

4 Related Work

This section compares our work on DDS Middleware-based QoS provisioning with related research on pub/sub capabilities over WANs, which are an important part of cloud environments. We therefore focus on related work comprising pub/sub efforts for WANs.

QoS management in content-based pub/sub middleware: QoS management in content-based pub/sub middleware includes the *Publish/ Subscribe Applied to Distributed Resource Scheduling (PADRES)* [15]. PADRES allows powerful content-based routing mechanism based on the message content instead of IP-based routing. For example, a mechanism for congestion avoidance to optimize the network resource usage in Content-Based Routing Networks is presented in [4]. Despite this advantage, in practice core network resource allocation algorithms need to know more about an application and how to process its QoS requirements within the core routers.

In our approach, DDS provides the DCPS services described in Section 2.1. These services make it possible to pub/sub-structured data as a portion of distributed relational information model. Moreover, our approach focuses on how to use existing middleware and mapping it to provide the QoS for existing applications.

Network QoS broker in middleware: Related work in this area focuses on integrating the signaling process into the QoS provisioning mechanisms. For example, message-based signaling middleware for the control plane to offer per-class QoS is described in [21]. In practice, however, it is insufficient to deploy routers to assure the QoS between two hosts at the control plane. Likewise, a network communication broker to provide per-class QoS for multimedia collaborative applications is presented in [5]. This work, however, supports neither mobility service management nor scalability since it adds an interface to the application and middleware for QoS notification when an event occurs in the network and the application should adapt to this modification.

In contrast, our approach uses a SIP framework to manage sessions in the service plane. This framework provides interoperability and network QoS negotiation with the help of COPS-DRA Bandwidth Broker for adaptability and robustness. Our solution therefore requires no modifications to applications.

Network QoS management: Most related work on DRE system QoS support focuses on solving QoS-constrained routing problems separately from the middleware QoS policies. There have been several efforts, however, to bring the SIP protocol into the existing IP architecture. Current research includes adaptive and reflective middleware, and middleware for mobile and ubiquitous systems. For example, a programmable networking approach to provide QoS as component-based architecture is described in [3]. The authors in [13] designed a basic DDS/SIP Gateway to connect remote DDS domains, though they do not focus on the core network behavior and how resources are managed. The Common Open Policy Service (COPS) was extended in [11] to support SIP signaling over wide area networks, but COPS/SIP has not yet been integrated with DDS.

Despite the promise held by the COPS extensions, it does not account for the application needs of pub/sub DRE systems. Moreover, there is no straightforward approach to integrate SIP/SDP with DDS. Similarly, [20] is a mobility service proxy that deals with media session management for DDS mobile nodes.

Consequently, to bridge the gap between DDS and SIP/SDP so that the desired QoS policies are supported both by the end-system middleware and the network, we defined a new SIP Signaling Class of Service (S-CoS) for transferring signaling messages. Our approach does not underutilize the resources because it reserves only the required resources, which performs packet classification, policing, and traffic shaping in the edge router. For RT CoS and NRT CoS, our approach does not use all the resources since 80% of them are used by Best Effort traffic.

Table 7 compares the related work presented in this Section with our approach. R1 refers to the QoS management in content-based publish subscribe middleware and R2 refers to the Network QoS broker in middleware. The adaptive and reflective approaches in R1 and R2 work well today when they receive all the resources required. Other research has shown, however, that over provisioning resources has several drawbacks in scalability and can fail completely under the slightest anomaly [19]. Our approach therefore provides an adaptive framework that enables the end-to-end QoS provisioning over multi-domains independently of the underlying transports.

Table 7. Comparing the Various Approaches

Features	R1	R2	Our approach
Content-based pub/sub middleware	*		*
QoS specification	*	*	*
Resource allocation		*	*
QoS assurance			*

5 Concluding Remarks

Despite the powerful QoS mechanisms provided by pub/sub middleware, such as OMG's *Data Distribution Service* (DDS), these technologies are currently confined to end-systems, so they can only control local resources in DRE systems. To support end-to-end QoS in clouds, therefore, pub/sub DRE systems require an approach that enables resource reservation over wide area networks, yet is transparent to application logic. This paper addresses these requirements and describes a solution that seamlessly integrates DDS with SIP/SDP over clouds as follows:

- At the service plane, DDS applications use SIP signaling messages that allow senders to contact receivers to obtain their IP addresses and to agree the media description and "qos-dds" attributes.
- At the control plane, the network QoS provisioning mechanism provided by the COPS-DRA entities encodes application QoS requirements embedded within the SDP messages supplied to the network elements. The COPS protocol is chosen as common signaling mechanism to enforce the policy control mechanisms (*e.g.*, QoS negotiation, coordinate the data path and signaling path management, performs resource reservation, etc.).

We use DDS latency, transport priority, and bandwidth QoS settings to map them to signaling class of service (S-CoS) that allows DDS applications to control the data delivery in the access and core network for resource provisioning. To help preserve compatibility with existing DDS applications, we impose minimum requirements on the set of QoS policies that are compatible with most deployed DRE systems. Our future work will focus on extending the set of supported DDS QoS policies using the *Next Steps In Signaling* (NSIS) protocol and extensive empirical measurements to validate our approach.

References

1. Birman, K., Chockler, G., van Renesse, R.: Toward a Cloud Computing Research Agenda. SIGACT News 40(2), 68–80 (2009)
2. Rosenberg, J.: Integration of Resource Management and Session Initiation Protocol (SIP). In: Camarillo, G., Marshall, W. (eds.) RFC 3312 (October 2002)
3. Capra, L., Emmerich, W., Mascolo, C.: Reflective Middleware Solutions for Context-Aware Applications. In: Matsuoka, S. (ed.) Reflection 2001. LNCS, vol. 2192, pp. 126–133. Springer, Heidelberg (2001)
4. Chen, M., Hu, S., Muthusamy, V., Jacobsen, H.A.: Congestion Avoidance with Selective Filter Aggregation in Content-Based Routing Networks. Middleware Systems Research Group (November 2010)
5. Chi, Z., Sadjadi, M., Weixiang, S., Raju, R., Yi, D.: A user-centric network communication broker for multimedia collaborative computing. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2006, November 7-20, pp. 1–5 (2006)
6. JAIN-SIP-PRESENCE-PROXY, `snad.ncsl.nist.gov/proj/iptel/nist-sip-downloads.html`
7. NIST-SIP, `snad.ncsl.nist.gov/proj/iptel/`
8. Cho, E.-H., Shin, K.-S., Yoo, S.-J.: SIP-based Qos support architecture and session management in a combined IntServ and DiffServ networks. Journal of Computer Communications 29(15) (September 2006)
9. Data Distribution Service Interoperability Wire-Protocol Specification. DDSI v2.1, `www.omg.org/spec/DDSI/2.1/`
10. Durham, D., et al. (eds.): The COPS (Common Open Policy Service) Protocol Status of, RFC2748 (January 2000)
11. Gross, G., et al.: COPS Usage for SIP, draft-gross-cops-sip-01.txt, IETF Draft
12. Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol, RFC 4566 (July 2006)

13. López, J.M., et al.: DDS/SIP Interworking: A DDS-SIP Gateway. In: OMG Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems, May 24-26. Westin Arlington Gateway, Arlington (2010)
14. OMG-DDS, Data Distribution Service for Real-Time Systems Specification. DDSv1.2, www.omg.org/spec/DDS/1.2/
15. PADRES, padres.msrg.toronto.edu/Padres/WebHome
16. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol, RFC 3261 (June 2002)
17. Rosenberg, J., Schulzrinne, H.: An Offer/Answer Model with Session Description Protocol (SDP), RFC 3264 (June 2002)
18. Schmidt, D.C., et al.: Middleware R&D Challenges for Distributed Real-time and Embedded Systems. ACM SIGBED 1(1) (April 2004)
19. Huang, Y., Guerin, R.: Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger? In: Proceedings of the 13TH IEEE International Conference on Network Protocols, pp. 225–235 (2005)
20. Kwon, K.-J., Park, C.-B., Choi, H.: A Proxy-based Approach for Mobility Support in the DDS System. In: 6th IEEE International Conference on Industrial Informatics, INDIN 2008 (2008)
21. Teodora, G., et al.: A Session Initiation Protocol based Middleware for Multi-Application Management. In: IEEE International Conference on Communications - ICC, Multimedia Communications & Home Services Symposium, Glassgow, UK (June 24-27, 2007)

Continuous Access to Cloud Event Services with Event Pipe Queries

Qiming Chen and Meichun Hsu

HP Labs
Palo Alto, California, USA
Hewlett Packard Co.
{qiming.chen,meichun.hsu}@hp.com

Abstract. When cloud services become popular, how to consume a cloud service efficiently by an enterprise application, as the client of the cloud service either on a device or on the application tier of the enterprise software stack, is an important issue. Focusing on the consumption of the real-time events service, in this work we extend the Data Access Object (DAO) pattern of enterprise applications for on-demand access and analysis of real-time events.

We introduce the notion of *Operational Event Pipe* for caching the most recent events delivered by an event service, and the on-demand data analysis pattern based on this notion. We implemented the operational event pipe as a special kind of continuous query referred to as *Event Pipe Query* (EPQ). An EPQ is a long-standing SQL query with User Defined Functions (UDFs) that provides a pipe for the stream data to be buffered and to flow continuously in the boundary of a sliding window; when not requested, the EPQ just maintains and updates the buffer but returns noting, once requested, it returns the query processing results on the selected part of the sliding window buffer, under the *request-and-rewind* mechanism. Integrating event buffering and analysis in a single continuous query leverages the SQL expressive power and the query engine's data processing capability, and reduces the data movement overhead.

By extending the PostgreSQL query engine, we implement this operation pattern as the Continuous Data Access Object (CDAO) – an extension to the J2EE DAO. While DAO provides static data access interfaces, CDAO adds dynamic event processing interfaces with one or more EPQs.

1 Introduction

Due to the growing data volume and the pressing need for low latency, event service-has become a new kind of cloud data services [8]. Commonly, the event server (e.g. NYSE, Yahoo Finance, Twitter, GPS servers) sends events to its subscribers on the fly, as well as stores them in a data warehouse with certain latency. A client application retrieves real-time events or stored data when needed.

When cloud services become popular, how to consume these services efficiently by an enterprise or consumer application, as the client of the cloud services, is an important issue. In this work we focus on the consumption of the real-time event service, and extend the Data Access Object (DAO) pattern of data access applications for integrating the continuous caching and the on-demand analysis of real-time events.

As shown in Fig. 1, conventionally a data-intensive application interacts with the shared data sources through a Data Access Object (DAO) [11-13], a system component that deals with database connections through JDBC, maintains the prepared (compiled) SQL queries required by the application, launches these queries on-demand, interprets the results and delivers them to the application. The use of DAO allows all the data accesses to be made through a single entry efficiently, and allows the application and the database access to be developed and optimized independently. A DAO may be initially generated from specifications by the J2EE container.

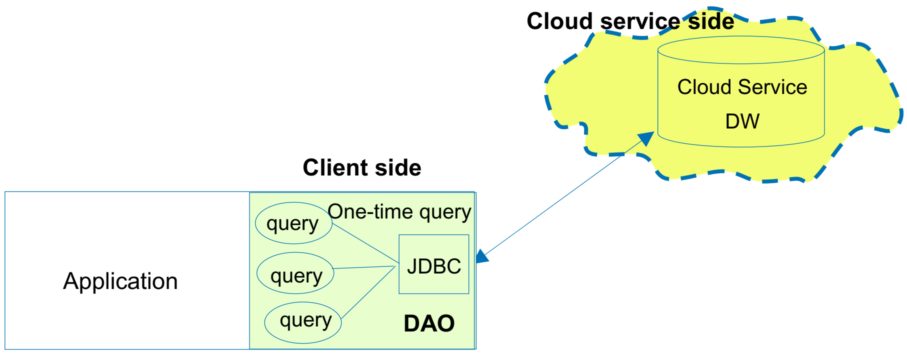


Fig. 1. Data Access Object (DAO) interfaces an enterprise application and databases

In this work we propose an extension to the J2EE DAO design pattern for consuming cloud event services efficiently by the cloud clients.

1.1 Support Continuous Data Access Pattern

We introduce the Continuous Data Access Object (CDAO) for accessing the real-time events and the warehoused events provided by cloud data services. Since the most recent events, such as the current readings of sensors, the trading prices of a stock in the last 10 minutes, the traffic volume in the past 5 minutes, etc, are frequently accessed by the client applications, we focus on the efficient event access pattern for using the cloud event service.

We introduce the notion of *Operational Event Pipes*, for buffering and analyzing the most recent events falling in the specified sliding time windows (Fig. 2).

The traditional DAO functionality is retained in the CDAO. An information request may be fulfilled by merging the event access result and the data warehouse access result.

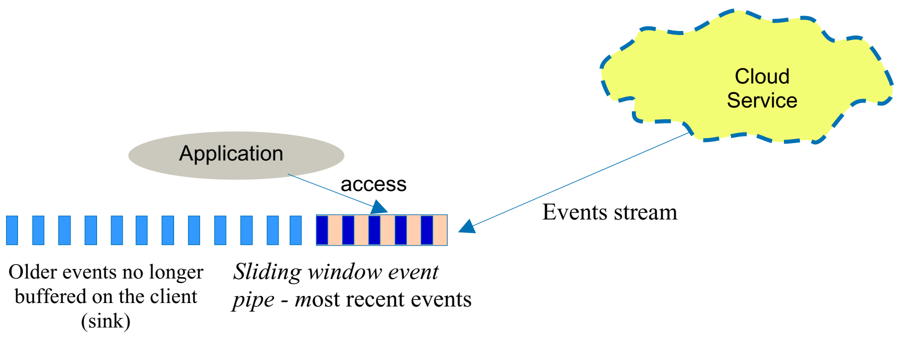

Cloud Service
Application
access
Events stream
Older events no longer buffered on the client (sink)
*Sliding window event pipe - m*ost recent events

Fig. 2. Event pipe for holding the most recent events falling in a sliding window boundary

1.2 Prior Art

Compared with the efforts for supporting real-time applications by ODS (Operational Data Store), we focus on continuous stream analytics [1,2,8-10].

Compared with the current generation of Complex Event Processing (CEP) systems and the Data Stream Management System (DSMS) [1,2,9], we focus on leveraging the SQL's expressive power and extending an existing query engines without requiring a separate data management technology. We reported our experience in using the query engine for stream processing as a server-side solution [5] but in this work our focus is placed on providing the operational event pipe as a client-site solution.

In the specific context of the Java programming language, DAO[11-13] as a design concept can be implemented in a number of ways. Commercial products like TopLink are available based on Object-relational mapping. Popular open source ORM products include Hibernate, pureQuery, iBATIS and Apache OpenJPA. We adopt DAO - the relatively simple and rigorous separation between two important parts of an application which can and should know almost nothing of each other, and which can be expected to evolve independently. However, beyond DAO, CDAO supports dynamic event access.

Finally, as described below, the unique feature of our approach lies in the integration of event caching and stream analytics, which is also in line with the principle of combining the data analytics layer with the data buffering layer for fast data access and reduced data move [3,4,7].

1.3 Our Solution

Motivated by integrated real-time event caching and analysis on the client application, in this work we propose a special kind of continuous query, referred to as *Event Pipe Query* (EPQ). An EPQ is capable of (a) piping event streams, (b) maintaining sliding window containers for the latest events, and (c) delivering sliding-window oriented analytical query results on-demand. With these capabilities, the introduction of EPQ allows us to integrate event caching and on-demand analysis within a single continuous SQL query.

A CDAO contains multiple EPQs as *pre-prepared* stream access queries.A capability is added for merging the query results on the piped events retrieved from the cloud event service, together with the required historical events retrieved from the cloud data warehousing service (Fig. 3). Like a DAO, a CDAO is private to a client application.

Note this solution is not about cloud service provisioning, but cloud service consumption, which we believe is an equally important issue.

An EPQ is a long-standing SQL query that updates the sliding window-based event pipe continuously; it generates query result once requested, but returns nothing if not requested. An EPQ has three basic components.

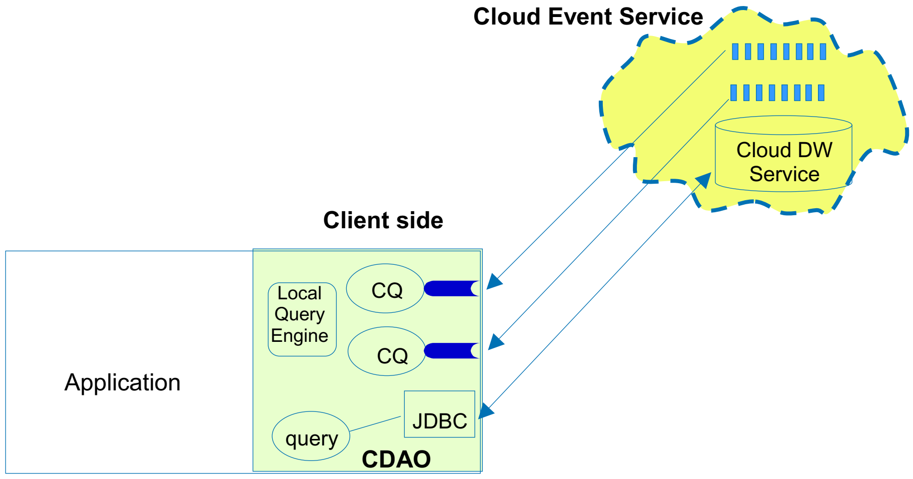


Fig. 3. CDAO for accessing event stream and data warehouse

- Streaming Capture Function (SCF) that reads the events from the subscribed event source, as well as receives information request (*info-req*) from the application to trigger the EPQ to deliver the designated window query results.
- Sliding Window Functions (SWF) that buffers the most recent events falling in a sliding window and continuously updates the buffer. Upon receipt of an *info-req* (passed to it from the SCF), this SWF returns the selected tuples from its buffer, as the local data source to the upstream query; otherwise it keeps updating the buffer but returns nothing.
- Stream Window Query (SWQ) that applies to the result set of the above SWF; it generates query result on the event pipe once requested.

We propose the *request-and-rewind* mechanism to support data supply punctuation, on-demand query evaluation and continuous query instance. Under this approach, the SCF, upon receipt of the *info-req*, punctuates the input data and signals the query engine to *terminate* and commit the current query execution, and then to *rewind* the query instance for serving the next request. This ensures the continuity of the EPQ instance without being shutting down, thus allows the sliding window content to be maintained continuously regardless of the query runs cycle by cycle.

The proposed EPQ approach leverages the SQL's expressive power and the query engine's data processing capability, and provides a tight integration of querying and data buffering which reduces the data movement overhead.

We have extended the open-sourced PostgreSQL query engine to support the proposed EPQ. We generalized the table UDF data buffering mechanism for supporting SCF and SWF, and extended the query engine to handle *request-and-rewind*. Our experience shows the potential of the EPQ approach in integrating real-time data buffering and querying for efficient and low-latency event analytics.

In the rest of this paper we will refine the proposed approach step by step. Section 2 gives an overview to the Continuous Data Access Object (CDAO) pattern; Section 3 describes the special kind of continuous query, EPQ, as the structure for CDAO; Section 4 focuses on the necessary extension of the query engine and the UDF framework for supporting EPQ; Section 5 illustrates with an example and some experimental results; Section 6 concludes the paper.

2 Data Access Pattern of CDAO

While a DAO provides static data access interfaces, a CDAO adds dynamic event processing interfaces for a complex enterprise application, e.g. a J2EE application. It buffers the most recent events on the fly and analyzes these events on-demand.

2.1 Operational Event Pipe

A CDAO is provided with one or more event pipes as the event flow channels. Each event pipe buffers the most recent events falling in a sliding window boundary, and serves as the data container of on-demand event analytics. The events are subscribed from the cloud event service.

The operational event pipes are implemented in terms of special kind of continuous queries, Event Pipe Queries (EPQs). An EPQ integrates event buffering and analysis in a single query.

The events held in an EPQ is maintained and updated on the fly. Without being requested by the application, it does not return any actual result; upon receipt of an *info-req* from the application, the query operations are performed on the buffered data to return a result.

2.2 Combination of Stream and Static Data Access

As mentioned above, CDAO is an extension to DAO where the event streams can be accessed using EPQs, and the stored historical events can be accessed in the regular way as supported by the conventional DAO facility.

Let us consider the following scenario: a CDAO represents a client of a cloud data service; it subscribes to the event service for providing the latest events to the application on-demand; it also subscribes to the data warehouse service where the past events are stored.

For low latency data access, the application's information request is fulfilled using the data buffered in the event pipe as the first priority. Since only the most recent events are kept in the event pipe, older data, if required, are still retrieved from the data warehouse. In this case the CDAO must be aware the boundary of the data available in the data warehouse and in the event pipe. Let us consider the following time marks (Fig 4):

- The *Water Level* (WL), t_{WL}, of the stored events in the database table is the timestamp of the latest events that are available for accessing.
- The *High Water Mark* (HWM), t_{HWR}, of the event pipe is the timestamp of the latest event in the event pipe.
- The *Low Water Mark* (LWM), t_{LWR}, of the event pipe is the timestamp of the earliest event in the event pipe.
- We ensure $t_{HWR} > t_{WL} > t_{LWM}$.

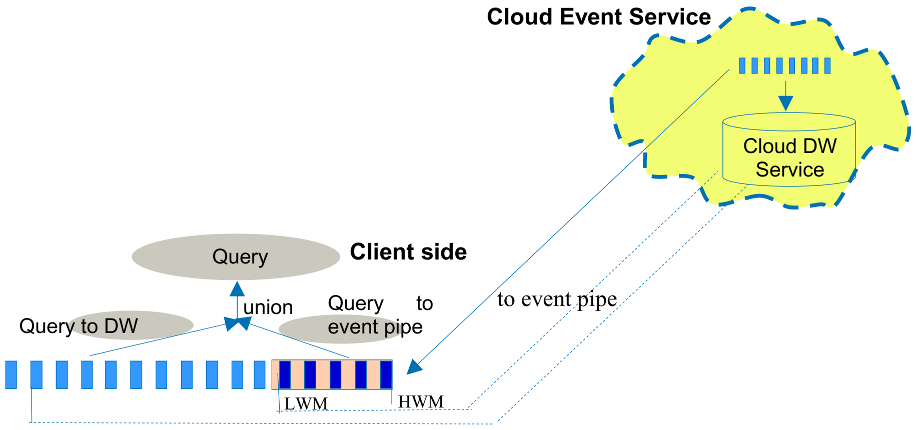


Fig. 4. Integrating stream access and data warehouse access

Then given the current timestamp t,

- The events with timestamps later than t_{LWR} can be accessed from the event pipe; in the range those earlier than t_{WL} can be accessed from the data warehouse; otherwise they are accessible from either stores. ;
- A query for accessing events in a given time range may be rewritten into two queries, one for accessing from the pipe and the other for accessing from the data warehouse; the results will be merged before returning to the caller.
- As mentioned above, the content of the event pipe is continuously inserted into the data warehouse. Whenever possible the event pipe is chosen for low latency When no new event shows up for a while, the whole event pipe may become "under water".

2.3 On-Demand Query Evaluation

An enterprise application with a CDAO may have multiple modules which retrieve and analyze event data in multiple ways, and the CDAO may be provided with multiple parameterized queries for retrieving and analyzing the warehoused events, as well as multiple EPQs for retrieving and analyzing the piped, most recent events. A special EPQ, called *ID-EPQ* is provided that simply returns the events in the given range on-demand, with a dummy query operation (i.e. just SELECT *).

The application makes a request through passing the CDAO a "*query object*" that contains the query ID, the time range of the queries events, and other parameters. Based on the *query object* the CDAO determines which query and EPQ to use. There exist three EPQ use cases.

- Only the events kept in the EPQ's event pipe are requested: This is the primary use of the EPQ. In this case the CDAO sends an *info-req* to the EPQ as a special event (not a parameter since EPQ is a continuous query), to trigger the EPQ to deliver event analysis result on-demand.
- Only the past events stored in the data warehouse are requested: In this case the CDAO launches the corresponding query with the actual parameters extracted from the *query object*.
- The requested events are kept in both the event table and the event pipe: In this case the CDAO launches a query and triggers an EPQ, and conceptually union the two results. Composing queries and merging results are *rule based* (e.g. union SUM and COUNT for deriving AVG). If the CDAO cannot ensure the correctness of result union (e.g. the query involves a UDF) the ID-EPQ will be used to merge the data at the source layer rather than at the query result layer.

It is worth noting that we use the query engine as the "streaming executor" to support EPQ, which is orthogonal to data warehousing.

2.4 Cloud Event Service Consumer

Analogous to a DAO, a CDAO is a component of an enterprise application, rather than a public data service such as the stock quote service provided by Yahoo; it is private to the supported enterprise application or application suite, although the application may have multiple building blocks.

- A CDAO, on the client side, handles the most recent events subscribed from event services, but not responsible for the overall event management such as event warehousing, fault-tolerance, etc.
- A CDAO may hold multiple EPQs on the same event stream but with different query constructs expressing different event analysis functionalities, and with different sliding window coverage.
- An EPQ is provided with the option to continuously and automatically output events one by one, or by time intervals, without explicitly receiving requests from applications. The stream data "chunking criterion" is specified as a parameter of the Streaming Capture Function SCF. The details are described in the next section.

3 Event Pipe Query

Introducing EPQ aims at integrating real-time event buffering and on-demand analysis using continuous query model.

3.1 EPQ Constructs

An EPQ typically has three components, for catching the events, maintaining the most recent events falling in a sliding window time range, and querying the content of the sliding window buffer on-demand. When not requested, the EPQ acts as an event flow pipe. When requested, it returns the query result on the event pipe.

For example, the EPQ for caching and delivering the most recent stock market trades may be expressed as below.

```
SELECT (s).symbol, (s).order_type, SUM((s).volume), MAX((s).price), MIN((s).price) FROM
 (SELECT sliding_window_swf(1800,t.symbol,t.price,t.volume,t.order_type,t.time,t.low,t.high) as s
   FROM stream_reader_scf ("daily_stock_exchange_stream") t
) sw
GROUP BY (s).symbol, (s).order_type;
```

The stream source, “daily_stock_exchange_stream”, has attributes

```
[time, symbol, price, volume, order_type]
```

where time is the timestamp in second, order_type can be “buy” or “sell”.

Below we describe the three components of the above EPQ (Fig. 5).

- *stream_reader_scf* () is a Streaming Capture Function (SCF) for (a) getting input data, and (b) receiving *info-req* from the application. It is a table function (i.e. a function returns a set of tuples) with result set schema

  ```
  [symbol, price, time, volume, order_type, low, high]
  ```

  where *low*, *high* mark the range of the sliding window content to be queried; when not requested, both *low* and *high* are NULL (0); when *high* represents the latest time, it is set to 0. For instance, <36000, 0> stands for the time period from 10AM to now in a day; <36000, 36010> for the range from the 10:00AM to 10:10AM <0, 0> means not requested.
- *sliding_window_swf* () is a Sliding Window Functions (SWF) for maintaining the most recent stream elements falling in the sliding window boundary of 1800 seconds (30 minutes); it receives data tuple by tuple from the SCF, *stream_reader_scf* (), for maintaining and updating the sliding window state. This function is also a table function. When not requested, i.e. the values of the attributes *low* and *high* of the input tuple is NULL (0), it returns NULL; otherwise it returns tuples in the requested range with schema

  ```
  [symbol, price, volume, order_type, time]
  ```

- A Sliding Window Query (SWQ) that applies to the tuples returned from the SWF, if any. In the above EPQ, the SWQ is applied to the result of the sub-query with alias sw (i.e. the result of the SWF), as

```
SELECT (s).symbol, (s).order_type, SUM((s).volume), MAX((s).price), MIN((s).price)  FROM sw
GROUP BY (s).symbol, (s).order_type;
```

The SWQ has no effect with NULL input, and is evaluated based on the query logic otherwise.

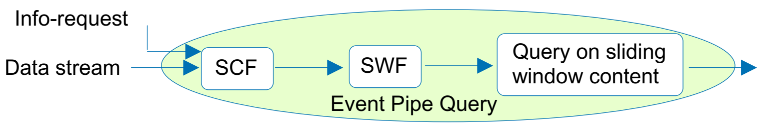


Fig. 5. Event Pipe Query (EPQ) constructs

The notions of SCF, SWF and SWQ can be described more generally as follows.

Streaming Capture Function (SCF). An EPQ is running continuously with stream data as its input; thus the query's access method is to receive the stream element on the fly, e.g. read from a socket or a log-file. At the bottom of the query tree, the regular table-scan is replaced by function-scan with which is a kind of table function. An SCF has two input channels (not parameters), one for reading event data, turning them to tuples and feed to the SWF; the other for receiving the *info-req* from the J2EE application to trigger the EPQ to deliver the designated window query results. The triggering is actually made by sending the "end-of-data" signal to the query engine to have the query engine "complete" the query evaluation on the sliding window content, and then rewind for other request in the next "execution cycle".

The above stream punctuation, or data chunking, can also be made continuous and automatic without explicit requests from applications, for processing events one by one or by time intervals (e.g. minute by minute). The "chunking criterion" is specified as a parameter of the SCF. In the per-event (per-tuple) processing case, dividing query execution into cycles is unnecessary. This mechanism is similar to the one we reported in [5].

Sliding Window Function (SWF). A SWF is a kind of table function for buffering and continuously maintaining the latest events, or the data derived from them, falling in a *sliding window* boundary. In an EPQ, the SWF is the actual data source of the SWQ operation. It acts in the following way.

- It continuously maintains and updates the sliding window buffer by appending new elements and removing the older ones.
- The *info-req* received by the SCF is embedded into the next tuple feed in the SWF.

- When not requested it simply returns NULL resulting in no data fed into SWQ and thus no result returned from the EPQ. The whole EPQ just provides a pipe for the events to pass.
- Once requested, the current sliding window content will become the local data source of the SWQ; as mentioned above, the current query execution will be completed with result returned. However, since the query instance is just rewound without shutting down, the buffered events will not be lost.

Sliding Window Query (SWQ). While the SCF is used for data capturing, SWF for data buffering, SWQ is used for data analysis. It is a continuous query operation defined on the results of the sub-query, i.e. the result returned from the SWF. As indicated above, the SWQ is evaluated continuously as far as the EPQ execution is not terminated. Without *info-req* the SWQ behaves like a dummy function since there is no input from SWF. When an *info-req* from the application is passed through the SCF to the SWF, the SWF returns the selected events to feed into the SWQ.

3.2 EPQ Execution Pattern

The query results of an EPQ are generated on-demand and only on the events kept in the event pipe. EPQ must satisfy several requirements: first, after receiving the *info-req* the incoming event stream to the buffer must be cut to allow the query to apply to the existing content of the buffer. Next, the query execution must be completed and terminated in a regular way, in order to return query result; finally, the query instance must be kept alive for holding the sliding window state continuously across requests, but rewound for processing the next request.

Request-Rewind. To meet the above requirements, we propose the EPQ execution model, referred to as the *request-rewind model.* Based on this model, the EPQ does not return any result without being requested, but upon receipt of an *info-req* by the SCF, that *info-req* is embedded into the next event tuple passed to the SWF to instruct the SWF to return the buffered events for the SWQ to process. The SCF also signifies the query engine with “end-of-data” to allow the current cycle of query evaluation to commit and to terminate in the regular way, thus making the query results accessible. After that, the EPQ instance rewinds but is kept alive. During this process, the input events are continuously received and buffered by the SCF; after this process, these events are piped into SWF.

This mechanism allows the EPQ to apply the query to the event pipe on-demand, rewind for servingthe next request, and keep a live instance for retaining the event pipe content.

3.3 The Conceptual Model of EPQ

Conceptually, an EPQ is an object with <S, Q, f_e, f_q> where

- S is the state of a sliding window buffer object which we call event pipe;
- Q is a query object as described by the SWQ construct above;

- f_e : <S, e> → S is a core function for maintaining the sliding window buffer; upon receipt of an event e, it updates the state S;
- f_q : <S, r> → $Q(S)$ is a core function that, upon receipt of a request r, applies Q to S, i.e. applies the SWQ to the content held in the sliding window buffer.

Note that S and Q are themselves objects with their own constructs (e.g. S is a subclass of a queue object with a time boundary).

Besides the above explicit semantics, it is implied that

- f_e : <S, e> → S has no side effect to the SWQ query operation (if not requested). This expresses the semantics that without a request, the EPQ serves as the event pipe to continuously buffer the events that flow by.
- f_q : <S, r> → $Q(S)$ has no side effect to the state of the event pipe. This expresses the semantics that once requested, a query result on the buffered events is generated but the buffer is continuously retained without interruption. This semantics is guaranteed by implementing EPQ as a long-standing, continuous query, and by the request-rewind mechanism for keeping the query instance alive all the time.

The proposed EPQ approach supports all the above semantic features using a single continuous query which leverages the SQL expressive power and the query engines data processing capability. It provides a tight integration of querying and data buffering which reduces the data movement overhead in serving continous query results.

To gain the above benefits we have proposed solutions for problems such as how to allow a continuous query to generate results on a particular set of data, and how to make query evaluation on-demand. The proposed request-rewind approach provides a simple, effective and efficient solution.

4 Supporting Continuous EPQ

To support the proposed EPQ, technically it is necessary to extend the query engine and the UDF framework for handling SCF, SWF and the on-demand query execution.

4.1 Replace Table Scan by Stream Capture Function Execution

We start with providing unbounded relation data to feed queries continuously. The first step is to replace the database table, which contains a set of tuples on disk, by the table function Stream Capture Function (SCF) that returns a sequence of tuples without first storing them on disk. An SCF can listen or read a data/event sequence and generate stream elements. An *info-req* from the application, as a particular event, instructs the SCF to signal end-of-data to the query engine to commit and terminate the current query execution and deliver the query result on-demand.

Conventionally, a table function first materializes its result set and then sends its content to the upstream query tuple by tuple. For streaming purpose we eliminate the materialization phase of SCF.

The function scan is supported at two levels, the SCF level and the query executor level. A data structure containing the function call information bridges these two levels; it is initiated by the query executor and passed to the SCF for exchanging function invocation related information. We use this mechanism for minimizing the code change while maximizing the extensibility of the query engine.

4.2 Extend UDF Data Buffering Mechanism to Support Sliding Window Function

As described earlier, a Sliding Window Function (SWF) buffers the events across multiple input tuples, and returns a set of tuples on demand.

The current generation of SQL systems offers scalar, aggregate and table functions as UDFs. The input of a scalar or table function is bound to the attribute values of a single tuple, where an aggregate function is actually evaluated incrementally tuple-by-tuple. A scalar or aggregate function can maintain a data buffer across multiple input tuple, but since it cannot return a set out of a single input, it is not suitable for our purpose; instead we have to use the table function with set return. However, a table function cannot buffer data across multiple *input tuples*, and therefore we need to extend it to satisfy the requirements for SWF.

The extension primarily focuses on the UDF buffer management. A SQL query is evaluated tuple by tuple on its input, and therefore a function, including a UDF, is called multiple times by the host query. Accordingly, a UDF is optionally coded with three code sections: the init-section, the regular-section and the final-section. The data buffer and the application state are initiated in the first call of the function, which can be retained across multiple calls; the regular-section deals with the application logic; the final-section is executed last for cleanup purpose. We refer to this as the *multi-call cycle*.

The multi-call cycles of a scalar/aggregate function and of a table function are different – the former spans all the input tuples; the latter is only associated with the multiple returns out of one input tuple, which means a table function lacks the capability of buffering data across multiple input tuples. We have extended the query engine and UDF framework as the technical backbone to allow a table UDF to retain a data buffer across input tuples.

The "top-level" memory context of a table function is local to one input tuple only. To buffer multiple input data, an additional level of memory context, or buffer type, must be provided. With such an extension, three types of states can be maintained during the query processing:

- the state relating to the function invocations throughout the query, i.e. wrt all the input tuples;
- the state relating to processing one input tuple which may involve multiple calls to deliver multiple returns; and
- the state relating to the handling of one return tuple.

We have extended the query executor to support all these memory contexts. The extension starts from the function node in the query tree and then goes to the system handles for managing function invocation. New data structures and switching mechanisms are added for extending buffers. Accordingly new APIs for creating data buffer for each of these three contexts are implemented. In general, the buffers of a UDF at all levels are linked to the system handle for function invocation, and accessible through system APIs.

Further, since the EPQ instance is always alive and never shut down, the SWF buffer can be retained continuously across the EPQ execution cycles.

4.3 Extend Query Engine to Support Request-and-Rewind

An EPQ buffers the most recent events falling in a sliding window, and acts as a dataflow pipe no matter the contents of the sliding windows are queried or not. The query results are generated on demand**.**

Request. When a client requests the real-time events or event processing results, it sends an *info-req* to the SCF, which has the following effects.

- The request is embedded in the next event fed to the SWF, to instruct it to return the requested events from the event pipe, to be evaluated by the SWQ.
- Then the SCF signals *end-of-data* to the query engine through setting a flag on the function call handle, resulting in the completion and termination of the current query execution cycle in the regular way, and allowing the query results accessible.

Rewind. Upon termination of an SWQ execution cycle, the query engine does not shut down the query instance but *rewinds* it for continuously piping vents and serving requests. During termination-rewind period, the upcoming events, if any, are buffered by the SCF to be processed in the next cycle.

Rewinding a query is a top-down process along the query plan instance tree, with specific treatment on each node type. In general, the intermediate results of the standard SQL operators (associated with the current chunk of data) are discarded but the application context kept in the SWF is retained. The query will not be re-parsed, re-planned or re-initiated.

Note that rewinding the query plan instance aims to pipe new events and to serve new request, rather than re-deliver the current query result; therefore it is different from “rewinding a query cursor” for re-delivering the current result set from the beginning.

The proposed *request-rewind* approach has the ability to keep the continuity of the query instance over the entire stream, and therefore to integrate sliding window based stream buffering and querying for low latency analysis.

5 Examples and Experiments

5.1 A Data Retrieval Example

To illustrate how the streaming data and the warehoused data are queried in combination through CDAO, let us consider an application request for the summary of stock market trades, with parameters < Q_a, t_1, t_2> where Q_a is treated as an "*abstract query*" to be applied to the data in the time range of < t_1, t_2>.

[Abstract Query: Q_a]

```
SELECT (s).symbol, (s).order_type, SUM((s).volume), MAX((s).price), MIN((s).price)
FROM daily_stock_exchange
GROUP BY (s).symbol, (s).order_type;
```

where "daily_stock_exchange" is an "abstract data source" that can be specialized to the stored table, "daily_stock_exchange_table", and the event pipe, "daily_stock_exchange_stream". For simplicity, we count the timestamps only for the current day starting from 0 (second) at 0:00 AM. The event pipe keeps the transactions in the latest 30 minutes. We also assume the events loaded to the table are made accessible within 10 minutes; then the event table and the event type have overlapped coverage.

In this example, the request is given at 10:30 AM, for the stock market summary from 9:00 AM to the current time. Based on the above settings, the CDAO specializes the abstract query, Q_a, as follows.

- It launches a "table query" for retrieving the stored events with timestamps from 9:00AM to 10:15 AM; and
- It sends an info-req to the corresponding EPQ to trigger it to deliver the event processing result in the time range from 10:15AM to now.

The "table query" is listed below.

[Table Query: Q_{tb}]

```
SELECT (s).symbol, (s).order_type, SUM((s).volume), MAX((s).price), MIN((s).price)
FROM daily_stock_exchange_table
where time >= 32400        /* starting from 9:00AM */
and time < 36900           /* to 10:15AM */
GROUP BY (s).symbol, (s).order_type;
```

The EPQ, Q_{epq}, is the same as shown before in Section 3.1. Note that the time range (from 10:15 to now) is not a parameter of the query (since the query is continuously running); instead, it is read-in by the SCF, "*stream_reader_scf*()" as a special event.

[EPQ: Q_{epq}]

```
SELECT (s).symbol, (s).order_type, SUM((s).volume), MAX((s).price), MIN((s).price) FROM
 (SELECT sliding_window_swf(1800,t.symbol,t.price,t.volume,t.order_type,t.time,t.low,t.high) as s
    FROM stream_reader_scf ("daily_stock_exchange_stream") t
 ) sw
GROUP BY (s).symbol, (s).order_type;
```

5.2 EPQ Performance

The primary goal of introducing EPQ is to support low latency event analysis by means of in-memory data access and continuous querying, reducing the overhead of disk I/O as well as query parsing, planning, optimizing and launching. These advantages can be illustrated by our experiments.

We measured the response time of the EPQ, Q_{epq}, using a set of synthetic stock exchange data covering 3000 stocks. The buffer sizes of the EPQ event pipe, by the number of transactions, are set to 10K, 50K, 100K, 250K, 500K, 750K and 1M respectively. The experimental results are measured on a laptop, HP EliteBook 8350 with 2 x Intel Xeon E54102 2.33 Ghz CPUs and 4 GB RAM, running Windows Vista (x86_32) and PostgreSQL 9.0.3.

The impact of the sliding window buffer size on query response time is illustrated by Fig. 6, where the query operation (SWQ) is applied to the whole content of the buffer.

We also compared the data access time based on the EPQ mechanism and the store-first-query-later mechanism. Using EPQ, the stream elements are captured on the fly and buffered in the event pipe. Under the store-first-query-later mechanism, the data are first inserted in the data warehouse table and then queried. The sizes of the data sets are also set to 10K, 50K, 100K, 250K, 500K, 750K and 1M respectively. The comparison results are shown below and also illustrated in Fig. 7.

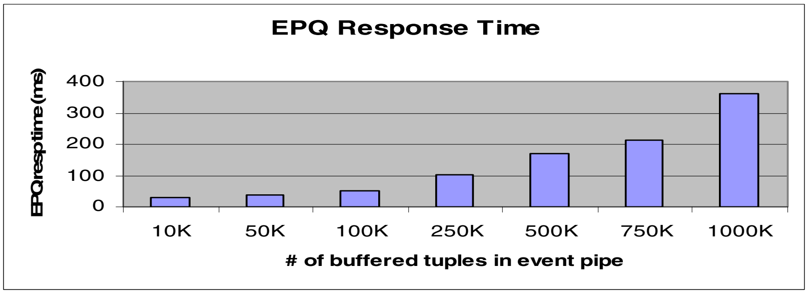


Fig. 6. EPQ response time

# buffered tuples	10K	50K	100K	250K	500K	750K	1M
EPQ resp time (sec)	0.031	0.040	0.052	0.101	0.169	0.212	0.361
Store-query time (sec)	0.119	0.371	0.892	2.853	4.748	7.713	8.993

It can be seen that the EPQ approach significantly outperforms the store-first-analyze-later approach. When the event pipe buffer size is set to 1M, querying EPQ is 25X faster than querying from the data warehouse.

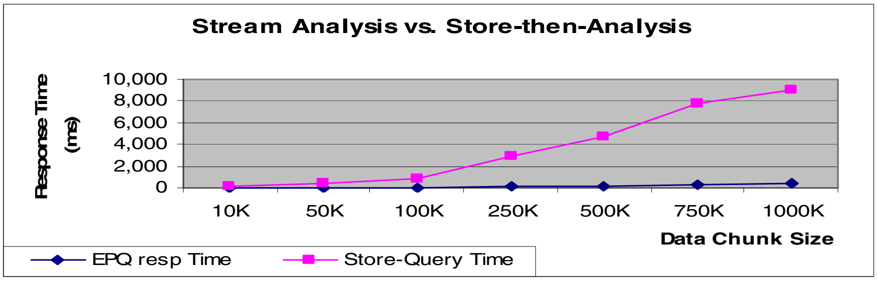


Fig. 7. Data access latency comparison: EPQ vs. store-then-query

6 Conclusions

In many enterprise and consumer applications, real-time events and historical data are used in combination. In this work we focus on how to efficiently consume cloud data services that offer both real-time streaming events and warehoused historical data.

We introduced the continuous data access object (CDAO) pattern as the extension to the data access object (DAO) pattern used in developing data access clients, for on-demand access and analysis of real-time events provided by cloud event service. We introduced the notion of *Operational Event Pipe* for keeping the most recent events in a sliding time window, and for analyzing the events in the pipe, as the basic construct for CDAO. We implemented the operational event pipe as a form of continuous query referred to as *Event Pipe Query* (EPQ) and developed the *request-and-rewind* mechanism for retrieving and analyzing the piped events on demand. An EPQ is a long-standing SQL query that updates the event pipe continuously, generates query result once requested, but returns nothing if not requested. We use the *request-and-rewind* mechanism to support the data supply punctuation, on-demand query evaluation and continuous query instance. Although the EPQ runs cycle by cycle, the query instance is never shut down, allowing the sliding window buffer to be maintained continuously.

As a client of cloud data service, a CDAO is private to one (or a suite) application, and it can contain multiple EPQs.

The proposed EPQ approach leverages the SQL's expressive power and the query engine's data processing capability, and provides a tight integration of querying and data buffering which reduces the data movement overhead between the server (the stream source and the data warehouse) and the client (the application that responds to data access calls on a device or from another application).

We have implemented CDAO using the open-sourced PostgreSQL query engine, and extended it to support the continuous data buffering of table UDFs and the request-and-rewind query execution model. Our experience shows its merit in efficient and low-latency real-time data buffering and retrieval.

References

[1] Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. VLDB Journal 2(15) (June 2006)

[2] Abadi, D.J., et al.: The Design of the Borealis Stream Processing Engine. In: CIDR (2005)

[3] Bryant, R.E.: Data-Intensive Supercomputing: The case for DISC, CMU-CS-07-128 (2007)

[4] Chaiken, Jenkins, B., Larson, P-Å., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In: VLDB 2008 (2008)

[5] Chen, Q., Hsu, M., Zeller, H.: Experience in Continuous analytics as a Service. In: EDBT 2011 (2011)

[6] Chen, Q., Hsu, M.: Cooperating SQL Dataflow Processes for In-DB Analytics. In: Proc. 17th International Conference on Cooperative Information Systems, CoopIS (2009)

[7] DeWitt, D.J., Paulson, E., Robinson, E., Naughton, J., Royalty, J., Shankar, S., Krioukov, A.: Clustera: An Integrated Computation And Data Management System. In: VLDB 2008 (2008)

[8] Franklin, M.J., et al.: Continuous Analytics: Rethinking Query Processing in a NetworkEffect World. In: CIDR 2009 (2009)

[9] Gedik, B., Andrade, H., Wu, K.-L., Yu, P.S., Doo, M.C.: SPADE: The System S Declarative Stream Processing Engine. In: ACM SIGMOD 2008 (2008)

[10] Liarou E., et al.: Exploiting the Power of Relational Databases for Efficient Stream Processing. In: EDBT 2009 (2009)

[11] Core J2EE Patterns - Data Access Objects. Sun Microsystems Inc. (2007), http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[12] IBM pureQuery: A high-performance data access platform, http://www-01.ibm.com/software/data/optim/purequery-platform/

[13] JingDAO, http://jingdao.sourceforge.net/

QoS-Enabled Distributed Mutual Exclusion in Public Clouds

James Edmondson, Doug Schmidt, and Aniruddha Gokhale

Dept. of EECS, Vanderbilt University, Nashville, TN 37212, USA
{james.r.edmondson,doug.schmidt,a.gokhale}@vanderbilt.edu

Abstract. Popular public cloud infrastructures tend to feature centralized, mutual exclusion models for distributed resources, such as file systems. The result of using such centralized solutions in the Google File System (GFS), for instance, reduces scalability, increases latency, creates a single point of failure, and tightly couples applications with the underlying services. In addition to these quality-of-service (QoS) and design problems, the GFS methodology does not support generic priority preference or pay-differentiated services for cloud applications, which public cloud providers may require under heavy loads.

This paper presents a distributed mutual exclusion algorithm called Prioritizable Adaptive Distributed Mutual Exclusion (PADME) that we designed to meet the need for differentiated services between applications for file systems and other shared resources in a public cloud. We analyze the fault tolerance and performance of PADME and show how it helps cloud infrastructure providers expose differentiated, reliable services that scale. Results of experiments with a prototype of PADME indicate that it supports service differentiation by providing priority preference to cloud applications, while also ensuring high throughput.

Keywords: mutual exclusion, public cloud, QoS, file systems.

1 Introduction

The Google File System (GFS) was designed to support the sustained file throughput capacities of the Google search engine [1,2,3]. GFS provides high throughput in a single cluster of thousands of computers, each servicing the Google search engine. Although the GFS scaled well to hundreds of terabytes and a few million files in append-mode (GFS does not support overwriting a file), other quality-of-service (QoS) properties (*e.g.*, latency, throughput of small files—which is common in many applications, and differentation amongst applications) were not the focus of its initial design.

Scalability problems with GFS began appearing when the centralized master server was forced to process tens of petabytes worth of data requests and appends [2]. As a short-term solution, Google engineers used a centralized master server to manage a *cell* of the overall cluster. Although this approach provided some fault tolerance against the single master failing, some failures still occurred, and throughput and scalability suffered [1].

As Google grew, so did its list of services and applications. Since GFS focused on throughput rather than latency and scalability, performance issues appeared with certain applications, such as GMail, Youtube, and Hadoop [1]. Google's temporary solution to overcome this problem was the BigTable application, which was layered atop GFS and packed small files into the large 64 MB file metadata that had been in place since their Internet crawler was first deployed [1,2].

For cloud applications (such as Youtube) that can be buffered, the latency of the GFS system has mostly been acceptable. For applications with file accesses and writes on the cloud, however, Google is looking into replacements for GFS that provide better QoS [1]. Deferring these changes can be costly for applications that have a GFS-like interface or these applications face mounting integration issues.

Figure 1 shows the scalability problems of a centralized server for reading and writing to a segregated file system. In this figure, the light-shaded lines represent the computational and bandwidth resources that are utilized and dark represents the wasted resources. According to Google network engineers, the major bottlenecks of the GFS system include the inability to provide native overwrite operations (GFS supports only append mode and applications have to work around this), the 64 MB metadata for even small files, and the centralized master controller that every request and write goes through in each cluster cell. The centralized master controller provides mutual exclusion for read-write operations, garbage collection, and replication and persistence. Even with extra computing resources, this controller reduces scalability and throughput, and significantly increases latency due to queuing [1].

In addition to the latency bottleneck, reduced overall throughput, and lack of fault tolerance, GFS's centralized architecture also treats all application requests equally. Applying this system to cloud providers with scarce resources and steady client application deployments means there is no built-in differentiation between client priorities according to their payscale or other factors.

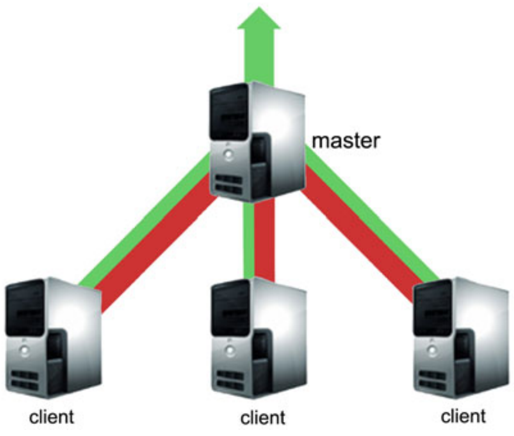


Fig. 1. GFS Centralized Master Controller

In general, a cloud file system should address the following challenges:

1. **Avoid centralized bottlenecks**, such as a master controller which restricts file throughput, and increases latency due to queuing and limited bandwidth through a single host.
2. **Handle failures of nodes and applications**, *e.g.*, GFS requires that master controllers be manually reset by an engineer when they fail [1].
3. **Support aggregated priorities**, *e.g.*, based on expected cost, payment, etc. of cloud applications.

This paper presents a distributed mutual exclusion algorithm called *Prioritizable Adaptive Distributed Mutual Exclusion* (PADME) that can be used to address these challenges in cloud file systems by decentralizing the mutual exclusion problem, as shown in Figure 2. As with Figure 1, the decentralized PADME algorithm still has a certain amount of overhead (shown in dark) due to the replicas and clients working together and coming to a consensus. In contrast, however, the overall capacity of the decentralized PADME approach scales with the number of participating nodes, rather than being limited by a single master controller.

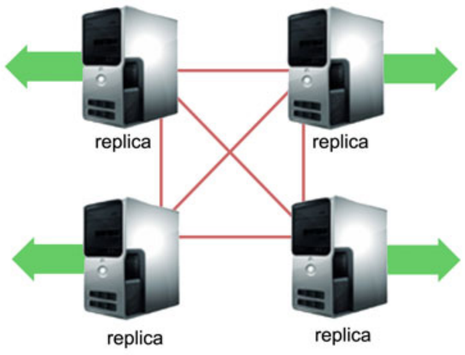


Fig. 2. Scalability of a Decentralized Methodology

The remainder of this paper is organized as follows: Section 2 describes the design and capabilities of of PADME algorithm; Section 3 evaluates results from experiments conducted on a simulated message-oriented prototype of the PADME algorithm over shared memory; Section 4 compares our work on PADME with related research; and Section 5 presents concluding remarks and lessons learned.

2 Distributed Mutual Exclusion in Public Clouds

This section presents an algorithm called *Prioritizable Adaptive Distributed Mutual Exclusion* (PADME) that we developed to meet the cloud file system challenges described in Section 1. The PADME algorithm performs two main operations:

1. It maintains a spanning tree of the participants (*e.g.*, applications, customers, or anything that wants access to the file system) in the network. The spanning tree is based upon the customer or application priorities and the root of the spanning tree will be the highest priority entity in the system. Depending on the network topology and customer demands this root can and will change during runtime operations.
2. It enforces mutual exclusion according to user-specified models and preferences for messaging behavior. The models supported by the PADME algorithm include priority differentiation and special privileges for intermediate nodes in the spanning tree (intermediate nodes are nodes between requesting nodes). Each model may be changed during runtime if required by the cloud provider, applications, or users.

Below we describe the PADME algorithm and show how it can be implemented efficiently in cloud middleware platforms or cloud applications, wherever the distributed mutual exclusion is appropriate.

2.1 Building the Logical Spanning Tree

PADME builds a logical spanning tree by informing a cloud participant (*e.g.*, an application that is using the cloud infrastructure to manage the file system) of its parent. Under ideal circumstances, all the spanning tree construction should be performed in the cloud infrastructure without affecting user applications.

A participant need not be informed of its children as they will eventually try to contact their parent, establishing connections on-demand. PADME uses this same mechanism to reorder the tree when optimizing certain high priority participants. Each participant is responsible for reconnecting to its parent through the cloud API, middleware, or however the cloud offers file system services.

To establish which parent to connect to, PADME's joining process multicasts its priority and waits for responses during a (configurable) timeout period. The closest priority above the joining process becomes the parent. The process can arbitrarily connect to a different parent later, in which case the parent will need to remove pending requests from the child. This step can be postponed until the file access permission returns to this node for the specific requester to eliminate the need for a parent to care about children moving to other parents. If the connection no longer exists the token is sent back up to higher priority processes in the logical spanning tree of cloud participants.

Cloud middleware developers could decide to let each cluster form its own priority-based spanning tree and connect roots of each tree to form the cloud file system mutual exclusion tree. The PADME algorithm presented in Section 2.2 will work with any spanning tree as long as connected nodes can communicate. Section 2.3 covers PADME's fault tolerance support. Figure 3 shows the construction of such a spanning tree out of priority information. In this case, the tree is balanced, though it need not be. During runtime, an application or user may add or remove a participant, rename participants, or conduct other such operations to organize the intended tree and dynamically respond to changes in request load or priority changes.

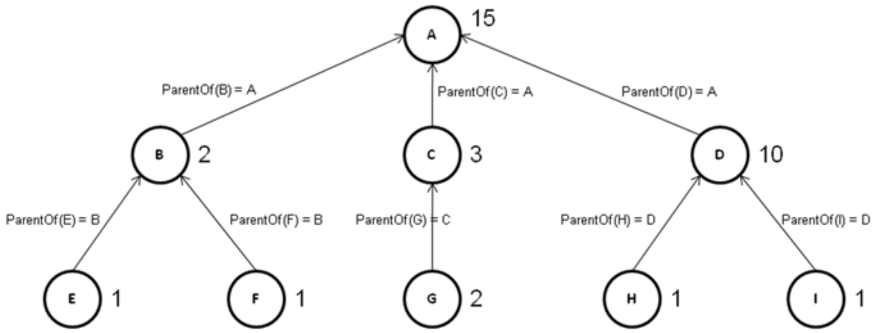


Fig. 3. Building a Balanced Spanning Tree

PADME's tree building phase requires updating affected participants with parent information (*i.e.*, informing them of the direction of the logical root of the tree). The messaging overhead of maintaining a logical spanning tree is not included in our algorithm details because cloud providers can simply build a spanning tree once manually if desired. For example, the cloud provider may know that the application deployments are static, or the provider is anticipating a specific scenario like spikes of file access during a major sporting event and wants to give preference to this activity during the championship match.

Even in statically assigned spanning trees, higher priority cloud participants should be pushed towards the root to give them preferential service. The logical token will be passed back to the root of the tree before continuing on to the next participant in our algorithm. Participants at higher levels of the tree experience lower message complexity, faster synchronization delay, better throughput, and even higher QoS for the target system and a preference for high priority customers, as shown in Section 2.3.

2.2 Models and Algorithm for Distributed Mutual Exclusion

Before accounting for faults, PADME requires just three types of messages: *Request, Reply,* and *Release.* A Request message is made by a participant that wants to acquire a shared resource, such as cloud file system access. A Request message traverses up the spanning tree from the participant node to the root via its parent and ancestor nodes. A Reply message is generated by the root after access to the resource is granted. The Reply message traverses from the root to the requesting participant node. The Release message traverses up the tree from the node that holds the shared resource towards the root once the node is ready to release the resource. The interaction of these messages is shown in Figure 4.

PADME supports four models (*Priority Model, Request Model, Reply Model,* and *Release model*) that describe the semantics of actions performed by any participant in the spanning tree that receives one of the three types of messages, as

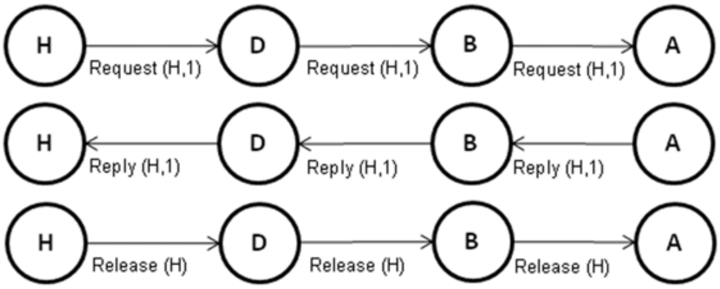


Fig. 4. PADME Messages for Mutual Exclusion

well as QoS differentiation that must be supported. The latter three models are named according to whether an intermediate participant can enter its own critical section, *i.e.*, exclusive access to the cloud's distributed file system resource, upon receipt of the message type. These models stem from our approach to distributed mutual exclusion and optimizations that allow shorter synchronization delay between critical section entries and improved QoS via user-specified requirements to middleware.

The configurations of the Request, Reply, and Release models may be changed at runtime to yield different QoS, including

- **Higher critical section throughput** – *i.e.*, the number of file accesses possible to the cloud's distributed file system,
- **Changes in fairness** – *e.g.*, going from preferring higher priority participants to giving everyone a chance at the critical section – a requirement of our motivating scenario in Section 1,
- **Fewer priority inversions** – *i.e.*, alleviating the situation where a low priority participant gets a critical section entry before a high priority participant, even though a critical section request from a higher priority participant exists, and
- **Lower average message complexity** – *i.e.*, fewer messages being required per critical section entry.

These four models are described below and each are integral components in the algorithm that may be tweaked to affect performance, usually at the cost of possible priority inversions.

Request Models. PADME provides two Request Models: *Forward* and *Replace*. The Forward Request Model requires a parent to immediately forward all requests to its own parent. The Replace Request Model requires a parent to maintain a priority queue of child requests, which should have the same Priority Model as the root participant. Under the Replace Request Model, a node only sends a Request to its parent if there are no Request messages in its priority queue, or if the new Request is of higher priority than the last one that was sent. The Replace Request Model is slightly harder to implement, but it results in messages only being sent when appropriate and may alleviate strain on the root node. It also will result in less message resends if a parent node fails.

Reply Models. PADME provides two Reply Models: *Forward* and *Use*. The Forward Reply Model requires a parent to immediately forward a reply to its child without entering its own critical section, regardless of whether or not it has a request pending. The Use Reply Model allows a parent P_c to enter its critical section upon receiving a Reply message from its parent P_p, if P_c currently has a Request message outstanding.

Release Models. PADME provides two Release Models: *Forward* and *Use*. The Forward Release Model requires a participant to immediately forward a Release message to its parent without entering its own critical section, regardless of whether it has a request pending. The Use Release Model allows a participant to enter its critical section when it receives a Release message from one of its children, if the participant has an outstanding Request pending.

When applying the Use model, the participant must append its identifier onto the Release message if it entered its critical section (as shown in Figure 5, where each participant appends their release information to their parents when the critical sections have already been entered), which may result in a Release message containing multiple instances of the participant identifier. Consequently, appropriate data structures should be used to allow for these duplicates (*e.g.*, multisets). These duplicates enable proper bookkeeping along the token path since up to two Request messages may require removal from each affected priority queue.

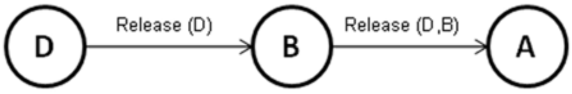


Fig. 5. Appending Identifier to Release Message

Priority Models. PADME provides two Priority Models: *Level* and *Fair*. The Level Priority Model means that one Request of the tuple form Request $< I_m, P_m, C_m >$ should be serviced before Request $< I_n, P_n, C_n >$ if $P_m < P_n$. P_x stands for the priority of the participant identified by I_x, and C_x refers to the request id or clock. If a tie occurs, the clocks C_x are compared first and then the identifiers. This ordering does not guarantee the absence of priority inversions, and priority inversions may happen when the token is in play (walking up or down the tree).

The Fair Priority Model means that one Request of the form Request $<$ $I_m, P_m, C_m >$ should be serviced before Request $< I_n, P_n, C_n >$ if $C_m < C_n$.

Upon a tie, the priority levels are compared, followed by the identifiers. The Fair Priority Model will result in all participants eventually being allowed into a critical section (assuming bounded critical section time and finite time message delivery), whereas the Level Priority Model makes no such guarantees.

Overview of PADME's Mutual Exclusion Algorithm. When a participant needs to enter its critical section (*e.g.* an agent is requesting exclusive access for writing to a cloud file system), it sends a Request message to its parent, who then forwards this Request up to its parent, until eventually reaching the root node. The Request message is a tuple of the form Request $< I, P, C, D >$, where I is the identifier of the requesting participant, P is the priority level (level), C is a timer or request id, and D is a user data structure that indicates the shared resource id (*e.g.*, the name of the file that will be accessed in the cloud file system) and any other data relevant to business logic. There is no reason that any of these variables be limited only to integers. For more information on the election of cloud entity and volume identifiers that may be useful for a cloud file system implementation, please see the Renaming Problem [4].

The choice of a timer mechanism (also known as a request id) may result in varying ramifications on the Fair Priority Model, discussed in Section 2.3. A timer should be updated (1) only when sending a Request or (2) any time a Request, Reply, or Release message with the highest time that of the agent who is receiving message or the time indicated in the message sent. The latter method will result in time synchronization across agents which can be helpful in synchronizing fairness in late joining agents or when switching from Level Priority Model to Fair Priority Model. Resending a Request does not increase the local request count. A Request may be resent if the parent participant faults or dies to ensure that a Request is serviced eventually by the root.

The root participant decides which Request to service according to a priority mechanism. After determining who gets to enter their critical section next, a Reply message is sent of the form Reply $< I, C >$ or $< I, C, D >$ where I is once again the identifier of the requesting participant, C is the count of the Request, and D is an optional parameter that may indicate business logic information, *e.g.*, the name of the file to be overwritten. Once a Reply message reaches the intended requesting participant, the requesting participant enters its critical section.

Upon exiting the critical section, the requesting participant must send a Release message to its parent participant, who forwards this Release message to its parent until the root receives the message. Release messages have the form Release $< I_0, I_1, \ldots I_n >$ or $< I_0, D_0, I_1, D_1, \ldots I_n, D_n >$ where $I_0, I_1, \ldots I_n$ is a list of participant identifiers that used their critical section along this token path, and $D0, D1, \ldots Dn$ is a parameter that may indicate business logic information *e.g.*, the frequency that is being released. The root participant and any participant along the token path should remove the first entry of each identifier in$< I_0, I_1, \ldots I_n >$ before forwarding the Release to its parent for proper bookkeeping.

2.3 QoS Properties of the PADME Algorithm

PADME's Request, Reply, Release, and Priority Models described in Section 2.2 are orthogonal and may be interchanged by the user to accomplish different QoS, higher fault tolerance, reduced message complexity at key contention points, or critical section throughput during runtime. Each combination has certain QoS properties that may fit an application's needs better than the others, *e.g.*, each has certain synchronization delay characteristics, throughput, and even message complexity differences during fault tolerance. To simplify understanding of the different combinations of these models, we created a system of model combinations that we call Request-Grant-Release settings that codify these combinations.

Non-fault Tolerance Case. PADME's most robust Request-Reply-Release setting is the Replace-Use-Use model, which corresponds to the Replace Request Model, Use Reply Model, and Use Release Model. The Replace-Use-Use setting requires each participant to keep a priority queue for child Requests (described further in Section 2.2), but its primary purpose is to limit the number of message resends during participant failures or general faults to only the most important Requests in the queue. Replace-Use-Use is consequently very useful when reducing the number of messages in the network is a high priority.

PADME's Use Reply Model of the Replace-Use-Use combination allows a participant to enter its critical section before forwarding on a Reply message to an appropriate child. The Use Release Model allows a similar mechanism in the opposite direction, on the way back to root. Both of these use models work well in conjunction with the Fair Priority Model to not only decrease synchronization delay (and thus increase critical section throughput) but also favor higher priority participants, as those higher priority participants will be closer to root and may have up to two chances of entering a critical section along a token path from root to a requestor and back to root.

Even when the Forward-Forward-Forward combination is used, the higher priority participants closer to root will still have lower message complexity and lower average synchronization delay than lower priority participants (*e.g.*, leaf nodes). This results from the token path being longer from the leaf nodes to root. Consequently, placing frequently requesting participants closer to the root node in the logical routing network can result in increased performance (Section 2.3 analyzes message complexity and synchronization delay).

All of PADME's Fair Priority Model-based settings inherently may lead to priority inversions. PADME's Level Priority Model by itself, however, does not eliminate priority inversions. To eliminate priority inversions from occuring in PADME, the Level Priority Model must be used in combination with *-Forward-Forward.

If the settings contain Use Models for either the Reply or Release Models when the virtual token progresses towards an entity, it is possible that a higher priority request may be delayed at the root node that arrived after permission was granted to a lower priority entity. In practice, completely eliminating priority inversions is typically not as important as throughput. Settings that enable the Use Model for both Reply and Release models therefore have much higher throughput proportional to the depth of the spanning tree.

Fault Tolerance Case. There are several options for fault tolerance that can be supported by PADME, but we focus on the most straightforward to implement and still be robust to failures. We use a classic approach called a Byzantine view change [5] whenever a root node faults (*i.e.*, becomes unavailable). A Byzantine view change is a type of consensus that requires a majority agreement for electing a primary node. The only time this would be initiated would be when a non-root participant detects that its parent is unresponsive, attempts to connect to a new parent using the protocol discussed in Section 2.1, and receives no response from a higher priority entity.

Upon electing a new root node, the children of former root push all pending requests up to the new root, and the system resumes operation. When a token is believed to be lost by the root node, *e.g.*, after a configurable timeout based on a query of the participants for any outstanding tokens, a new token is generated.

If the root did not die—but believes a token has been lost—it can either multicast a query for the outstanding tokens and regenerate, or it can use targeted messages along the path the token took. In the latter case, the token will either pass normally with a release message or the root will have to regenerate a token. This process is shown in Figure 6. The root participant knows it sent a permission token to D and that it went through child B to get there. There is no reason to send a recovery message to anyone other than B and let it percolate down to D unless multicast or broadcast is being used and the operation is inexpensive for the network.

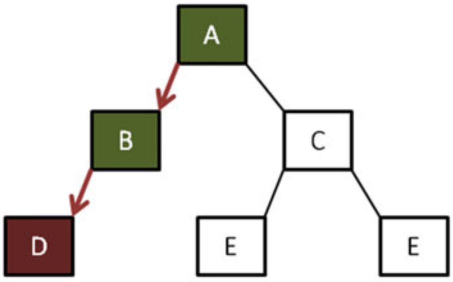


Fig. 6. Targeted Recovery of an Outstanding Token

Any time a parent connection is lost, the orphaned child establishes a new parent with the protocol outlined in Section 2.1 and resends all pending requests from itself and its children.

3 Evaluating the PADME Algorithm

This section evaluates results from experiments conducted on a simulated message-oriented prototype of the PADME algorithm over shared memory. We simulate critical section time (the time a participant uses its critical section), message transmission time between participants (the time it takes to send a Request, Reply, or Release message between neighboring cloud participants in the spanning tree), and critical section request frequency (how often a participant will request a critical section if it is not already in a critical section or blocking on a request for a critical section). Our experiments focus on the following goals:

1. **Quantifying the degree of QoS differentiation.** The goal of these experiments is to gauge whether or not the PADME algorithm provides QoS differentiation for participants in a public cloud (see Section 1) and whether or not the Request-Reply-Release models described in Section 2.2 have any tangible effects on QoS differentiation and throughput. Our hypothesis is that the PADME algorithm will provide significant differentiation based on proximity to the root participant.
2. **Measuring critical section throughput.** The goal of these experiments is to measure the critical section throughput of the PADME algorithm. Our hypothesis is that the PADME algorithm will provide nearly optimal critical section throughput for a distributed system, which is the situation where synchronization delay is t_m—the time it takes to deliver one message to another participant.

We created a simulator that allowed us to configure the Priority, Reply, and Release Models for several runs of 360 seconds. The experiments ran on a 2.16 GHZ Intel Core Duo 32 bit processor system with 4 GB RAM. The experiments were conducted on a complete binary tree with seven participants and a depth of 3 on a simulated network of seven participants: one high importance, two medium importance, and four low importance.

3.1 Quantifying the Degree of QoS Differentiation

The QoS Differentiation experiments quantified the ability of the PADME algorithm to differentiate between cloud customers and applications based on their priority—a derived metric that may directly correlate to money paid by the users of the system or a service's importance in serving cloud customers. We present the results as they relate to algorithm configurations and the tradeoffs between reducing priority inversions and increasing file access throughput.

Setup. Two experiments are presented here. The first has a message transmit time of 1 sec and a critical section entry time of 1 sec. The second experiment has a transmit time (t_m) of 0.5 sec and a critical section entry time of 1 sec. The latter experiment more accurately emulates network and Internet traffic since transmit time is rarely 1 sec. We use a convention of referencing models as Priority-Request-Reply-Release when describing PADME settings for brevity. Our PADME prototype does not yet include the Replace Request Model, so no configurations with the Replace Request Model are included in this section. We expect, however, that throughput and latency for this model should be identical to the Forward Request Model.

Analysis of Results. Figure 8 and Figure 7 outline the results for this test. The root participant had high priority, the participants on the second level had medium priority, and the leaf nodes on the third level had low priority.

In the analysis below we reference the PADME model configurations as Priority-Request-Reply-Release for brevity. Differentiation increases under certain models as the message time is decreased. This result appears to occur in Fair-Forward-Forward-Use, but is likely true of Forward-Use-Forward. Of the Request-Reply-Release combinations that appear to show the best differentiation amongst priority levels, those with Level Priority Model differentiate the best. Those with any type of Level Priority Model differentiate according to priority levels, which makes sense.

More interesting, however, is how the Fair-Forward-Use-Use, Fair-Forward-Forward-Use, and Fair-Forward-Use-Forward model combinations allow for better QoS in comparison to Fair-Forward-Forward-Forward. Although we are being fair in priority policy, this policy shows favoritism to the lower priority levels, which have more participants, and consequently get more critical section entries under a fair priority policy. Forward-Use-Use, Forward-Forward-Use, and Forward-Use-Forward offset these policy decisions by allowing critical section entries as the Reply and Release messages pass through participants, to allow for higher critical section entries than would have been possible with the more intuitive Forward-Forward-Forward. If we increased the number of high priority and medium priority participants, we would have even better differentiation during Fair Priority Policy because less time is spent in pure overhead states where the token is percolating back up the tree and not being used.

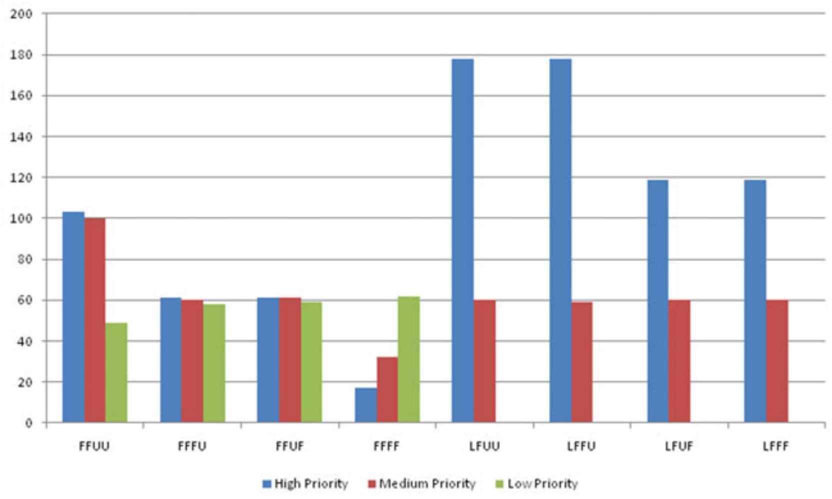


Fig. 7. Priority Differentiation with CS time of 1s and Message latency of 1ms

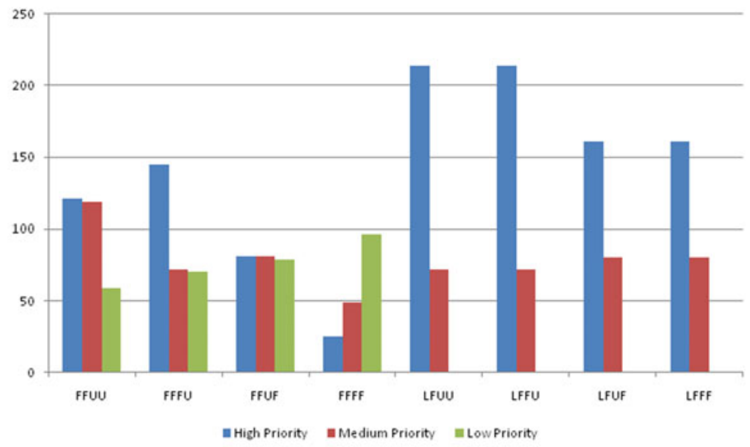


Fig. 8. Priority Differentiation with CS time of 0.5s and Message latency of 0.5ms

3.2 Measuring Critical Section Throughput

Differentiation can be useful, but if the system becomes so bogged down with messaging or algorithm overhead that file system or other shared resource throughput is greatly reduced, then no real benefits are available in the system. The experiments in this section guage the performance of the PADME algorithm in delivering file system access to customers, cloud applications, or persistent services.

Setup. Two experiments are presented here. The first experiment sets the message transmission time (t_m) to 1 ms, critical section usage time to 1 s, and we generate a new request once every 1 ms (when not using or blocking on a critical section request). The second experiment has a fast message transmission time of 0.5 ms (*i.e.*, more in line with a local area network transmit for a cluster within a cloud) and generates a new request every 0.5 ms (unless blocking on or using a critical section).

Our PADME prototype does not yet include the Replace Request Model, so no configurations with the Replace Request Model are included in this section. We expect, however, that throughput and latency for this model should be identical to the Forward Request Model.

Analysis of Results. Figure 9 and 10 show the results for these tests. These results are above our proposed theoretical max where synchronization delay = t_m, because participants are able to enter their critical sections (*e.g.*, access a file) both on a release and reply using the Use models.

Each model equals or outperforms a centralized solution. A centralized solution would have required a critical section entry (1 sec) plus two message transmissions—Release (1 sec) and Reply (1 sec)—per access resulting in only 120 critical section entries in a 360s test. The only configuration that performs worse than this one is the Fair-Forward-Forward-Forward combination. A centralized solution would have required a critical section entry (1 sec) plus two

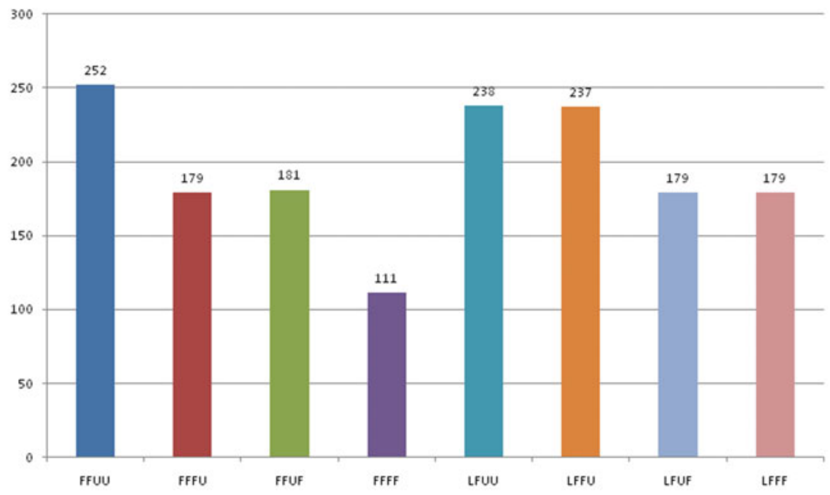


Fig. 9. Throughput with CS time of 1s and Message latency of 1ms

message transmissions—Release (0.5 sec) and Reply (0.5 sec)—per access resulting in just 170 critical section entries in a 360 sec test. Every model outperforms or equals a centralized solution in this scenario.

Some settings of the Priority-Request-Reply-Release models allow for the root participant (the highest priority participant) and medium priority participants to enter a critical section twice upon Reply or Release messages. This feature causes an additional critical section entry being possible during Use-Release with a synchronization delay = 0. This result occurs when a new request occurs in

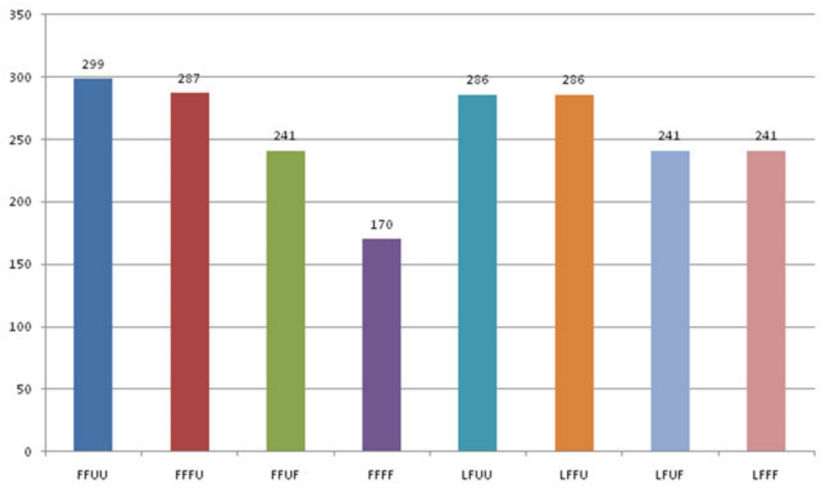


Fig. 10. Throughput with CS time of 0.5s and Message latency of 0.5ms

cloud applications on the root during or just after the root participant is servicing a separate request.

4 Related Work

This section compares our work on PADME with key types of mutual exclusion solutions in networks, grids, and clouds. We begin with discussions on central token authorities and end with descriptions of distributed techniques in clouds.

A basic form of mutual exclusion is a central authority that delegates resources based on priority or clock-based mechanisms. When a participant needs a shared resource, it sends a request with a priority or local timestamp to this central authority, and the central authority will queue up requests and service them according to some fairness or priority-based scheme. The Google File System (GFS) [2] uses such central authorities (called masters) and has tried to address these issues by creating a master per cell (*i.e.*, cluster). The GFS approach only masks the problems with the centralized model, however, and has a history of scaling problems [1].

The Lithium file system [13] uses a fork-consistency model with partial ordering and access tokens to synchronize writes to file meta data. These access tokens require a primary replica (centralized token generator) to control a branching system for volume ownership. Recovery of access tokens when primary replicas die requires a Byzantine view change of O(n) messages before another access token can be generated, and this can be initiated by almost anyone. In PADME, the view change should only be requested by someone along the token path. When the root node dies with a token still in it, the immediate children of the root would be in the token path and could request a view change. If there are no pending writes or reads, no change may even be necessary, and the root could resolve its fault and continue operations.

Distributed mutual exclusion algorithms have been presented throughout the past five decades and have included token and message passing paridigms. Among the more widely studied early distributed algorithms are Lamport [6] and Ricart-Agrawala [7], which both require O(n^2) messages, and Singhal [8], which uses hotspots and inquire lists to localize mutual exclusion access to processes that frequently enter critical sections. These algorithms are not applicable to cloud computing, where faults are expected. Singhal's algorithm also does not differentiate between priorities since it prefers frequent accessors (which might be free or reduced-payment deployments).

Message complexity has been further reduced via quorum-based approaches. In quorum-based approaches, no central authority exists and the application programmer is responsible for creating sets of participants that must be requested and approved for the critical section to be granted. For the Maekawa quorum scheme to function [9], each set must overlap each other set or it will be possible for multiple participants to be granted a critical section at the same time. If the sets are constructed correctly, each participant has a different quorum set to get permission from, and mutual exclusion is guaranteed. The problem is automatable and involves finding the finite projection plane of N points, but suffers performance and starvation problems (potentially of high priority participant) with faults.

More recently, a distributed mutual exclusion algorithm was presented by Cao et. al. [10]. This algorithm requires consensus voting and has a message complexity of O(n) for normal operation (*i.e.*, no faults). In contrast, our PADME algorithm requires O(d) where d is tree depth—$O(log_b n)$ where b is the branching factor of the spanning tree discussed in Section 3. The Cao et. al. algorithm also appears to require a fully connected graph to achieve consensus, and does not support configurable settings for emulating many of PADME's QoS modes (such as low response time for high priority participants).

Housni and Trehel [11] presented a grid-specialized token-based distributed mutual exclusion technique that forms logical roots in local cluster trees, which connect to other clusters via routers. Each router maintains a global request queue to try to solve priority conflicts. Bertier et. al. [12] improved upon Housni and Trehel's work by moving the root within local clusters according to the last critical section entry. This improvement, however, could result in additional overhead from competing hot spots, where two or more processes constantly compete for critical sections. Both algorithms treat all nodes and accesses as equals and are susceptible to the problems in Singhal [8] and consequently are non-trivial to implement for a public cloud where paying customers should have differentiation. Moreover, tokens can become trapped in a cluster indefinitely.

5 Concluding Remarks

This paper presented an algorithm called *Prioritizeable Adaptive Distributed Mutual Exclusion* (PADME) that addresses the need to provide differentiation in resource acquisition in distributed computing scenarios. We demonstrated its usefulness in cloud computing environments without requiring a centralized controller. Although we motivated PADME in the context of cloud file systems, it can be used for any shared cloud resource.

The following are lessons learned from the development of this mutual exclusion algorithm:

1. High critical section throughput with differentiation is possible. PADME provides differentiation based on participant priority levels and proximity to the logical root participant of the network. It also provides cloud application developers with four orthogonal models for variety and tighter control of critical section entry. The benefits of the PADME algorithm are high critical section throughput and low synchronization delay between critical section entries, especially when there is high contention for a shared resource.

2. The cost of differentiation is felt by those farthest from the root. In certain settings of the PADME algorithm—especially those using a Level Priority Model—the wait for access to the file system or shared resource could be indefinite. This situation will occur when the cloud is under heavy stress loads, and there is high contention for the shared resource, *e.g.*, when high priority cloud users or applications are constantly accessing the system.

3. Fault tolerance should be built-in. Retrofitting fault tolerance into a complex distributed system is hard. The Google File System attempted to solve this with redundant master controllers, but this caused the issue with faulty cloud hardware to just be harder to deal with. Building fault tolerance into a cloud solution from the inception helps reduce time and effort across the lifecycle.

4. Distributed mutual exclusion is still an important topic in Computer Science. We presented results and analysis that show clear differentiation based on priority level and high critical section throughput. Additional research challenges remain, however, including priority differentiation during tree rebuilding in heavy fault scenarios, reducing message complexity during root consensus changes, virtual overlay networks for individual resources or high priority file system volumes, and secure information flow across the spanning tree to prevent applications or users from snooping file information.

Our ongoing work is empirically evaluating PADME in the context of real public cloud platforms, such as Amazon EC2 and off-the-shelf hypervisors. We are also considering new Request, Reply, and Release Models, as well as more fault tolerance options, to PADME. The open-source PADME algorithm Java code and tests/simulator used in Section 3 are available for download at `qosmutex.googlecode.com`.

References

1. McKusick, M.K., Quinlan, S.: Gfs: Evolution on fast-forward. Queue 7, 10:1–10:20 (2009)
2. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 29–43. ACM, New York (2003)
3. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. SIGOPS Oper. Syst. Rev. 37, 29–43 (2003)
4. Kshemkalyani, A.D., Singhal, M.: Distributed Computing: Principles, Algorithms, and Systems, 1st edn. Cambridge University Press, New York (2008)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Third Symposium on Operating Systems Design and Implementation (OSDI). USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS, New Orleans, Louisiana (1999)
6. Lamport, L.: Ti clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558–565 (1978)
7. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. Commun. ACM 24, 9–17 (1981)
8. Singhal, M.: A dynamic information-structure mutual exclusion algorithm for distributed systems. IEEE Trans. Parallel Distrib. Syst. 3, 121–125 (1992)
9. Maekawa, M.: An algorithm for mutual exclusion in decentralized systems. ACM Trans. Comput. Syst. 3, 145–159 (1985)
10. Cao, J., Zhou, J., Chen, D., Wu, J.: An efficient distributed mutual exclusion algorithm based on relative consensus voting. In: International Symposium on Parallel and Distributed Processing, vol. 1, p. 51b (2004)

11. Housni, A., Trehel, M.: Distributed mutual exclusion token-permission based by prioritized groups. In: Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, p. 253. IEEE Computer Society, Washington, DC, USA (2001)
12. Bertier, M., Arantes, L., Sens, P.: Distributed mutual exclusion algorithms for grid applications: A hierarchical approach. J. Parallel Distrib. Comput. 66, 128–144 (2006)
13. Hansen, J.G., Jul, E.: Lithium: virtual machine storage for the cloud. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 15–26. ACM, New York (2010)

Towards Pattern-Based Reliability Certification of Services

Ingrid Buckley[1], Eduardo B. Fernandez[1], Marco Anisetti[2], Claudio A. Ardagna[2], Masoud Sadjadi[3], and Ernesto Damiani[2]

[1] Department of Electrical Engineering and Computer Science
Florida Atlantic University
777 Glades Road, Boca Raton, Florida, USA
ibuckley@fau.edu,ed@cse.fau.edu

[2] Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
Via Bramante 65, 26013 Crema (CR), Italy
firstname.lastname@unimi.it

[3] School of Computing and Information Sciences
Florida International University
University Park, 11200 SW 8th St., Miami, Florida, USA
sadjadi@cs.fiu.edu

Abstract. On Service-Oriented Architectures (SOAs), the mechanism for run-time discovery and selection of services may conflict with the need to make sure that business process instances satisfy their reliability requirements. In this paper we describe a certification scheme based on machine-readable reliability certificates that will enable run-time negotiation. Service reliability is afforded by means of reliability patterns. Our certificates describe the reliability mechanism implemented by a service and the reliability pattern used to implement such a mechanism. Digital signature is used to associate the reliability claim contained in each certificate with the party (service supplier or accredited third-party) taking responsibility for it.

1 Introduction

When Service-Oriented Architectures (SOA) came of age, no specific reliability technology for Web services was available; reliability mechanisms designed for Web servers, such as server redundancy, were used in its stead. Later, standards for reliable service invocation like WS-Reliability and WS-ReliableMessaging emerged [4], but the problem of achieving reliability of SOA-based Web services remains unsolved.

When trying to address the problem of Web service reliability, one has to recognize that many of the features that make SOAs attractive, such as run-time service recruitment and composition, conflict with traditional reliability models and solutions. Reliability relates to a system's ability to function correctly in the presence of faults, errors, and failures. This ability is traditionally achieved at system design time; but on a SOA – especially one hosted on a virtual execution environment like a cloud – the "system" that will execute a given business process instance is put together run-time via service discovery and selection. How can we ensure that the overall reliability requirements will be met? Our proposal to answer this question is a scheme for the *reliability certification* of services using *reliability patterns*. In our approach, a Web Service reliability certificate is a machine-readable representation of the reasons why a given service claims to be reliable, including the reliability mechanism the service relies on and the reliability pattern used to implement it.

As we shall see in the following, we use a POSA format for reliability patterns that provide a concise yet expressive way to specify how a given reliability mechanism is implemented by a service. Like other certification schemes, we envision that the party taking responsibility for a reliability claim will digitally sign the corresponding reliability certificate. Depending on the business scenario, the signing party can be the service supplier itself, or a third party accredited in the framework of a regulated certification process, as the ones adopted for life- or mission-critical applications [10]. Such certificates can then be used at service discovery time to make sure that the reliability requirements of a business process are met by all the services recruited to execute it.

A major difference between our reliability certificates and other types of Service Level Agreement-style metadata is runtime monitoring of reliability using reliability patterns [1,10,21]. Indeed, the SOA framework makes it possible to instrument Web services for monitoring the reliability mechanism implemented by the pattern mentioned in the certificate. At service discovery time, monitoring rules can be evaluated to check whether the certificate is still valid or not for a specific process execution, without requiring re-certification.

In the remainder of the paper we describe the building blocks of our proposal and how they can be put together to obtain a full scheme for the reliability certification of SOA Web services. Namely, in Section 2, we present the concept of pattern for improving system reliability. In Section 3, we describe how reliability patterns can be used to validate the reliability of services. In Section 4, we present our certification scheme, illustrating our machine-readable certificate format and a first solution to the integration of SOA and reliability certification. Finally, in Section 5, we present our related work and, in Section 6, we give our conclusions.

2 Using Patterns for Improving Reliability

The widespread diffusion of Web services and SOA is raising the interest for SOA-based implementation of life- and mission-critical applications for which reliability is a crucial requirement.

Often, in a traditional component-oriented scenario, reliability is added to the systems after the implementation phase, but experience has shown that this is not the most effective way to implement reliable services [13,20]. In this paper, we aim to define an approach for building reliable services that involves incorporating reliability in every phase of the system design and throughout the entire software development life cycle [5,27]. There are five major reliability techniques that are used to handle failures and their consequences in a system [23] as follows.

– *Detection*. Detecting the occurrence of an error.
– *Diagnostic*. Locating the unit or component where the error has occurred.
– *Masking*. Masking errors so as to prevent malfunctioning of the system if a fault occurs.
– *Containment*. Confining or delimiting the effects of the error.
– *Recovery*. Reconfiguring the system to remove the faulty unit and erasing the effects of the error.

These five techniques can be implemented using the following reliability mechanisms:

1. *Redundancy*. The duplication of critical components in a system with the intention of increasing the reliability of the system. This mechanism is often applied to chemical, power, nuclear, and aerospace applications.
2. *Diversity*. Requires having several different implementations of software or hardware specifications, running in parallel to cope with errors or failures that could arise directly from a specific implementation or design.
3. *Graceful degradation*. This mechanism is essential in systems where, in the event of a failure, a system crash is highly unacceptable. Instead, some functionality should remain in the event of a failure. If the operating quality of the system decreases, the decrease should be proportional to the severity of the failure.
4. *Checking and monitoring*. Constant checking of the state of a system to ensure that specifications are being met is critical in detecting a fault. This mechanism, while very simple, plays a key role in obtaining a fault tolerant system.
5. *Containment*. Faults are contained within some specific execution domain, which prevents error propagation across system boundaries.

Challenges in software reliability not only stem from the size, complexity, difficulty, and novelty of software applications in various domains, but also relate to the knowledge, training, and experience of the software engineers involved. Our approach is based on the notion of reliability patterns [23,24]. A pattern is an encapsulated solution to recurrent software or system problems in a given context, and it can be described using UML diagrams [28]. Reliability patterns support widespread application of best practices and best solutions, and offer an effective guideline for software developers that may not have expert knowledge and experience in reliable system development. Specifically, a reliability pattern [23,24] consists of several parts which provide a detailed description of the patterns' objective, and serves as a tangible reference for an effective reliability solution. We utilize the POSA template (see Table 1) to describe reliability patterns, mainly because it provides a complete encapsulation of a solution to a given problem. Additionally the POSA template is a widely accepted format that sufficiently

Table 1. POSA template

Intent
Example
Context
Problem and forces
Solution
Implementation
Example resolved
Known uses
Consequences
Related patterns

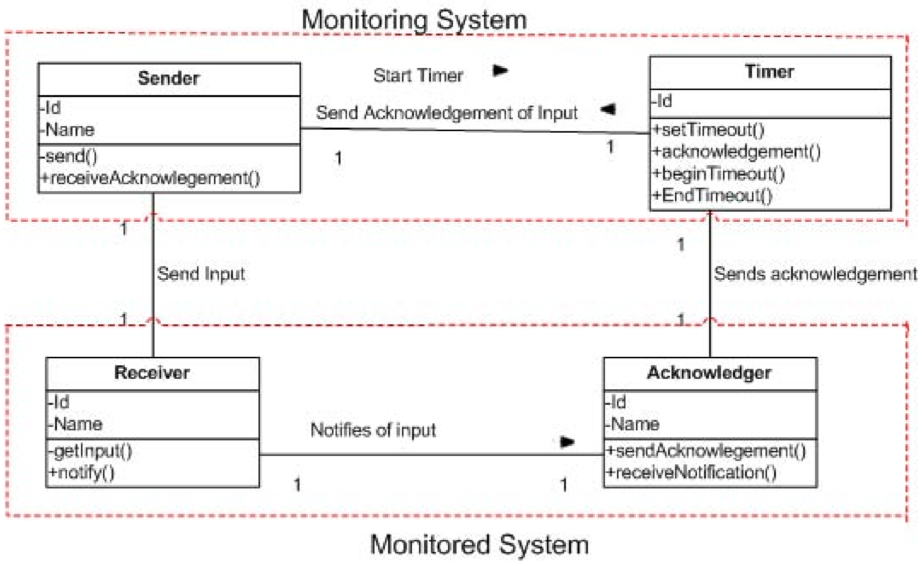
describes a reliability pattern. Reliability patterns described using the POSA template provide class and sequence diagrams that can be used to help generate monitoring rules consistent with the reliability aspect being sought.

3 Certifying Services Built Using Reliability Patterns

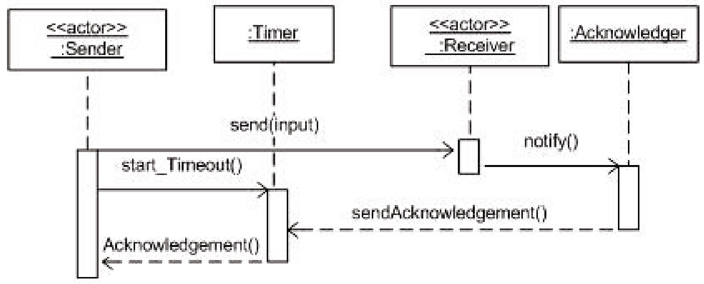
Once a system is built using some methodology that uses reliability patterns, we need a way to show it has reached a given level of reliability. In a SOA environment we can go even further; we can certify that the services satisfy some standards of reliability making digitally signed information available at runtime, that is, in SOA-based business process enactment. In our approach, this information includes the reliability pattern used in the service implementation. Therefore, we start by looking at some patterns to ascertain their effect on the reliability of a service.

3.1 A Reliability Pattern Solution

A reliability pattern includes a section which describes its solution (see Figure 1). This section includes a class diagram illustrating the structure of the solution and showing the class functions, and their relationships. Figure 1(a) depicts an example of a class diagram for the well-known *Acknowledgment* pattern whose intent is “to detect errors in a system by acknowledging the reception of an input within a specified time interval”. In the *Acknowledgment* pattern, the *Sender* in conjunction with the *Timer* constitute the *Monitoring System*, and the *Receiver* in conjunction with the *Acknowledger* entity constitute the *Monitored System*. In particular, the *Sender* is responsible for contacting the *Monitored System*. Whenever the *Sender* has sent the input to the *Receiver*, the *Timer*, that is responsible for counting down the timeout period every time an input is provided to the *Monitored System*, is activated. Upon receiving an input by the *Sender*, the *Receiver* notifies the *Acknowledger*. The *Acknowledger* is then responsible for sending an acknowledgment to the *Timer* for the received input. If the timeout period of the *Timer* expires for N consecutive times without



(a)



(b)

Fig. 1. Class diagram (a) and sequence diagram (b) for the Acknowledgment pattern

receiving an acknowledgment from the *Monitored System*, the *Timer* detects an error on the *Monitored System* and notifies the *Sender*.

The sequence diagram in Figure 1(b) provides the dynamics of one of the use cases of the pattern's solution. The two diagrams can be evaluated by transforming them to a formal representation and conducting some model checking to test that they perform some reliability function, that is, they avoid some failure.

3.2 A Priori Validation of Reliability Patterns

A priori validation of reliability patterns provides an estimation of the level of reliability that can be achieved before the actual system is implemented. A priori validation can be performed using the consequences and the failure/fault coverage of a reliability pattern.

The *consequences* of a reliability pattern describe the advantages and disadvantages of using the pattern. This information can be used a priori to compare patterns. A widely used criterion is the amount of computational resources required by the pattern.

The *failure/fault coverage* of a pattern, instead, is described as the number of failures that can be identified, masked, contained, or recovered with its use. This information can also be used for a priori selection of a pattern. For instance, the Dual Modular Redundancy (DMR) pattern can detect one fault but does not mask any faults; the Triple Modular Redundancy (TMR) pattern can detect two faults and mask one; the N-Modular Redundancy (NMR) pattern can detect (N-1) faults and mask (N-2) faults. Thus, the DMR pattern provides a lower level of reliability than the TMR pattern. Similarly, NMR provides a higher level of reliability than TMR. Figure 2 illustrates the structure of DMR, TMR, and NMR.

The evaluation of pattern consequences and coverage can permit to compare functionally equivalent services a priori (i.e., before they are invoked) on the basis of the level of reliability provided by the corresponding patterns. In this paper, however, we will focus on a posteriori reliability certificate checking.

3.3 A Posteriori Validation of Service Reliability

A posteriori validation of service reliability is an evaluation of the level of reliability provided by a given service implementation. We can assess the level of reliability in an implemented system that was built with the use of reliability patterns by evaluating different reliability metrics. There are many metrics in the literature [3,16,17,22,27,28] that can be used to calculate the reliability using data collected by monitoring an implemented system. We have selected some reliability metrics from the literature and proposed some of our own. Metrics (see Tables 2 and 3) are classified based on time- and cardinality-related failures. Such metrics correspond to the five reliability mechanisms discussed earlier in Section 2. Additionally, the metrics in Table 2 and Table 3 are easily measurable in a SOA environment, using available logging services and toolkits [7,18].

The metrics and related monitoring features can be used to enhance a priori comparison of services by taking into account the service implementation.

4 Machine Readable Certificates for Reliable Services

Current software certification schemes for security and dependability (e.g., Common Criteria [12]) provide human-readable, system-wide assurance certificates to be used at deployment and installation time. This type of certificates does not match the requirements introduced by a SOA in terms of runtime selection and composition of services. A solution providing machine-readable certificates is therefore required. Similarly to the definition of security property in [1], here we define a concrete specialization of the reliability property as the one that *provides enough information to support monitoring procedures aiming to establish*

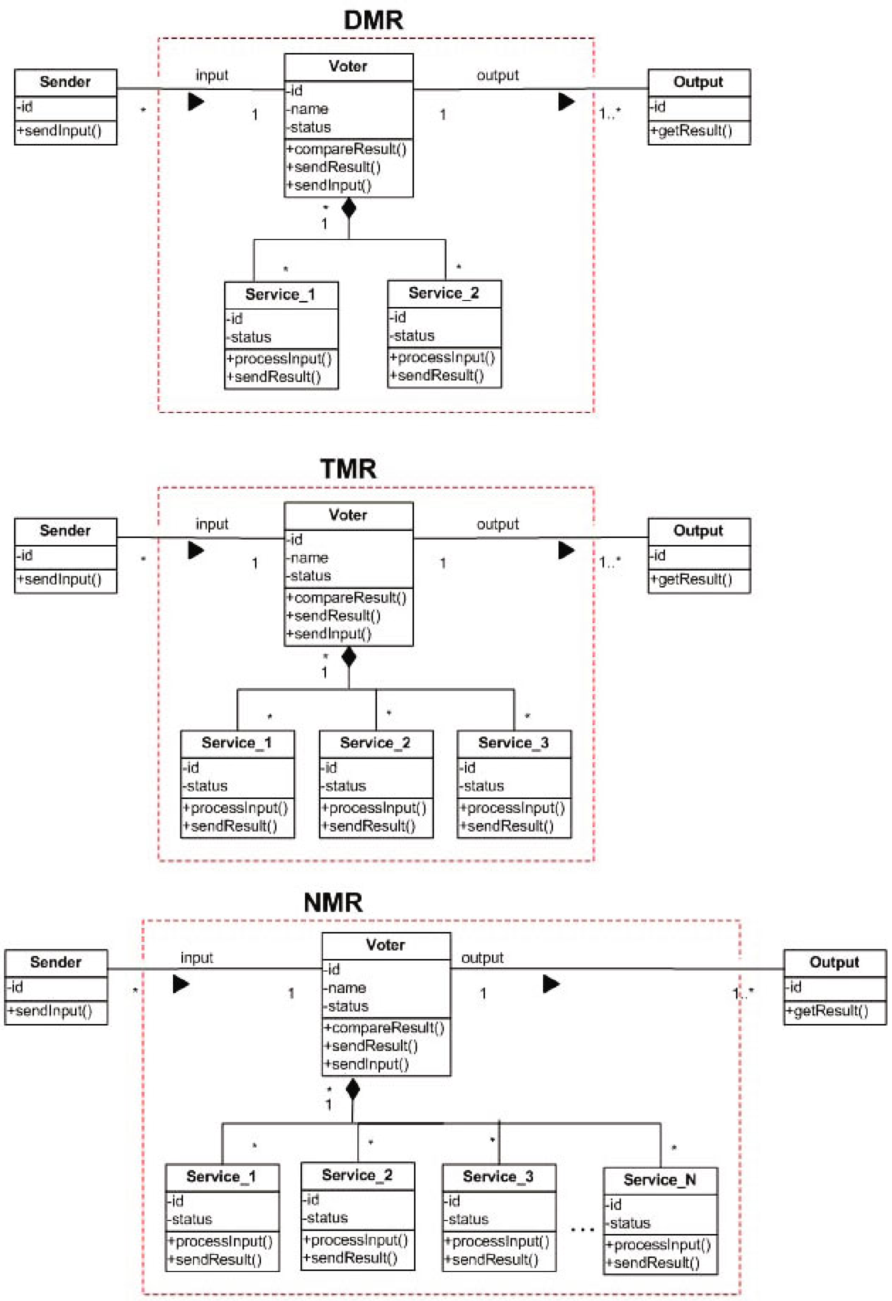


Fig. 2. Class diagram for DMR, TMR, and NMR patterns

Table 2. Time-related reliability metrics

Reliability Mechanism	**Time-Related Metric**	**Description**
Redundancy - Invokes one or more copy of the same mechanism	Time-to-Failure (TTF)	The time the service runs before failing
	Time-of-Failure (TF)	The time at which a failure occurs
	Mean Time to Fail (MTTF)	The average time it takes for the system to fail
	Failure Occurrence Rate (FOR)	The rate at which failures occur when the system is active
	Mean Time Between Failures (MTBF)	The average time before a failure occurs in the system
Diversity - Invokes one or more copy of a particular mechanism that performs the same function	Time-to-Failure (TTF)	The amount of time the service runs before failing
	Time-of-Failure (TF)	The time at which the failure occurred
Monitoring - Checks the system continuously to identify failures and sends alerts	Response-Time (RT)	The amount of time it takes to send an alert
	Time-to-Identify-Failure (TIF)	The time it takes to identify that a failure has occurred in the system
	Time-to-Failure (TTF)	The time the service runs before failing
Diagnosis - Identifies the source of failure	Investigation-Time (IT)	The time it takes to identify the unit that has failed
	Mean-time-to-Investigate (MTTA)	The average time it takes to investigate a failure
Masking - Hides the effects of a failure	Time-to-Replace-Failed-Component (TRFC)	The time it takes to replace a failed component
Containment - confines a failure to stop its propagation	Time-of-Failure-Arrest (TFA)	The time at which the failure was confined
	Time-to-Arrest-Failure (TAF)	Time it takes to confine the failure so that it does not propagate throughout the system
Recovery - Erases failure and restores normally	System-Recovery-Time (SRT)	The time needed for the system to recovery from a failure and return to a failure-free operational state
	Time-to-Recover (TTR)	The time needed to repair and restore service after a failure

if the property holds or not. In other words, a concrete definition of reliability specifies the mechanism in place to assert it (e.g., a redundancy mechanism) and the faults/failures the property is meant to support (e.g., loss of service failure). This information is represented as a set of *class attributes* specifying the mechanisms and/or the faults/failures. For instance, we can define a specialization of the *reliability* property whose class attributes are: *mechanism*=redundancy, *level*=4, *swapping time*=10ms, and *failure*=loss of service failure.

Besides the mechanisms and faults used to assert reliability, our machine-readable certificate includes all the other information in the reliability pattern used in service implementation and any available evidence supporting reliability. Our certificates are designed to be checked a posteriori, that is, on a working implementation of a service (Section 3). As we shall see, our evidence consists of a set of metrics that must be continuously monitored and updated using *monitoring rules.*

More in details, our machine-readable certificates for reliability include the following information:

Table 3. Cardinality-related reliability metrics

Reliability Mechanism	**Cardinality-Related Metric**	**Description**
Redundancy - Invokes one or more copy of the same mechanism	Number-of-Simultaneous-Failure (NSF)	The number of failures occurring at the same time
	Number-of-Invocation(NI)	Total number of calls made to a service
	Number-of-Failure (NF)	Total number of failures that occurred in the system
	Number-of-Expected-Failures (NEF)	The expected number of failures over a time interval
Diversity - Invokes one or more copy of a particular mechanism that performs the same function	Number-of-Simultaneous-Failure (NSF)	The number of failures occurring at the same time
	Number-of-Invocation(NI)	Total number of calls made to a service
	Number-of-Failure (NF)	Total number of failures that occurred in the system
	Number-of-Expected-Failures (NEF)	The expected number of failures over a specified time interval
Monitoring - Checks the system continuously to identify failures and sends alerts	Number-of-Failure-Alerts (NFA)	The total number of failure alerts sent
	Number-of-Successful-Acknowledgments (NSA)	The total number of successful acknowledgments sent
Diagnosis - Identifies the source of failure	Number-of-successful-Investigations (NSI)	The total number of times when the source of a failure is identified
	Number-of-Unsuccessful-Investigations (NUI)	The total number of times when the source of a failure is not identified
Masking - Hides the effects of a failure	Number-of-Failed-Components-Replaced (NFCR)	The total number of times a failed component is replaced
Containment - confines a failure to stop its propagation	Number-of-Confinement-Attempts (NUA)	The total number of times a confinement attempt is made
	Number-of-Resource-Needed-to-Contain-Failure (RNCF)	The percentage of system resources that was needed to prevent the failure from propagating throughout the system
	Number-of-Successful-Failure-Arrest (NSFA)	The total number of times a failure was detained
	Number-of-Unsuccessful-Failure-Arrest (NUFA)	The number of times a failure was not detained
Recovery - Erases failure and restores normally	Number-of-Successful-Recovery (NSR)	The total number of successful recoveries
	Number-of-Unsuccessful-Recovery (NUR)	The total number of failed or aborted recovery attempts
	Mean-time-to-Recover (MTTR)	The average time it takes to recover from a failure in the system

- *Reliability property*: a description of the concrete reliability property including class attributes with reference to mechanisms/faults used to assess it.
- *Reliability pattern*: a concise description of the reliability solution. We adopt the POSA template to describe reliability patterns.
- *Evidence*: a set of elements that specify the metrics and monitoring rules used for supporting the reliability in the certificate as follows.
  - *Set of metrics*: the metrics used to verify that a given property holds. For each metric, we define the expected value that is requested for the metric.
  - *Monitoring rules*: the rules used for monitoring the metrics in the evidence. Each rule contains a human-readable description and a reference to a standard toolkit for reliability monitoring on SOAs that permits to do the measurements of the corresponding metrics.[1] A violation of the monitoring rules produces a runtime revocation of the certificate.

Figure 3 shows our XML schema of the reliability certificate, which includes all information explained above. We note that the certificate contains the link to the certified service (*ServiceBinding* element), the reliability property (*Property* element), and the pattern used for the implementation (*Pattern* element). Then, it contains the evidence composed by a set of monitoring rules (*MonitoringRules* element) and a set of metrics (*Metrics* element). The *MonitoringRules* element includes a set of rule groups (*RuleGroup* element) that in turn contain a set of rules each one with an *ID* attribute. The *Metrics* element contains a set of metrics each one specifying the id of the rule to which the metric refers (*RuleID* attribute), the runtime and expected values for the metric, and an operator that specifies how to compare the two values. The runtime validity of the certificate is obtained by monitoring each rule in the evidence and comparing the metric runtime values with the expected values. This comparison can be simple (e.g., equality or inequality) or complex (e.g., including tolerance, bounding values).

When we come to the evaluation of the validity of the certificate for the service, we have to consider that all rules assume a boolean value at each time instant. A rule assumes value *true* if and only if *all* metric elements that refer to it are satisfied (i.e., the runtime value is compatible with the expected value). Rules in the *RuleGroup* element are then ANDed, while different *RuleGroup* elements are ORed, finally producing a boolean value for the *MonitoringRules* evidence. If it is *true*, the certificate is valid, otherwise it is revoked.

We now provide two examples of XML-based certificate for the service *Voter* (see Figure 2) implementing TMR and DMR patterns, respectively.

Example 1. Figure 4 shows an example of an XML-based certificate that proves the property *Reliability* for a *Voter* service available at the endpoint *http://www.acme.com/wsdl/voter.wsdl*. The Voter implementation follows the TMR pattern and includes a single *RuleGroup* element with two rules. The TMR requires software redundancy including at least three parallel instances of the

[1] Several commercial products are available, including the Microsoft Reliability Monitor [18].

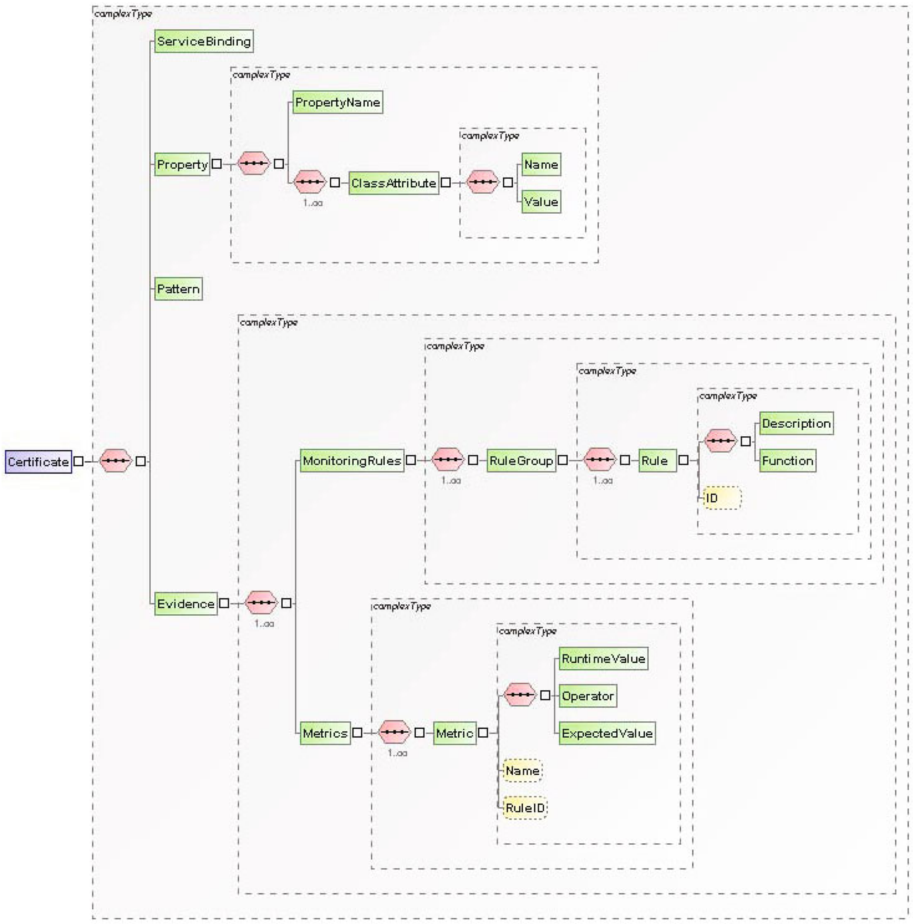


Fig. 3. XML schema of the reliability certificate

service. The first rule (rule 1) requires to continuously count all available service instances, using a toolkit function called *CountingRedundancy*(). The second rule (rule 2) requires that in case of an instance failure, the recovery time is measured. This measure is done via a function called *EvaluateRecoveryTime*(). The number of available instances and the recovery time are used in the *Number-of-Simultaneous-Failure* (*NSF*) and *Time-to-Recover* (*TTR*) metrics, respectively. In particular, the expected value for the number of available instances (or in other words the number of simultaneous failures the system can manage) is three, while the recovery time is 1 minute. The operators used in the comparison are both $\geq$. Since the runtime values of the metrics in the certificate are equal to/greater than the expected values, the certificate is valid.

Example 2. Figure 5 shows an example of XML-based certificate that does not prove the property *Reliability* for the *Voter* service available at the endpoint

```
<Certificate xsi:noNamespaceSchemaLocation="..." xmlns:xsi="...">
    <ServiceBinding>http://www.acme.com/wsdl/voter.wsdl</ServiceBinding>
    <Property>
        <PropertyName>http://www.acme.com/reliability/Reliability</PropertyName>
        <ClassAttribute>
            <Name>mechanism</Name>
            <Value>redundancy</Value>
        </ClassAttribute>
        <ClassAttribute>
            <Name>level</Name>
            <Value>3</Value>
        </ClassAttribute>
        <ClassAttribute>
            <Name>failure</Name>
            <Value>loss of service failure</Value>
        </ClassAttribute>
    </Property>
    <Pattern>TMR</Pattern>
    <Evidence>
        <MonitoringRules>
            <RuleGroup>
                <Rule ID="1">
                  <Description>Count all available service instances</Description>
                  <Function>CountingRedundancy()</Function>
                </Rule>
                <Rule ID="2">
                  <Description>
                    In case of an instance failure, measure the recovery time
                  </Description>
                  <Function>EvaluatingRecoveryTime()</Function>
                </Rule>
            </RuleGroup>
        </MonitoringRules>
        <Metrics>
            <Metric Name="NSF" RuleID="1" >
                <RuntimeValue>4</RuntimeValue>
                <Operator>greaterThan/equalTo</Operator>
                <ExpectedValue>3</ExpectedValue>
            </Metric>
            <Metric Name="TTR" RuleID="2">
                <RuntimeValue>1m</RuntimeValue>
                <Operator>greaterThan/equalTo</Operator>
                <ExpectedValue>1m</ExpectedValue>
            </Metric>
        </Metrics>
    </Evidence>
</Certificate>
```

Fig. 4. An example of valid certificate

http://www.acme.com/wsdl/voter.wsdl. This Voter implementation follows the DMR pattern. The monitoring rules and metrics are the same as the ones in Example 2, except for the expected value of *Number-of-Simultaneous-Failure* (*NSF*) metric that is equal to two. Since the runtime value of the redundancy metric is less than the expected value for the metric, the monitoring rule *rule 1* is not satisfied and the certificate is revoked.

A final aspect to consider is the integration of the reliability certification process and metadata within the SOA infrastructure. We need to provide a solution that allows clients to select the service that best fits their reliability requirements at runtime, on the basis of the information specified in the reliability certificate. This selection is performed by matching client requirements with service certificates.

4.1 An Architecture for Reliability Certificates Checking

Let us consider an enhanced SOA infrastructure composed by the following main parties. *i) Client* (*c*), the entity that needs to select or integrate a remote service based on its reliability requirements. *ii) Service provider* (*sp*), the entity implementing remote services accessed by *c*. *iii) Certification Authority* (*CA*),

```
<Certificate xsi:noNamespaceSchemaLocation="..." xmlns:xsi="...">
    <ServiceBinding>http://www.acme.com/wsdl/voter.wsdl</ServiceBinding>
    <Property>
        <PropertyName>http://www.acme.com/reliability/Reliability</PropertyName>
        <ClassAttribute>
            <Name>mechanism</Name>
            <Value>redundancy</Value>
        </ClassAttribute>
        <ClassAttribute>
            <Name>level</Name>
            <Value>4</Value>
        </ClassAttribute>
        <ClassAttribute>
            <Name>failure</Name>
            <Value>loss of service failure</Value>
        </ClassAttribute>
    </Property>
    <Pattern>TMR</Pattern>
    <Evidence>
        <MonitoringRules>
            <RuleGroup>
                <Rule ID="1">
                  <Description>Count all available service instances</Description>
                  <Function>CountingRedundancy()</Function>
                </Rule>
                <Rule ID="2">
                  <Description>
                    In case of an instance failure, measure the recovery time
                  </Description>
                  <Function>EvaluatingRecoveryTime()</Function>
                </Rule>
            </RuleGroup>
        </MonitoringRules>
        <Metrics>
            <Metric Name="NSF" RuleID="1" >
                <RuntimeValue>1</RuntimeValue>
                <Operator>greaterThan/equalTo</Operator>
                <ExpectedValue>2</ExpectedValue>
            </Metric>
            <Metric Name="TTR" RuleID="2">
                <RuntimeValue>1m</RuntimeValue>
                <Operator>greaterThan/equalTo</Operator>
                <ExpectedValue>1m</ExpectedValue>
            </Metric>
        </Metrics>
    </Evidence>
</Certificate>
```

Fig. 5. An example of revoked certificate

an entity trusted by one or more users to assign certificates. *iv) Evaluation Body* (*EB*), an independent, trusted component carrying out monitoring activities. *EB* is trusted by both *c* and *sp* to correctly check the certificate validity on the basis of the monitoring rules and metrics. *v) Service Discovery* (UDDI), a registry of services (e.g., [26]) enhanced with the support for reliability certificates and requirements.

Our service invocation process enhanced with reliability certification is composed by two main stages (Figure 6). In the first stage (Steps 1-2), *CA* grants a reliability certificate to a service provider *sp* based on a service implementation *s* and a reliability pattern. In the second stage (Steps 3-9), upon receiving the certificate for the service *s*, *sp* publishes the certificate together with the service interface in a service registry. Then the client *c* searches the registry and compares the reliability certificates of the available services. Once the client has chosen a certificate, it will ask to the trusted component *EB* to confirm its validity. *EB* checks that the corresponding monitoring rules hold and returns a result to *c*. If the result is positive *c* proceeds to call the service.

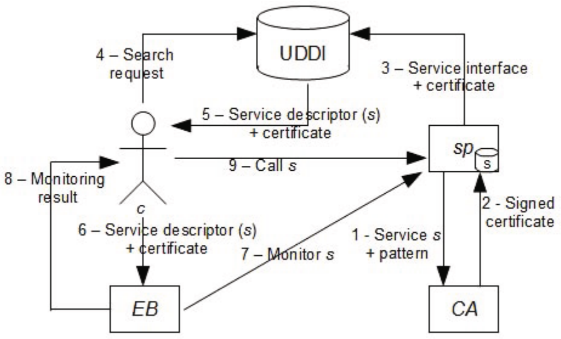


Fig. 6. A SOA enhanced with reliability certification

5 Related Work

Many approaches to software reliability have been proposed in the past and are presented in the following.

Becker et al. [2] proposed the Palladio Component Model (PCM) as a meta-model for the description of component-based software architectures with a special focus on the prediction of QoS attributes, especially performance and reliability. Spanoudakis et al. [25] proposed a tool called SERENITY which provides dynamic configuration and assembly of both security and dependability at runtime. Here, patterns are incorporated to realize the properties of security through location-based access control and dependability through monitoring and diagnostics. Monitoring and diagnostics are achieved with the use of a runtime framework called Everest. Everest employs event calculus to detect threats and failures in the system. The proposed solution also provides a monitoring framework for runtime checks of conditions related to the correct operation of security and dependability patterns. These conditions are specified as monitoring rules in Event Calculus.

Bernardi et al. [3] proposed an approach called MARTE-DAM for dependability modeling and analysis using UML. MARTE-DAM extends UML and includes features for the modeling, evaluation, and analysis of real-time systems. The authors first created a domain model that considers the main dependability concepts and organized them into top- and low-level packages. These packages include attribute descriptions and their relationships with each other. After the packages are defined they are stereotyped or mapped to a concrete profile. The packages usually contain dependability stereotypes and attributes called tags. They used a library of defined non-functional dependability types to measure different dependability metrics. They conducted a case study using a Message Redundancy Service (MRS) which is described using sequence diagrams. They then annotated the MRS with dependability properties using MARTE-DAM. They conducted an analysis and assessment of MARTE-DAM by transforming the MRS case into a Deterministic and Stochastic Petri Nets (DSPN). The DSPN model was then simulated to measure its dependability and performance. Similarly the work by Koopman [15] proposed the Ballista approach to quantitatively assess fault tolerance, quality assurance and computer security. Ballista can be used for robustness testing in operating systems. The aim is to use Ballista as a quality check for software assurance, by measuring generic robustness for fault tolerance and transforming security aspects into analogs of Ballista data types. Ballista uses both software testing and fault injection to test robustness.

Lastly, Mustafiz et al. [20] proposed a model-based approach for developers to analyze the dependability of systems whose requirements are described as use cases, and to help identify reliable and safe ways of designing the interactions in a system. They introduced probabilities at each step of the use case; this probability represents the success of that step. They utilized a probabilistic extension of state charts which are deterministic finite state automata to formally model the interaction requirements defined in the use cases. AToM^3, a tool used for formal modeling, was used to model the state charts. An analytical evaluation of the formal model is then carried out based on the success and failure probabilities of events in the environment.

Several other approaches adopted a Markovian approach to evaluate system reliability (e.g., [14,19]). Most of the above approaches produce model-based reliability evidence. However, experience with security certification suggests that the integration in our reliability certification scheme is not unfeasible.

Other works focused on evaluating dependability and reliability of services in a SOA environment. Cardellini et al. [6] proposed a model-based approach to the realization of self-adaptable SOA systems with the goal of addressing dependability requirements. The authors introduced a SLA (System Level Agreement) Monitor for monitoring aspects of the services agreed in SLA. According to their definition of SLA, the proposed solution includes a large set of parameters for different kinds of functional and non-functional service attributes, as well as for different ways of measuring them. In our vision, this type of SLA monitoring can be responsible for doing the measurements of the monitoring rules specified in the service certificate. Cortellessa and Grassi [9] analyzed the problem of reliability in SOA environments with focus on the composition and on mathematical aspects of the reliability modeling. They define a generic set of information to support SOA reliability analysis that can be monitored also in the case of composed services. Grassi and Patella [11] presented a reliability evaluation algorithm for SOA services. The algorithm uses the flow graph associated with a service to compute its reliability. Finally, Challagulla et al. [8] proposed a solution based on AI reasoning techniques for assessing the reliability of SOA-based systems based on dynamic collection of failure data for services and random testing. Again, failure data could be integrated as evidence in our reliability certificates.

6 Conclusions

Reliability is a key concern in most systems today. The SOA paradigm, which supports runtime selection and composition of services, makes it difficult to guarantee a priori the reliability of a process instance. In this paper, we presented a technique based on machine-readable reliability certificates using reliability pattern. In our work, we used the certificates to conduct a posteriori evaluation of reliable services. We are currently extending our approach to support a wider number of reliability patterns. Also, we are working on the integration of other types of dependability evidence.

Acknowledgements. This work was partly funded by the European Commission under the project ASSERT4SOA (contract n. FP7-257351). This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

1. Anisetti, M., Ardagna, C.A., Damiani, E.: Fine-grained modeling of web services for test-based security certification. In: Proc. of the 8th International Conference on Service Computing (SCC 2011), Washington, DC, USA (July 2011)
2. Becker, S., Koziolek, K., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software (JSS) 82(1), 3–22 (2009)
3. Bernardi, S., Merseguer, J., Petriu, D.: A dependability profile within marte. Journal of Software and Systems Modeling 10(3), 313–336 (2009)
4. Buckley, I., Fernandez, E., Rossi, G., Sadjadi, M.: Web services reliability patterns. In: Proc. of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009), Boston, MA, USA (July 2009), short paper
5. Buckley, I., Fernandez, E.: Three patterns for fault tolerance. In: Proc. of the International Workshop OOPSLA MiniPLoP, Orlando, FL, USA (October 2009)
6. Cardellini, V., Casalicchio, E., Grassi, V., Presti, F.L., Mirandola, R.: Towards self-adaptation for dependable service-oriented systems. In: Proc. of the Workshop on Architecting Dependable Systems (WADS 2008), Anchorage, AK, USA (June 2009)
7. Challagulla, V., Bastani, F., Paul, R., Tsai, W., Yinong Chen, Y.: A machine learning-based reliability assessment model for critical software systems. In: Proc. of the 31st Annual International Computer Software and Applications Conference (COMPSAC), Beijing, China (July 2007)
8. Challagulla, V., Bastani, F., Yen, I.L.: High-confidence compositional reliability assessment of soa-based systems using machine learning techniques. In: Tsai, J., Yu, P. (eds.) Machine Learning in Cyber Trust: Reliability, Security, Privacy, pp. 279–322. Springer, Heidelberg (2009)
9. Cortellessa, V., Grassi, V.: Test and analysis of web services. In: Baresi, L. (ed.) Reliability Modeling and Analysis of Service-Oriented Architectures, vol. 154, pp. 339–362. Springer, Heidelberg (2007)
10. Damiani, E., Ardagna, C.A., El Ioini, N.: Open source systems security certification. Springer, New York (2009)
11. Grassi, V., Patella, S.: Reliability prediction for service-oriented computing environments. IEEE Internet Computing 10(3), 43–49 (2006)
12. Herrmann, D.: Using the Common Criteria for IT security evaluation. Auerbach Publications (2002)

13. Holzmann, G.J., Joshi, R.: Reliable Software Systems Design: Defect Prevention, Detection, and Containment. In: Meyer, B., Woodcock, J. (eds.) VSTTE 2005. LNCS, vol. 4171, pp. 237–244. Springer, Heidelberg (2008)
14. Iyer, S., Nakayama, M., Gerbessiotis, A.: A markovian dependability model with cascading failures. IEEE Transactions on Computers 58(9), 1238–1249 (2009)
15. Koopman, P.: Toward a scalable method for quantifying aspects of fault tolerance, software assurance, and computer security. In: Proc. of the Computer Security, Dependability, and Assurance: From Needs to Solutions (CSDA 1998), York, U.K (July 1998)
16. Kopp, C.: System reliability and metrics of reliability, http://www.ausairpower.net/Reliability-PHA.pdf (accessed in date July 2011)
17. Lyu, M.: Handbook of Software Reliability Engineering. McGraw-Hill (1995)
18. Microsoft: Using Reliability Monitor, http://technet.microsoft.com/en-us/library/cc722107(WS.10).aspx (accessed in date July 2011)
19. Muppala, J., Malhotra, M., Trivedi, K.: Markov dependability models of complex systems: Analysis techniques. In: Ozekici, S. (ed.) Reliability and Maintenance of Complex Systems. NATO ASI Series F: Computer and Systems Sciences, vol. 154, pp. 442–486. Springer, Berlin (1996)
20. Mustafiz, S., Sun, X., Kienzle, J., Vangheluwe, H.: Model-driven assessment of system dependability. Journal of Software and Systems Modeling 7(4), 487–502 (2008)
21. O'Brien, L., Merson, P., Bass, L.: Quality attributes for service-oriented architectures. In: Proc. of the IEEE International Workshop on Systems Development in SOA Environments (SDSOA 2007), Minneapolis, MN, USA (June 2007)
22. Pan, J.: Software reliability.18-849b dependable embedded systems. Tech. rep., Carnegie Mellon University, http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/ (accessed in date July 2011)
23. Saridakis, T.: A system of patterns for fault tolerance. In: Proc. of the EuroPLoP Conference, Kloster Irsee, Germany (2002)
24. Saridakis, T.: Design patterns for fault containment. In: Proc. of the EuroPLoP Conference, Kloster Irsee, Germany (2003)
25. Spanoudakis, G., Kloukinas, C., Mahbub, K.: The serenity runtime monitoring framework. In: Spanoudakis, G., Kokolakis, S. (eds.) Security and Dependability for Ambient Intelligence, pp. 213–238. Springer, Heidelberg (2009)
26. Tsai, W., Paul, R., Cao, Z., Yu, L., Saimi, A., Xiao, B.: Verification of Web services using an enhanced UDDI server. In: Proc. of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003), Guadalajara, Mexico (January 2003)
27. Walter, M., Schneeweiss, W.: The modeling world of reliabiliy of reliability/safety engineering. LiLoLe Verlag (2005)
28. Walter, M., Trinitis, C.: Automatic generation of state-based dependability models: from availability to safety. In: Proc. of the 20th International Conference Architecture of Computing Systems (ARCS 2007), Zurich, Switzerland (March 2007)

Direct Anonymous Attestation: Enhancing Cloud Service User Privacy

Ulrich Greveler, Benjamin Justus, and Dennis Loehr

Computer Security Lab
Münster University of Applied Sciences
D-48565 Steinfurt, Germany
{greveler,benjamin.justus,loehr}@fh-muenster.de

Abstract. We introduce a privacy enhancing cloud service architecture based on the Direct Anonymous Attestation (DAA) scheme. In order to protect user data, the architecture provides cloud users with the abilities of controlling the extent of data sharing among their service accounts. A user is then enabled to link Cloud Service applications in such a way, that his/her personal data are shared only among designated applications. The anonymity of the platform identity is preserved while the integrity of the hardware platform (represented by Trusted Computing configuration register values) is proven to the remote servers. Moreover, the cloud service provider can assess user account activities, which leads to efficient security enforcement measures.

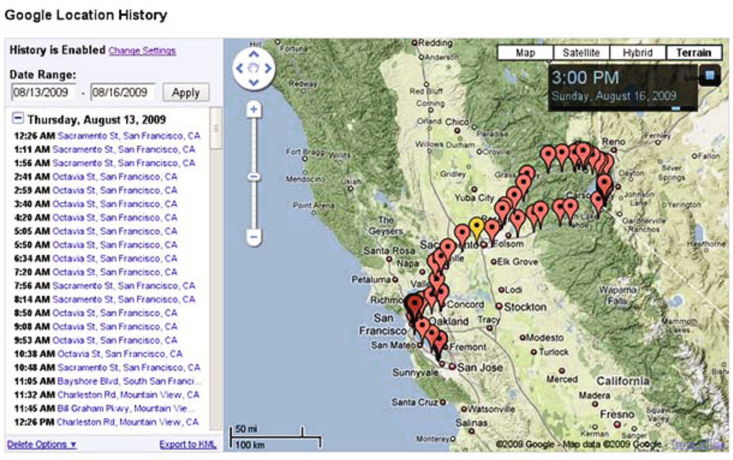
Keywords: Trusted Management, Privacy Enhancing, Direct Anonymous Attestation, Cloud Services.

1 Introduction

1.1 Cloud Services

Cloud Services are becoming ever more popular because they offer users mobility, scalability and reliability. For instances, users are able to access computer systems using a web browser regardless of their locations or what devices they are using. And as the cloud infrastructure is off-site (typically offered by the third party), it drastically reduces the operational expenditure of a business.

Google is currently one of the most popular Cloud Service providers. The Google business model is likely to set a trend for future Cloud Service providers. Under a Google account, a user is able to access his registered services offered by Google. This includes the popular *Gmail*, *Picasa Web Album*, and *Google Talk*. Google offers other tools, we shall name a few more as they pertain to the privacy discussion in the next section. *Google Buzz* [2] is a social networking and messaging tool integrated with *Gmail*. *Google Latitude* [3] is a geo-location tool that allows friends to know where the user is via *Google Map* (Figure 1). The tool has a “Location History” feature which stores and analyzes a user’s location (via user’s mobile phone) over time, and it also attempts algorithmically to determine where a person lives and works and other information relating to a person’s profile [23].


Google Location History

History is Enabled Change Settings

Date Range: 08/13/2009 - 08/16/2009 Apply

Thursday, August 13, 2009

12:26 AM Sacramento St, San Francisco, CA
1:11 AM Sacramento St, San Francisco, CA
1:56 AM Sacramento St, San Francisco, CA
2:41 AM Octavia St, San Francisco, CA
2:59 AM Octavia St, San Francisco, CA
3:40 AM Octavia St, San Francisco, CA
4:20 AM Octavia St, San Francisco, CA
5:05 AM Octavia St, San Francisco, CA
5:50 AM Octavia St, San Francisco, CA
6:34 AM Octavia St, San Francisco, CA
7:20 AM Octavia St, San Francisco, CA
7:56 AM Sacramento St, San Francisco, CA
8:14 AM Sacramento St, San Francisco, CA
8:50 AM Octavia St, San Francisco, CA
9:08 AM Octavia St, San Francisco, CA
9:53 AM Octavia St, San Francisco, CA
10:38 AM Octavia St, San Francisco, CA
10:48 AM Sacramento St, San Francisco, CA
11:05 AM Bayshore Blvd, South San Franci...
11:32 AM Charleston Rd, Mountain View, CA
11:45 AM Bill Graham Pkwy, Mountain Vie...
12:26 PM Charleston Rd, Mountain View, CA

Delete Options ▾ Export to KML

3:00 PM
Sunday, August 16, 2009

Fig. 1. Location History feature in Google Latitude

1.2 Privacy Concerns

Cloud Services often use a centralized approach when it comes to data storage of an individual's user account. For example, *Google Dashboard* [1] is a place where a user can login and view data that Google services have collected about the user. This includes user-provided information (names, addresses, profiles), and data contained in each of the service accounts (emails in *Gmail*, events in *Google Calendar*, photos in *Picasa*).

Some of the Google services have already drawn criticisms [22,7] because of privacy concerns. As Cloud Services such as Google collect more and more personal data and store them in a centralized manner, the consequence of exposing or leaking an account's information could be nightmarish. Just imagine someone taps into your account, can by clicking mouse a few times, discover who you are (name in the profile), where you live (home address), where you work (working address), who your friends are (mail contacts or *Google Buzz*), what your habits are (*Google Latitude*), and your financial data(*Google Checkout*). Such a scenario is not fiction-writing. According to Google [21] and its transparency report [4], the company receives constant requests from governments around the world to provide information on Google Service users. It is desirable that some measures of data control are available on the part of users.

1.3 Trusted Computing Background

The Trusted Computing Group (TCG) is an industry standards body formed to develop and promote specifications for trusted computing and security technologies. Trusted Platform Module (TPM) is a hardware chip embedded in platforms that can carry out various cryptographic functions. TPM has a set of

[1] www.google.com/dashboard

special volatile registers called *platform configuration registers* (PCRs). These 160-bit long registers are used to keeping track of the integrity information during a bootstrap process. The TCPA specification defines a set of functions for reporting PCR values [19,20]. When the TPM reports the values of the integrity metrics that it has stored, the TPM signs those values using a TPM identity.

The process of reporting the integrity of a platform is known as *remote attestation*. To achieve the goals of *remote attestation*, TCG has introduced in version 1.1 specifications the concept of privacy certification authority (Privacy CA) [19]. It works briefly as follows. Each TPM is equipped with an RSA key pair called an Endorsement Key (EK). The Privacy CA is assumed to know the Endorsement Keys of all valid TPMs. Now, when TPM needs to authenticate itself to a verifier, it generates a second pair of RSA key called an Attestation Identity Key (AIK), it sends the AIK public key to the Privacy CA, and authenticates this public key w.r.t the EK. The Privacy CA will check whether it finds the EK in its list and, if so, issues a certificate to the TPM's AIK key. The TPM can then forward this certificate to the verifier and authenticate itself w.r.t. this AIK. A user may lose his anonymity in this scheme if the privacy CA and the verifier collude.

As discussed by Brickell, Camenisch and Chen [10], version 1.2 of the TCG specifications incorporate the Direct Anonymous Attestation (DAA) protocol. This protocol is designed to address anonymity issues of remote attestation. In such a scenario, the server only learns that the platform is trusted, but not the identity of the platform. The DAA protocol offers user-controlled linkability.

1.4 Contribution

We introduce in this paper a Cloud Service Architecture based on the Direct Anonymous Attestation Scheme. The key features of the proposed service architecture are:

– A user is able to link Cloud Service applications in such a way, that his/her personal data are shared only among the designated application group.
– The service provider has better assessment and control of a user's accounts, which leads to efficient security enforcement measures.

The plan of the paper is as follows. Section 2.1 - 2.2 presents a high-level description of the Direct Anonymous Attestation (DAA) scheme. Some of the technical features of DAA are explained in section 2.3 - 2.5. The DAA-enabled Cloud Service Architecture is presented in section 3. Section 3.1 - 3.4 explains the components of this architecture and related implementation issues.

2 DAA Protocol Overview

The Direct Anonymous Attestation (DAA) scheme [10] draws upon techniques from the Camenisch-Lysyanskaya (CL) signature scheme [13], identity escrow and credential systems. The protocol allows remote attestation of a trusted platform while preserving the privacy of the system user. We outline below the important features of the DAA protocol. A more comprehensive description of the DAA scheme can be found in [20,10].

The DAA scheme is made up of two sub-protocols: *DAA join* and *DAA sign*.

2.1 DAA Join Protocol

The Join protocol enables the Host/TPM to obtain a DAA certificate from the DAA issuer.

Let (n, S, Z, R) be the DAA issuer public key, where n is an RSA modulus, and S, Z, R are integers modulo n. We assume that the platform (TPM) is already authenticated to the DAA issuer via its Endorsement Key, EK.

The TPM first generates a secret value f, and constructs the blind message $U := R^f S^{\nu'} \bmod n$ where ν' is a "blinding" value chosen randomly. The TPM also computes $N_I = \zeta_I^f$, where $\zeta_I = (hash(1||bsn_I))^{(\Gamma-1)/\rho} \pmod{\Gamma}$, and Γ, ρ are components of DAA issuer's public key. The TPM then sends (U, N_I) to the DAA issuer, and convinces the DAA issuer that U and N_I are correctly formed (using zero knowledge proof). If the DAA issuer accepts the proof, it will sign the blind message U, by computing $A = \left(\frac{Z}{US^{\nu''}}\right)^{1/e} \bmod n$, where ν'' is a random integer, and e is a random prime. The DAA issuer then sends the TPM the triple (A, e, ν''), and proves that A was computed correctly. The DAA certificate is the then $(A, e, \nu = \nu' + \nu'')$.

2.2 DAA Sign Protocol

The *sign protocol* allows a platform to prove to a verifier that it possesses a DAA certificate, and at the same time, to sign and authenticate messages. The TPM signs a message m using its DAA secret f, its DAA certificate, and the public parameters of the system. The message m may be an Attestation Identity Key (AIK) generated by TPM, or an arbitrary message. If m is an AIK, the key can be later used to sign PCR data or to certify a non-migratable key. In addition, the TPM computes $N_V := \zeta^f \bmod \Gamma$ where ζ is random or derived from the DAA verifier's basename depending on the anonymity requirement (see section 2.3). The value N_V allows for rogue tagging. The output of the *sign protocol* is known as the DAA Signature, σ.

The verifier verifies the DAA signature σ. The verifier needs to be convinced that the TPM has a DAA certificate (A, e, ν) from a specific DAA issuer. This is accomplished by a zero-knowledge proof of knowledge of a set of values f, A, e, and ν such $A^e R^f S^\nu \equiv Z \pmod{n}$. Further it needs to be shown that a message m is signed by the TPM using its DAA secret f, where f is the same value in the DAA certificate.

2.3 Variable Anonymity

The DAA protocol provides user-controlled anonymity and linkability. Precisely, a platform/TPM can achieve the following two statuses: 1. verifier linkable transaction 2. verifier non-linkable transaction. The statuses are controlled by the parameter: ζ. If non-linkable transactions are desired, a different and random value of ζ should be used for every interaction with a verifier. If linkable transactions are desired, ζ should be selected from a static basename based on the verifier, e.g $\zeta = (hash(1||bsn_V))^{(\Gamma-1)/\rho} \pmod{\Gamma}$.

2.4 Rogue Tagging

The DAA protocol has a built-in rogue tagging capability. A rogue TPM is defined when its secret value f has been extracted. Once a rogue TPM is discovered, the secret f values are distributed to all potential issuers/verifiers who add the value to their rogue-list. Upon receiving N_V, the verifier can check if N_V is equal to $\zeta^{\tilde{f}}$ for all $\tilde{f}$ stemming from rogue TPMs, and hence tag the TPM if necessary.

2.5 A Privacy Flaw Involving Corrupt Administrators

It is shown in [26] that an issuer and verifier can collude to break the anonymity of the user when linkable transactions are used. This privacy violation relies on the assumption that an issuer and a verifier share the same basename (i.e. $bsn_I = bsn_V$). The authors in the same paper suggest the following security fix. In the Join Protocol, compute $\zeta_I = (hash(0||bsn_V))^{(\Gamma-1)/\rho} \pmod{\Gamma}$. And in the Sign Protocol, compute $\zeta = (hash(1||bsn_V))^{(\Gamma-1)/\rho} \pmod{\Gamma}$. Now this leads to $N_I \neq N_V$ regardless $bsn_I = bsn_V$. So the issuer and the verifier can not identify the user by matching the values N_I and N_V.

3 DAA-Based Cloud Service Architecture

The proposed Cloud Service Architecture is displayed in Figure 2. For simplicity, our discussion is restricted to a single Cloud Service provider who acts both as an issuer and verifier. The user-privacy is preserved even when the Cloud Service

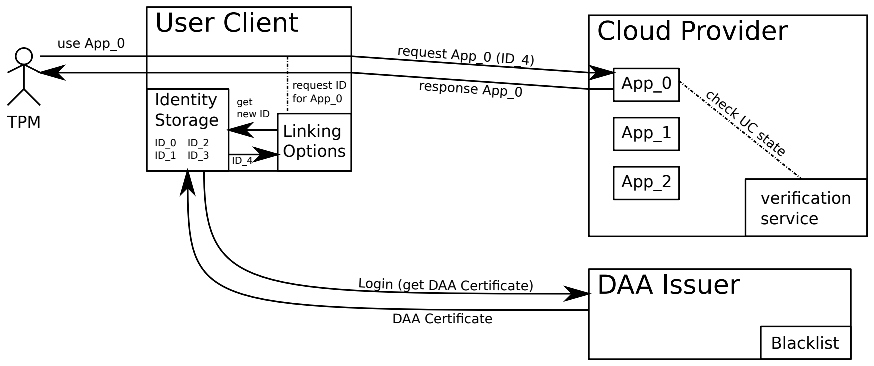


Fig. 2. Privacy-Enhanced Cloud Service Architecture

decides to employ another service for issuing DAA certificates (see section 2.5). A DAA certificate is issued when a user registers/login an account. The credential of the user is checked at this stage following the DAA Join protocol. After the login, a user is able to set linkability options (section 3.2) among the services offered by the Cloud. When a user requests the usage of a particular service, the permission is granted (or denied) by the Cloud Service acting as a verifier after further security analyses (see section 3.4). We discuss below in details the various components of this architecture and some involved implementation issues.

3.1 Service Login

A user is required to login in order to use the services offered by the Cloud. A successful login provides the user a one-time DAA certificate, so that the user can proceed to request further services. Figure 3 shows the behind-scene of a login session for a TPM based platform. The user login session is implemented based on the DAA Join protocol, and since ζ_I is constant (derived from the Cloud Service basename), a rogue-list can be computed and kept afresh at desired time interval. As soon as a TPM is compromised and its secret key f distributed, the service provider can tag the rogue TPM by augmenting the rogue-list. The tagged TPM will not be able to access the account at next login session.

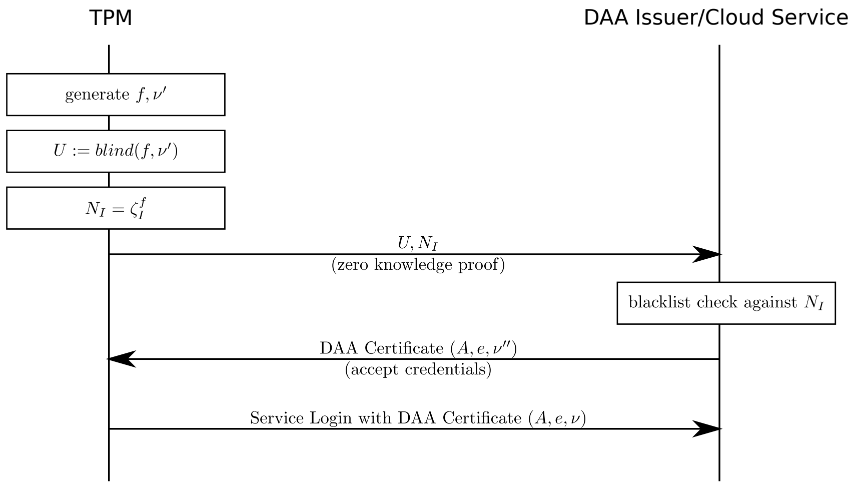


Fig. 3. Cloud Service Login for a TPM

3.2 User-Controlled Linkablility

Upon a successful login, a user is allowed to set linking options among the application services offered by the Cloud. By *Linkable* Applications, we mean that given two or more of service requests/usages originating from the same user, the cloud service is not able to link them and conclude they originate from the same user. Of course, linkbility has meaning only when anonymity is achieved. Anonymous attestation is achieved at the Service Login stage as explained in the previous section. There are three service link-statuses. Figure 4 shows a typical application linking scenario.

1. **Non-linkable Application.** Service is not able to link a user's transactions
2. **Single Application Linkability.** Service is able to link a user's transactions and data in a single application
3. **Multiple Applications Linkability.** Service is able to link a user's transactions and data across the application group

The linkability mechanism hinges upon how a basename is selected (see section 2.3). For non-linkable applications, a random basename (random ζ) is used during the service transactions. Typical non-linkable applications could be areas where a user performs web searches/browsing, and the user wishes to keep anonymous his search content and also his search history non-linkable. Linkable applications employ a static basename. In particular, each application group (services a user wishes to link) employs a basename which should reflect the service content of the application group (see section 3.3).

The decision as to what service applications to link is of course a personal one. For example, some users may prefer to link services that contain their financial data (credit card number, investment portfolio, etc.). Some limited service users may choose not to link any of the registered services. A Cloud Service should be able to track a user's activities in particular accounts. The history information contained in an account is necessary for purposes such as billing, or further

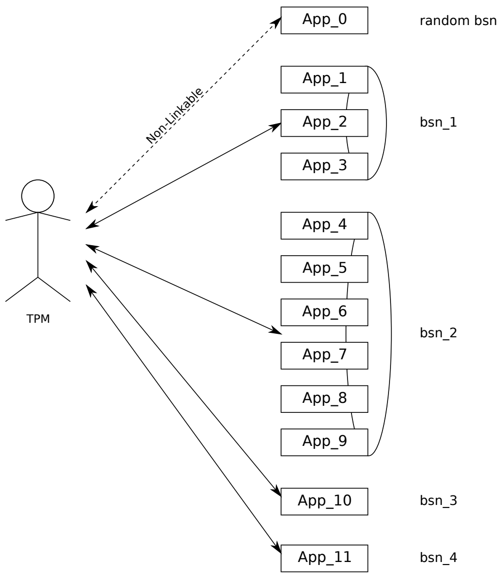


Fig. 4. User-Controlled Linkable Cloud Service Applications

service upgrading. On the other hand, by restricting information sharing among the service accounts, a user can be assured his/her information when divulged will only contain those parts of the profile.

3.3 Selecting a Basename

The linkable applications share a common basename. The basenames must be available as soon as a user sets the linking configuration, and before requesting service usages. The current DAA scheme does not provide protocol procedures specifying how basenames should be generated. One possible solution is to pre-generate a list of basenames containing all possible combinations of linkable applications. This list is stored on the server and becomes available once a user sets a particular linking configuration. However, this solution may become impractical as the number of services increases (leads to an exponential growth of options). A more efficient solution might be to create a basename generating program. Whenever a user sets the linking configuration, suitable basenames can be generated and saved on the servers. Smyth et als. [26] discussed some alternative approaches in constructing and managing the basename lists.

3.4 Account Suspension/Closing

DAA rogue tagging capability allows Cloud Service providers to suspend or close a user's account when they see suspicious behavior on the part of a user. The specifics of the rogue behavior is of course application dependent. For example, in an application involving software download/upgrade, the Cloud Service may require that remote platform to prove its trustworthiness by providing the platform's PCR values. The failure of compliance or unsatisfactory reporting results on the user part may lead to a rogue status. Other suspicious behavior could be: above-normal usage of a particular account, repeated account creation and deletion, and any other discretionary rules decided by the verifier.

Also since a common basename (constant ζ) is used among linked-applications, the Cloud Service (as a verifier) is able to update the rogue list regularly. The DAA Sign protocol can be efficiently carried out using batch proof and verification techniques [14,8]. In fact, Chen's asymmetric pairing based Sign Protocol [14] is extremely computationally efficient. For each signing process, the TPM is only required to compute one exponentiation if linkability is not required, and two exponentiations when linkability is required. The efficiency of this scheme comes from an ingenious use of a batch proof and verification scheme in proving the discrete logarithmic equality between two group elements y_1 and y_2 to two bases g_1 and g_2 respectively (i.e. $\log_{g_1} y_1 = \log_{g_2} y_2$) .

4 Related Work

The DAA scheme is introduced in the seminal paper [10]. The DAA protocol is designed to address anonymity issues of remote attestation. Privacy flaws were found after the introduction of the original protocol [25,26] and the corresponding security fixes are suggested in [24,26]. There have been work done to enhance capabilities of the original DAA scheme. Camenisch [12] suggested a hybrid anonymous attestation scheme which combines the DAA and the privacy CA approaches. Brickell and Li [11] introduced a new DAA scheme called *Enhanced Privacy ID*. The new DAA scheme while providing non-linkability, is capable of revoking a TPM even if the TPM private key is unknown.

Data security and privacy is one of the biggest challenges in Cloud Computing. Cloud data must be protected not only against external attackers, but also corrupt insiders (administrators). The *information-centric* approach [5,17] aims to make cloud data self-intelligent. In this approach, cloud data are encrypted and packaged with a usage policy. The data when accessed will consult its policy, create a virtualization environment, and attempt to assess the trustworthiness of the data environment (using Trusted Computing).

Applied Cryptography offers tools to address privacy and security questions related to cloud data. Recent advances in cryptography allow remote operations, manipulations and computations on encrypted cloud data. The predicate encryption scheme [27,9] allows cloud based searches on encrypted documents. Homomorphic encryption [18,28] and Private Information Retrieval (PIR) [15] can perform computations on encrypted data without decrypting.

To make cloud services more secure and reliable, Google has launched a prototype hardware **Cr-48**, which is tailor-designed to run the Google Chrome Operating System [1]. The prototype hardware is shipped with Trusted Platform Modules. About 60,000 Cr-48s were manufactured and distributed to testers and reviewers in early December 2010. Reviews published in mid-December 2010 indicated that while the project holds promise, it is still not market-ready [29]. In the EU framework, **PrimeLife** [6] is an ongoing research project funded by the European Commission. The main objective of the project is to bring sustainable privacy and identity management to future networks and cloud services.

5 Conclusion

Cloud Service is clearly becoming one of today's most popular Internet-based services, due to its cost-efficiency and flexibility. The future development of Cloud Services relies on mature technology deployment in areas, such as hardware/software security, data provision infrastructure, and reliable Third-party data control [16].

To better protect user data, we have in this paper introduced a cloud service architecture based on the Direct Anonymous Attestation scheme as outlined in the Trust Computing Group specification [20]. The theoretical DAA-based architecture provides cloud users the abilities of controlling the extent of data sharing among his service accounts, proving the integrity of his platform (PCR values) to remote servers, and the most important of all, preserving anonymity of the platform identity.

Acknowledgements. Many thanks go to the anonymous reviewers who provided detailed and insightful commentary on this paper.

References

1. Chrome Notebook, http://www.google.com/chromeos/pilot-program-cr48.html
2. Google Buzz, http://www.google.com/buzz
3. Google Latitude, http://www.google.com/latitude
4. Google Transparency Report, http://www.google.com/transparencyreport/governmentrequests/
5. An information-centric approach to information security, http://virtulization.sys-con.com/node/171199
6. The Primelife Project, http://www.primelife.eu/
7. Warning: Google buzz has a huge privacy flaw (February 2010), http://www.businessinsider.com/warning-google-buzz-has-a-huge-privacy-flaw-2010-2
8. Bellare, M., Garay, J.A., Rabin, T.: Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)
9. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
10. Brickell, E.F., Camenisch, J., Chen, L.: Direct Anonymous Attestation. In: ACM Conference on Computer and Communications Security, pp. 132–145 (2004)
11. Brickell, E., Li, J.: Enhanced Privacy ID: a direct anonymous attestation scheme with enhanced revocation capabilities. In: WPES, pp. 21–30 (2007)
12. Camenisch, J.: Better Privacy for Trusted Computing Platforms (Extended Abstract). In: Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 73–88. Springer, Heidelberg (2004)
13. Camenisch, J.L., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
14. Chen, L.: A DAA Scheme Using Batch Proof and Verification. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 166–180. Springer, Heidelberg (2010)
15. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM 45(6), 965–981 (1998)
16. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW 2009, pp. 85–90. ACM, New York (2009)
17. EMC. Information-centric security, http://www.idc.pt/resources/PPTs/2007/IT\&Internet_Security/12.EMC.pdf
18. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 169–178. ACM (2009)
19. Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.1b (2001), www.trustedcomputing.org

20. Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification, version 1.2 (2003), www.trustedcomputing.org
21. Privacy International. An interview with google on government access to personal information, https://www.privacyinternational.org/article/interview-google-government-access-personal-information
22. Privacy International. Privacy international identifies major security flaw in google's global phone tracking system, https://www.privacyinternational.org/article/privacy-international-identifies-major-security-flaw-google's-global-phone-tracking-system
23. Lambert, C.: Google latitude, now with location history and alerts (November 2009), http://googlemobile.blogspot.com/2009/11/google-latitude-now-with-location.html
24. Leung, A., Chen, L., Mitchell, C.J.: On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA). In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) Trust 2008. LNCS, vol. 4968, pp. 179–190. Springer, Heidelberg (2008)
25. Rudolph, C.: Covert Identity Information in Direct Anonymous Attestation (DAA). In: SEC, pp. 443–448 (2007)
26. Smyth, B., Ryan, M., Chen, L.: Direct Anonymous Attestation (DAA): Ensuring Privacy with Corrupt Administrators. In: Stajano, F., Meadows, C., Capkun, S., Moore, T. (eds.) ESAS 2007. LNCS, vol. 4572, pp. 218–231. Springer, Heidelberg (2007)
27. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
28. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption Over the Integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
29. Wolfgang, G.: Chrome OS is ahead of its time (December 2010), http://www.conceivablytech.com/4624/products/chrome-os-is-ahead-of-its-time

Trust Management Languages and Complexity

Krzysztof Sacha

Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warszawa, Poland
k.sacha@ia.pw.edu.pl

Abstract. Trust management is a concept of automatic verification of access rights against distributed security policies. A policy is described by a set of credentials that define membership of roles and delegation of authority over a resource between the members of roles. Making an access control decision is equivalent to resolving a credential chain between the requester and the role, which members are authorized to use a resource. A credential is an electronic document, formulated using a trust management language. This way, trust management languages are a tool for describing credentials and specifying access control policies in a flexible and modifiable way. This paper discusses the expressive power of trust management languages, describes a new extension to Role-based Trust Managements language RT^T, and evaluates the complexity of algorithm that is used for answering security queries.

Keywords: Access control, trust management, role-based trust management language, credential graph, credential chain.

1 Introduction

The traditional access control mechanism assigns to each protected resource an access control list (ACL), which enumerates the entities that have permissions to access the resource [1]. A decision on whether to allow or to deny access to a resource is based on a verification of identity of the requester. If the number of entities is big and many entities may have the same access rights with respect to resources, then assigning access rights to roles, i.e. sets of entities, rather than to individual entities, can help in reducing the size of the access control problem [2]. Such a schema can easily be implemented in such a system, in which the identity of entities that can make requests for resources is known in advance, e.g. in the form of user accounts that are created before the access to resources is really requested.

Quite another approach to access control is needed in open environments, in which the identity of potential users is not known in advance. For example, the users are identified by public keys and no user accounts are required. Trust management is a concept of decentralized access control, in which the decisions are based on credentials presented by the requesters, regardless of their identity. A credential is an electronic document, which describes a delegation of access rights from one entity (the issuer) to another entity or a group of entities. Such an approach separates a symbolic representation of trust (credentials) from the identity of users.

A set of credentials describes a security policy in terms of roles, role membership and delegation of authority between the members of roles. A security policy can be decentralized, if the members of particular roles may issue the credentials locally. Making an access control decision is equivalent to resolving a credential chain between the requester and the role, which is authorized to use the resource. This way, trust management becomes a concept of automatic verification of access rights against security policies [3]. The policies are created externally to the applications, and can be modified without the need to modify the executable program code of the application.

Credentials are implemented as electronic documents, composed of statements in a trust management language. Several languages to describe credentials and security policies have been developed and described in the literature. The goal of this research is to look at syntax, semantics and the expressive power of these languages, make some extensions to a Role-based Trust Managements language RT^T, and evaluate the complexity of an algorithm that is used for answering security queries.

The remaining part of the paper is organized as follows. Related work is described in Section 2. Trust management languages are summarized in Section 3. A motivating example of a policy-based access control system in an open SOA environment is presented in Section 4. Possible system architecture is discussed in Section 5, and an extension to RT^T is described in Section 6. An algorithm for resolving a set of RT^T credentials that define a policy and an evaluation of computational complexity are given in Section 7. The conclusions and plans for further research are in Section 8.

2 Related Work

There can be many independent entities in distributed computing environments, such as SOA systems, with authority to control access to a variety of system resources. Moreover, there can be a need of multiple entities to have input to the access control policy for a single resource. A secure implementation of such a distributed access control mechanism can be based on credentials that convey authorization to use a resource from one entity to another. A *credential* is a digitally signed certificate, which provides a copy of the public key of the issuer, the receiver and a description of the authority that is delegated from the issuer to the receiver of this credential.

A few such distributed authorization environments have been implemented and described in the literature. Examples are PolicyMaker [3], KeyNote [4], SPKI/SDSI [5] and Akenti [6]. All those systems use languages that allow assigning privileges to entities and use credentials to delegate permissions from their issuers to subjects.

In SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) a concept of distributed name space was introduced, according to which an entity could define a local name, related to certain privileges with respect to system resources, and then define members of the local name space or delegate responsibility for the name space membership to another entities. The definition of names and delegation of responsibility was expressed by means of credentials. An algorithm for building the semantics of a given set of credentials was defined with efficiency of order $O(n^3 l)$, where n was the number of certificates and l was the length of 'hops' in the longest responsibility delegation chain [5].

The concept of distributed name space was moved at a higher level of abstraction by the introduction of roles and Role-based Trust management languages [7-9].

There is a family of RT languages, with varying expressive power and complexity. The basic language RT_0 allows describing roles and role membership, delegation of authority over roles and role intersection. The issuers as well as the members of roles are entities, and no means for expressing complex security policies, such as threshold policy, exists in this language.

RT^T extends the expressive power of RT_0 significantly, by adding manifold roles and providing extensions to express threshold and separation of duties policies. A manifold role is a role that can be satisfied only by a group of cooperating entities. A threshold policy requires a specified minimum number of entities to agree before access is granted. Separation of duties policy requires that different entities must hold the conflicting roles. If a cooperation of these roles is required before access is granted, then a set of entities is needed, each of which fulfils only one of these conflicting roles. Both of these two policies mean that some roles cannot be fulfilled by a single entity and a group of entities must cooperate in order to satisfy these roles.

Apart of an informal interpretation, a formal definition of the language semantics, which gives meaning to a set of credentials in the application domain, has been provided. In [8], constraint DATALOG was used as the semantic foundation for the RT language family. According to this approach, credentials in RT were translated into DATALOG rules. This way, the semantics was defined in an algorithmically tractable way. A strict, set theoretic semantics of RT_0 was defined in [9] and of RT^T in [10,11]. An important difference between the two (RT_0 and RT^T) is such that RT_0 supports singleton roles only, and the meaning of a role is a set of entities, each of which can fulfill this role, while RT^T supports manifold roles, and the meaning of a role is a set of groups of entities that fulfill the role.

Security queries can be resolved with respect to a set of credentials that define the security policy within a system, by means of a credential graph, which is described in Section 7. The nodes of this graph are roles and groups of entities, which define the meaning of roles. Making a decision on the membership of a group of entities in a given role is equivalent to checking whether a path from this group to the role exists in the graph. An algorithm to construct a credential graph of a set of RT_0 credentials was introduced in [8]. An early version of the algorithm to construct a credential graph of RT^T credentials was described in [11], however, without an estimation of computational complexity. An improved algorithm to construct a credential graph of extended RT^T is presented in Section 7, together with an evaluation of complexity.

3 Trust Management Languages

All RT languages are built upon a set of three basic notions: Entities, role names and roles. An **entity** is someone who can participate in issuing certificates, making requests to access a resource, or both. An entity can, e.g., be a person or a program identified by a public key within a computer system. A **role name** represents access rights with respect to system resources, similar to Admin, Guest or Simple user in Windows operating system, granted by an entity. A **role** represents groups of entities (may be singletons) that have access rights related to a certain role name and granted by the role issuer. Credentials are statements in a RT language, which are used for describing access control policies, assigning entities to roles and delegating authority to the members of other roles.

In this paper, we use nouns beginning with a capital letter or just capital letters, e.g. A, B, C, to denote groups of entities. Role names are denoted as identifiers beginning with a small letter or just small letters, e.g. r, s, t. Roles take the form of a group of entities (role issuer) followed by a role name separated by a dot, e.g. $A.r$. A credential consists of a role, left arrow symbol and a valid role expression, e.g. $A.r \leftarrow B.s$.

BNF specification of the RT^{T} syntax can be written as follows.

```
<credential>        ::= <role> ← <role-expression>
<role>              ::= <entity-set> . <role-name>
<role-expression>   ::= <entity-set>
                      | <role>
                      | <role> . <role-name>
                      | <role> ∩ <role
                      | <role> ⊕ <role>
                      | <role> ⊗ <role>
```

There are six types of role expressions, according to this specification, and six types of credentials in RT^{T}, which are interpreted in the following way:

$A.r \leftarrow B$	– *simple membership*: a group of entities B can satisfy role $A.r$.
$A.r \leftarrow B.s$	– *simple inclusion*: role $A.r$ includes all members of role $B.s$. This is a delegation of authority over r from A to B, as B may cause new groups of entities to become members of role $A.r$ by issuing credentials that define $B.s$.
$A.r \leftarrow B.s.t$	– *linking inclusion*: role $A.r$ includes role $C.t$ for each C, which is a member of role $B.s$. This is a delegation of authority over r from A to all the members of role $B.s$.
$A.r \leftarrow B.s \cap C.t$	– *intersection inclusion*: role $A.r$ includes all the groups of entities that are members of both roles $B.s$ as well as $C.t$. This is a partial delegation from A to B and C.
$A.r \leftarrow B.s \oplus C.t$	– *role product*: $A.r$ is a manifold role that can be satisfied by a union set of one member of role $B.s$ and one member of role $C.t$. However, the same person can play both of these roles.
$A.r \leftarrow B.s \otimes C.t$	– *disjoint role product*: $A.r$ is a manifold role that can be satisfied by a union set of one member of role $B.s$ and one member of role $C.t$, where both members are disjoint.

This allows expressing separation of duties policy, in which different entities must hold the conflicting roles $B.s$ and $C.t$.

In case when the roles are identical, e.g. $A.r \leftarrow B.s \otimes B.s$, the credential can express threshold policy in which two (or more, if we use more credentials) members of $B.s$ can jointly fulfill $A.r$.

The syntax of a language describes the rules for constructing language expressions, such as credentials in RT^T. The semantics of a language describes the meaning of expressions in the application domain. Such a definition consists of two parts [12]: A definition of a semantic domain, which gives meaning to the language expressions, and a semantic mapping from the syntax to the semantic domain. A set theoretic semantics is the one that takes sets or power sets of entities as the semantic domain. If singleton roles are considered, the meaning of a role can be a set of entities that fulfill this role. If manifold roles are taken into account, the meaning of a role is a set of groups (sets) of entities that fulfill the role.

Let E be a set of entities and R be a set of role names. Denote the power set of entities by $F = 2^E$. Each element in F is a set of entities from E. Each element in 2^F is a set, composed of sets of entities from E. The semantic mapping can now be described as a function:

$$\hat{S} : 2^E \times R \to 2^F$$

that maps each role from $2^E \times R$ to a set of all such sets of entities, which are members of this role. Such a mapping from the set of RT^T roles to the power set of entities can be defined formally in the language of first-order logic as shown in Table 1, where A, B, C, X, Y are groups of entities, r, s, t are role names, $A.r$, $B.s$, $C.t$ are roles, and $\hat{S}(A.r)$ denotes the semantics of role $A.r$.

Table 1. The interpretation of first-order formulas in RT^T

RT^T credential	Meaning of the credential
$A.r \leftarrow B$	$B \in \hat{S}(A.r)$
$A.r \leftarrow B.s$	$(\forall x)\ (\ x \in \hat{S}(B.s) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s.t$	$(\forall x)\ (\forall y)\ (\ y \in \hat{S}(B.s) \wedge x \in \hat{S}(y.t) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \cap C.t$	$(\forall x)\ (\ x \in \hat{S}(B.s) \wedge x \in \hat{S}(C.t) \Rightarrow x \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \oplus C.t$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(C.t) \Rightarrow x \cup y \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s \otimes C.t$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(C.t) \wedge x \cap y = \phi \Rightarrow x \cup y \in \hat{S}(A.r)\ )$

SPKI/SDSI allows the first three types of credentials only. However, linking inclusion of arbitrary length is allowed. This does not increase the expressive power of the language, because a linking inclusion of arbitrary length, e.g. $A.r \leftarrow B.s...t.u$, can always be substituted by a pair of credentials $C.v \leftarrow B.s...t$ and $A.r \leftarrow C.v.u$. This way linking inclusion of length l can always be substituted by a set of l credentials with linking inclusions of length 2. Therefore, the complexity $O(n^3 l)$ of the algorithm for building the semantics of a given set of SPKI/SDSI credentials can, in fact, be considered as equal to $O(n^4)$ in terms of the number n of RT credentials.

The language RT_0 allows the first four types of credentials, and supports singleton roles only. The language RT^T allows all six types of credentials and supports manifold roles. The use of manifold roles is inevitable, because the members of roles defined by the last two credentials are always groups rather than single entities.

4 Motivating Example

Consider a student management system of a university composed of a set of nearly independent faculties. The system offers a set of services for the university and for the faculties, according to the concept of service-oriented architecture (SOA). The faculties administer particular instances of each service separately. No centralized security policy for the system exists. Instead, the service owner defines a security policy for each service independently.

The university, the library, each faculty and each student has a public key and is an entity, which can participate in issuing credentials and requesting services from the system. The university defined a role that reflected the university structure and issued the following set of credentials:

$\{\mathit{University}\}.\mathit{faculty} \leftarrow \{\mathit{IT}\}$	// faculty of Information Technology
$\{\mathit{University}\}.\mathit{faculty} \leftarrow \{\mathit{Chemistry}\}$	// faculty of Chemistry
....	// other faculties of the university

Each faculty of the university defined a set of roles that reflected the main actors of the didactic process and issued a set of credentials:

$\{\mathit{IT}\}.\mathit{student} \leftarrow \{A\}$	// A is an IT student
$\{\mathit{IT}\}.\mathit{teacher} \leftarrow \{X\}$	// X is an IT staff member
$\{\mathit{IT}\}.\mathit{supervisor} \leftarrow \{X\}$	// staff X can supervise students
....	// other entities within IT faculty

The following types of services were identified for the first release of this system.

1. Library Service. Offered full on-line access to the library resources. The owner of the service was the university, which applied the following security policy: Access was granted to all the students and teachers of all faculties of this university. The access control list for the service contained a role $\{\mathit{University}\}.\mathit{library}$. Security policy was described by a set of two credentials:

$\{\mathit{University}\}.\mathit{library} \leftarrow \{\mathit{University}\}.\mathit{faculty}.\mathit{student}$
$\{\mathit{University}\}.\mathit{library} \leftarrow \{\mathit{University}\}.\mathit{faculty}.\mathit{teacher}$

2. Grade Book. A complex service with two separate entry points: For students – to read the grades, and for teachers – to add new grades. Particular faculties of the university owned separate instances of this service.

(a) The service offered the requesting student read access to all the grades for the requester. Security policy for each instance of the service was defined by each owner. Chemistry decided that access was granted to the students only, and no delegation of the access rights was allowed. The access control list for the service contained a role $\{\mathit{Chemistry}\}.\mathit{gradeVisitor}$. Security policy was described by a single credential:

$\{\mathit{Chemistry}\}.\mathit{gradeVisitor} \leftarrow \{\mathit{Chemistry}\}.\mathit{student}$

IT faculty selected another policy. Access was granted to students, who could delegate permission to another people, and these people could pass the delegation again. Such a policy was described by the following credentials:

$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.student$
$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.\ gradeVisitor.friend$

IT student A could now delegate permission to read his or her grades to another person, e.g. B, by issuing a new credential:

$\{A\}.friend \leftarrow \{B\}$

Because a member of role $\{A\}.friend$ became also a member of $\{IT\}.gradeVisitor$, then B could pass the delegation again to C:

$\{B\}.friend \leftarrow \{C\}$

It is important to note that the permission for delegation of access rights was an individual decision of IT faculty, and other faculties could decide differently. Such a decision can be changed at any time by simply changing the set of existing credentials – in this case, by removing the credential:

$\{IT\}.\ gradeVisitor \leftarrow \{IT\}.\ gradeVisitor.friend.$

If IT faculty removes this credential, then it will implement the same policy as Chemistry. No need for removing all the student's "*friend*" credentials exists.

(b) Teacher's entry point represented in fact a set of services to manipulate grades received by students in particular courses. Only the teachers (one or more) assigned to a course could add or change a grade. Therefore, there was a separate access control list maintained for each course, which was identified by a course number *NN*.

IT faculty decided that an assigned teacher could delegate access to another teacher (an assistant), but the delegates could not pass the permissions again. Such a policy was implemented at IT for a course *NN* by an access control list that contained a role $\{IT\}.grade\_NN$ and the following set of credentials:

$\{IT\}.grade\_01 \leftarrow \{IT\}.teacher\_01$
$\{IT\}.grade\_01 \leftarrow \{IT\}.teacher\_01.assistant \cap \{IT\}.teacher$
$\{IT\}.teacher\_01 \leftarrow \{X\}$
.... // the same for courses 02, 03,...

IT teacher X assigned to a course number 01 could now delegate permission to manipulate grades in the course to another person, e.g. Y, by issuing a new credential:

$\{X\}.assistant \leftarrow \{Y\}$

The delegation was effective only when the delegate was an IT teacher. The delegate could not pass the permission to another person.

3. Supervisor Assignment. The service allowed registering assignment of a student to the selected supervisor. Separate instances of this service were owned by particular faculties of the university. Security policy for each instance of the service was defined by the owner. IT faculty decided that the assignment was registered if the student as well as the teacher agreed on this fact. The access control list for the service contained a manifold role $\{IT\}.assignment$. Security policy was described by a single credential:

$$\{IT\}.assignment \leftarrow \{IT\}.student \otimes \{IT\}.supervizor$$

Successful registration of student A to supervisor X resulted in adding the pair of entities $\{A, X\}$ to role $\{IT\}.superStudent$, which was done by issuing a new credential by the service:

$$\{IT\}.superStudent \leftarrow \{A, X\}$$

4. Course Registration. The service allowed a student to register for optional program. The owner of the service was a faculty, which defined the security policy for the service: The registration was valid when it was signed jointly by a supervisor and the assigned student. The access control list for the service contained manifold role $\{IT\}.superStudent$ issued by the previous service (Supervisor assignment).

5 System Architecture

An SOA system consists of a number of services that can be located within multiple separate systems from several business and administration domains, interconnected by a computer network. For example, particular instances of the services described in the previous section can be deployed to local servers, and administered by particular faculties of the university. Service clients, i.e. students and teachers, can invoke the services from remote, e.g. personal, computers.

Access rights to services established by the service owners can vary from one service instance to another. If the access rights are expressed through policies and described by sets of credentials, then those credentials must be stored somewhere in the network and presented for verification, at each invocation of a service. The verification of the access rights requires finding a credential chain, which confirms the membership of the requester in the role placed in the access control list of the service. This is a complex process, which can be performed by a special service, called trust management (TM) service, and invoked as part of the client's invocation. A general architecture of the system is shown in Fig. 1.

An important decision to make is to find the right place to store credentials. In practice, credentials can be stored by requesters, by issuers or in a known place in the network. SPKI/SDSI [5] as well as Akenti [6] assumes that credentials are presented by the application, i.e. by the invoked service in Fig. 1. One another possibility is to

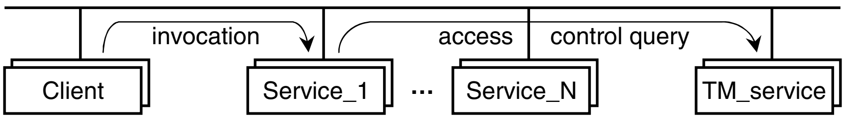


Fig. 1. Services and trust management servers in a SOA system

store at least part of credentials by a TM-service located in the administration domain. For example, each faculty can have a TM-service, which stores credentials issued by this faculty. However, credentials acting as personal certificates that define the attributes of particular entities can be stored by those entities, e.g., a credential:

$$\{IT\}.student \leftarrow \{A\}$$

can function as a student card, which is owned by the student. Credentials issued by particular entities, i.e. students and teachers in our example, to delegate permissions, can be stored by the subject entities.

After logging to a service, the client presents all the credentials, which are in his or her possession, and the service passes those credentials to the TM-service and asks for permission. TM-service builds the credential graph and resolves the query, using the presented credentials as well as the credentials stored in a local memory. TM-service can also look through the network in order to find other credentials.

Decentralized storage of credentials looks attractive, because it fits nicely into the general ideas of distribution and loose coupling that stand behind service-oriented architecture. However, the lack of control over the set of certificates exposes the system on a danger of inconsistency and raises the questions of certificate revocation, validity periods, etc. Neither of these questions can be answered by an analysis of the

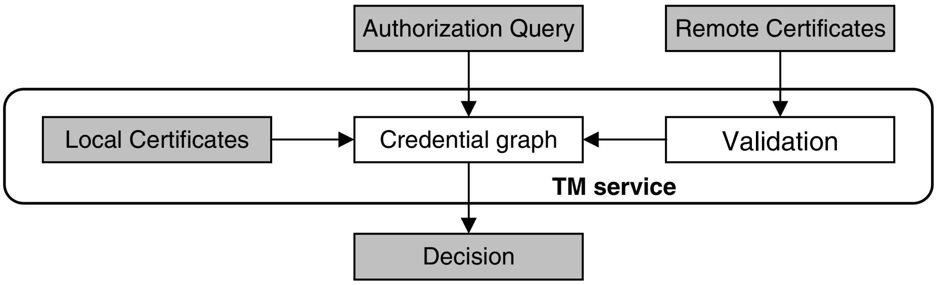


Fig. 2. The structure of a trust management (TM) service

credential graph. Therefore, the validity of certificates need to be assessed externally to the authorization logic [13]. Such a solution splits the process of resolving security queries into two separate layers, shown in Fig. 2, of the authorization logic, which performs a credential graph analysis, and the certificate validation, which searches through certificate revocation lists and verifies the validity periods with respect to the local time. This way, TM-service plays a part of a local certification authority, with the authority to decide, which credentials are taken into account, and which are not.

Single Sign-On. Consider another example adapted from [14]. There is a set of cooperating services (web sites), e.g. airline ticket reservation, car rental and hotel reservation. The users are identified on these sites by means of user accounts that contain access control data, such as user name and password, and other data, e.g. credit card numbers. When a user requests access to more than one service, the authentication of his or her identity can be done separately on each site, or on one site that asserts the user status to the other services. The latter possibility, which permits the user to enter one name and password in order to use multiple services, is called a single sign-on property of the access control system. It is convenient to the user, who need not repeat a login procedure many times in sequence. The implementation of a single sign-on is among the main goals of Security Assertion Markup Language (SAML 2.0), an OASIS standard [14], which defines a language and a protocol to create, request and pass the identity assertions between the interrelated services.

Trust management is quite another concept, in which the access control data is distributed among credentials, and no user accounts and no login procedure are used. Instead, a user must present a set of credentials when invoking a service. Single sign-on can mean in this case, that the user is not forced to resend the same certificates many times in sequence, when invoking more than one service. The existence of a trust management service can help in solving this problem. When a user accesses the first service, the necessary certificates are sent to TM service. The certificates and the credential graph that is built in the memory of TM service are valid for a predefined period of time, and can be used within this period to resolve queries issued by another services. For example, a teacher at IT, in the previous example, who gained access to the grade book, need not resend $\{IT\}.teacher \leftarrow \{X\}$ credential to access the library.

6 Local Certification Authority

Supervisor Assignment service, described in Section 4, acts as a centralized issuer of the membership credentials for role $\{IT\}.superStudent$, which controls access to Course Registration service. Each member of this role is a pair composed of a student and the assigned supervisor. A decentralized approach to supervisor assignment could relay on credentials issued by supervisors to students, without any contribution of the faculty. For example, if supervisor X agrees to supervise students A and B, then he or she may issue the following credentials to confirm the assignment:

$\{X\}.myStudent \leftarrow \{A\}$
$\{X\}.myStudent \leftarrow \{B\}$

The faculty may also decentralize decision-making on who can deputize X, if X is temporarily unable to perform supervisory duties. To do this, new role $\{X\}.supervisor$ can be introduced, with the membership defined by means of credentials, like:

$\{X\}.supervisor \leftarrow \{X\}$
$\{X\}.supervisor \leftarrow \{Y\}$

If a faculty accepts such a decentralized approach to supervisor assignment, then Supervisor Assignment service becomes useless, and role $\{IT\}.superStudent$, which controls access to Course Registration service, can be defined by a credential:

$\{IT\}.superStudent \leftarrow \{IT\}.supervisor.(supervisor \otimes myStudent)$

Table 2. The meaning of new credentials in extended RT^T

Extensions to RT^T	Meaning of the credential
$A.r \leftarrow B.s(t \oplus u)$	$(\forall x)\ (\forall y)\ (\forall z)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge z \in \hat{S}(x.u) \Rightarrow y \cup z \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s(t \otimes u)$	$(\forall x)\ (\forall y)\ (\forall z)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge z \in \hat{S}(x.u) \wedge y \cap z = \phi \Rightarrow y \cup z \in \hat{S}(A.r)\ )$
$A.r \leftarrow B.s(t \cap u)$	$(\forall x)\ (\forall y)\ (\ x \in \hat{S}(B.s) \wedge y \in \hat{S}(x.t) \wedge y \in \hat{S}(x.u) \Rightarrow y \in \hat{S}(A.r)\ )$

Credentials of this type, which join linking inclusion with other operators in a single role expression, do not exist in RT^T or any other Role based Trust management language. However, they can easily be added to the language with the semantics given by first-order formulae shown in Table 2.

The new types of credentials do not increase the expressive power of the language, however, they can help in reducing the number of roles and the number of credentials that are necessary to define a security policy. An equivalent definition of role $\{IT\}$*.superStudent* in RT^T requires introduction of a new role and two credentials:

$\{X\}$*superStudent* $\leftarrow$ $\{X\}$*.supervisor* $\otimes$ $\{X\}$*.myStudent*
$\{IT\}$*.superStudent* $\leftarrow$ $\{IT\}$*.supervisor.superStudent*

Decentralized approach to supervisor assignment raises a practical problem of resolving a conflict between two different teachers, say *X* and *Y*, who can independently agree to supervise a student, e.g. *A*, and issue credentials:

$\{X\}$*.myStudent* $\leftarrow$ $\{A\}$
$\{Y\}$*.myStudent* $\leftarrow$ $\{A\}$

Who of the two: *X* or *Y* is in this case responsible for signing course registration for student *A*? The problem can be solved if the supervisor assignment credentials are stored by TM-service (Fig. 1), which decides on the validity of credentials. Layered architecture of the access control mechanism resembles slightly a layered framework for modeling software and security policies introduced in [15].

7 Credential Graph

The semantics of a set *P* of credentials that define the security policy within a system can be represented by a credential graph. The nodes of this graph are role expressions, which are present in credentials, and the directed edges reflect inclusion of sets of groups of entities, which define the meaning of those expressions. Making a decision on the membership of a group *X* of entities in role *A.r* is equivalent to checking whether a path from *X* to *A.r* exists in the graph.

Let *P* be a set of extended RT^T credentials over a set *E* of entities and a set *R* of role names. A credential graph of *P* is defined in the following way.

Definition (Extended RT^T Credential Graph). Credential graph of a set *P* of extended RT^T credentials is a pair $G_P = (\ N_P, E_P\ )$ comprising a set N_P of nodes, which are role expressions that appear in credentials from *P* and groups of entities from *E*, and a set E_P of directed edges, which are ordered pairs of nodes from N_P. The sets N_P and E_P are the smallest sets that are closed with respect to the following properties:

1) If $A.r \leftarrow e$, where e is a role expression, belongs to P, then the nodes $A.r$ and e belong to N_P and a *credential edge* ($e, A.r$) belongs to E_P.
2) If role expressions $B.s.t$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ belongs to N_P and a path from X to $B.s$ exists in G_P, a *derived edge* ($X.t, B.s.t$) belongs to E_P.
3) If role expressions $B.s \cap C.t$, $B.s$, and $C.t$ belong to N_P, then for each $X \subseteq E$, such that paths from X to $B.s$ and from X to $C.t$ exist in G_P, a *derived edge* ($X, B.s \cap C.t$) belongs to E_P.
4) If role expressions $B.s \oplus C.t$, $B.s$ and $C.t$ belong to N_P, then for each $X, Y \subseteq E$, such that paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* ($X \cup Y, B.s \oplus C.t$) belongs to E_P.
5) If role expressions $B.s \otimes C.t$, $B.s$ and $C.t$ belong to N_P, then for each $X, Y \subseteq E$, such that $X \cap Y = \phi$ and paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* ($X \cup Y, B.s \otimes C.t$) belongs to E_P.
6) If role expressions $B.s.(t \oplus u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \oplus X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, then a *derived edge* ($X.t \oplus X.u, B.s.(t \oplus u)$) belongs to E_P.
7) If role expressions $B.s.(t \otimes u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \otimes X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, and a *derived edge* ($X.t \otimes X.u, B.s.(t \otimes u)$) belongs to E_P.
8) If role expressions $B.s.(t \cap u)$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ and $X.u$ belong to N_P, node $X.t \cap X.u$ belongs to N_P. If a path from X to $B.s$ exists in G_P, then a *derived edge* ($X.t \cap X.u, B.s.(t \cap u)$) belongs to E_P.

Credential graph G_P of a set P of credentials consists of nodes and edges. Part of the nodes can be defined by a static analysis of credentials from P. These nodes, called static nodes, are roles, which stand at the left hand side of symbol $\leftarrow$, and role expressions, which appear in credentials at the right hand side of symbol $\leftarrow$. Nodes that are added according to properties 6 through 8 are counted as static nodes. Other nodes are created dynamically in the process of building the graph, by repetitive scanning through the set of credentials and executing role expressions with operators $\oplus$ and $\otimes$. These additional nodes are union sets of entities, added to the graph according to properties 4 and 5. Credential edges (property 1) are defined statically, while derived edges (properties 2 through 8) are added dynamically.

One can note that dynamically added nodes can be connected directly only to role expressions of type $B.s \oplus C.t$, $B.s \otimes C.t$ and $B.s \cap C.t$. In order to enhance the efficiency of the graph building algorithm, the necessary search for paths within the graph will be restricted to a subgraph composed of static nodes and all the edges between these nodes. If a path from a node of type $B.s \oplus C.t$, $B.s \otimes C.t$ or $B.s \cap C.t$ to a

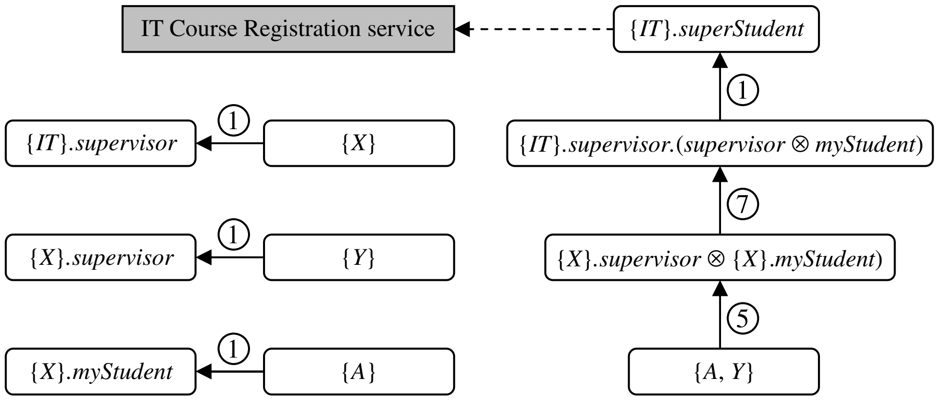

IT Course Registration service
{IT}.superStudent
①
{IT}.supervisor
{X}
{IT}.supervisor.(supervisor ⊗ myStudent)
⑦
{X}.supervisor
{Y}
{X}.supervisor ⊗ {X}.myStudent)
⑤
{X}.myStudent
{A}
{A, Y}

Fig. 3. Credential graph of the Course Registration service

certain node *N* is found, then paths from all groups of entities that are direct predecessors of this node to *N* are also considered.

Example. To observe the construction of a credential graph, assume that the credentials listed in Sections 4, excluding those related to Supervisor Assignment service, and the credentials listed in Section 6 have been issued, and an IT student *A* tries to register for an optional program.

To do this, *A* invokes Course Registration service of IT and presents a request signed jointly by *A* and *Y*. The access control list of the service contains a manifold role {*IT*}.*superStudent*, hence, the membership of pair {*A*, *Y*} in this role must be verified. The service calls TM-server, which looks through the accessible credentials and finds the following ones that are significant for resolving the query:

$\{IT\}.superStudent \leftarrow \{IT\}.supervisor.superStudent$
$\{IT\}.supervisor \leftarrow \{X\}$
$\{X\}.supervisor \leftarrow \{Y\}$
$\{X\}.myStudent \leftarrow \{A\}$

The authorization logic of the server builds a credential graph shown in Fig. 3. Small circled numbers placed near the edges of the graph refer to the numbers of properties in the definition of credential graph given above. After building the graph, TM-service verifies that a path from {*A*, *Y*} to {*IT*}.*superStudent* exists, and confirms authorization of *A* for registering for an optional course.

The complexity of the algorithm for building the credential graph of a set *P* of extended RT^T credentials can be evaluated with respect to the number of credentials in *P* (the cardinality of *P*), which is considered the input size of the problem. The method of evaluation is by assessing the complexity of each step of the algorithm and then counting the number of repetitions of particular steps. We assume that Dijkstra's algorithm is used for finding paths between two nodes in the graph [16]. The complexity of this algorithm is O(v^2), where v is the number of nodes.

Let n be the number of credentials in P and m be the number of role expressions other than roles and groups of entities in credentials in P. Obviously $m \leq n$. Moreover, let A, B, C, D, E, X, Y denote groups of entities from E.

The Algorithm (Creation of the Credential Graph)

1) For each credential $A.r \leftarrow e$ in P, add nodes $A.r$ and e to N_P and add an edge $(e, A.r)$ to E_P.

Remark. There are $2n$ nodes in the graph that has been built in step 1. The complexity of step 1 is of order O(n).

2) For each node $B.s(t \oplus u)$, $B.s(t \otimes u)$ and $B.s.(t \cap u)$ in N_P, if there exist a pair of nodes $X.t$ and $X.u$ in N_P, where X is an arbitrary group of entities, then add node $X.t \oplus X.u$, $X.t \otimes X.u$ or $X.t \cap X.u$, respectively, to N_P.

Remark. The number of nodes $B.s(t \oplus u)$, $B.s(t \otimes u)$, $B.s.(t \cap u)$ is not greater than m. Hence, a search through N_P in order to find pairs of nodes $X.t$ and $X.u$ is repeated at most m times. The complexity of step 2 is of order O(n^2).

The number of static nodes in N_P is not greater than $3n$.

Loop through the steps 3 through 6:

3) For each node $B.s.t$ find all the reverse paths (i.e. paths that start in $B.s$ and move along edges in the backward direction) from $B.s$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and role $X.t$ belongs to N_P, then add an edge $(X.t, B.s.t)$ to E_P.
   - If a reverse path exists from $B.s$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.y$ or $D.u \cap E.v$, then for each group X of entities, such that X is a direct predecessor of e and role $X.t$ belongs to N_P, add an edge $(X.t, B.s.t)$ to E_P.

Remark. The number of nodes $B.s$, which are the initial nodes in searching for paths, is not greater than the number m of role expressions $B.s.t$. Hence, the search for paths is repeated at most m times.

4) For each node $B.s \cap C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and from $C.t$ to X, then add an edge $(X, B.s \cap C.t)$ to E_P.
   - If a reverse paths exist from $B.s$ to e_1 and from $C.t$ to e_2, where e_1 as well as e_2 are or role expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all groups of entities X that are direct predecessors of e_1 as well as of e_2 and add an edge $(X, B.s \cap C.t)$ to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m$, hence, the search is repeated not more than $2m$ times.

5) For each node $B.s \oplus C.t$ and $B.s \otimes C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph. Select all the pairs of nodes e_1, e_2, such that

paths from $B.s$ to e_1 and from $C.t$ to e_2 exist, and e_1 as well as e_2 are groups of entities or role expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$.

In case of expression $B.s \otimes C.t$, in the following three points take into account only those pairs X, Y, for which $X \cap Y = \phi$.

- If both nodes e_1 and e_2 are groups X and Y of entities, then add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.
- If one node, e_1 or e_2, is a group X of entities, while the other node is an expression e, where e equals $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all the direct predecessors of e that are groups of entities. For each such group Y add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.
- If both nodes e_1 and e_2 are expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all pairs X, Y of the direct predecessors: X of e_1 and Y of e_2, which are groups of entities. For each of such pair add node $X \cup Y$ to N_P and add an edge $(X \cup Y, B.s \oplus C.t)$ or $(X \cup Y, B.s \otimes C.t)$, respectively, to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m$, hence, the search is repeated not more than $2m$ times.

6) For each node $B.s.(t \oplus u)$ and $B.s.(t \otimes u)$ find all the reverse paths from $B.s$ to the other nodes of the graph.
   - If a reverse path exists from $B.s$ to X and roles $X.t$ and $X.u$ exist in N_P, then add an edge ($X.t \oplus X.u$, $B.s.(t \oplus u)$) or ($X.t \otimes X.u$, $B.s.(t \otimes u)$), respectively, to E_P.
   - If a reverse path exists from $B.s$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.y$ or $D.u \cap E.v$, then for each group X of entities, such that X is a direct predecessor of e and roles $X.t$ and $X.u$ exist in N_P, add an edge ($X.t \oplus X.u$, $B.s.(t \oplus u)$) or ($X.t \otimes X.u$, $B.s.(t \otimes u)$), respectively, to E_P.

Remark. The number of nodes $B.s$, which are the initial nodes in searching for paths, is not greater than m, hence, the search is repeated not more than m times.

The number of static nodes is not greater than $3n$ in a graph that is searched for paths in steps 3 through 6 of the above algorithm. Therefore, the complexity of finding the paths that begin in a given node is of order $O(n^2)$. The total number of nodes, which are the initial nodes in searching for paths in steps 3 through 6 is also not greater than $2n$. Hence, the search is repeated not more than $2n$ times, and the complexity of a single pass through the loop (steps 3 through 6) is of order $O(n^3)$.

A single pass through the loop corresponds to a single search through the set of n credentials. Each pass adds edges to the static part of the graph. The possibility of adding an edge depends on the existence of certain paths in the graph, which means that it depends on the sequence in which the credentials are processed. Repeating the loop n times guaranties that all the possible sequences of credentials have been exercised. Therefore, the complexity of the entire algorithm is of order $O(n^4)$.

8 Conclusions

The main issue of public key infrastructure has been to provide secure means of authentication entities, based on cryptographic methods and techniques. The next step in developing an approach to the application security could be a research on politics and procedures for authorizing the entities. Trust management is an attempt to define security policies in a decentralized way, based on a delegation of authority. This paper describes a set of trust management languages, discusses their expressive power, suggests an extension to Role-based Trust Managements language RT^T and evaluates the complexity of algorithm that is used for answering security queries in RT^T.

Trust management languages SPKI/SDSI, RT_0, RT^T and extended RT^T are built upon the same set of basic operators for role membership, role inclusion, linking inclusion and role intersection. SPKI/SDSI allows linking inclusion of arbitrary length, while the other languages allow linking inclusion of length two. This is not a significant difference, because a credential with linking inclusion of length n can easily be converted into $n-1$ credentials with the length of linking inclusion not greater than two.

RT^T and extended RT^T support manifold roles and role product operators that are not present in the other languages. This is a significant difference, because the added features allow expressing threshold and separation of duties policies. Extended RT^T allows symmetrical superposition of the linking operator and the other operators of role intersection and role product. This does not increase the expressive power of RT^T and is a 'syntactic sugar' that can help in reducing the number of credentials and thus the size of the access control problem. The selected features of the four trust management languages are shown in Table 3.

Table 3. Roles and role expressions in the four trust management languages and SAML

Language	Roles	Manifold roles	Symmetric linking
SPKI/SDSI	no	no	no
RT_0	yes	no	no
RT^T	yes	yes	no
Extended RT^T	yes	yes	yes
SAML 2.0	no	no	no

The security queries are resolved in trust management languages by building a credential graph and searching for a path within this graph. The complexity of building the graph has been proved polynomial of order $O(n^4)$, with respect to the number n of credentials at hand. It is interesting to observe that the order of complexity is the same for extended RT^T credential graph and for much simpler language used in SPKI/SDSI environment.

Our plans for further research include implementation of a prototype of a trust management service outlined in Section 5.

Acknowledgments. This research was supported in part by the Ministry of Science and Higher Education under the grant number 5321/B/T02/2010/39.

References

1. A guide to understanding discretionary access control in trusted systems. National Computer Security Center, NCSC-TG-003, Maryland (1987)
2. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. IEEE Computer (2), 38–47 (1996)
3. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: 17th IEEE Symposium on Security and Privacy, pp. 164–173. IEEE Computer Society Press (1996)
4. Blaze, M., Feigenbaum, J., Ioannidis, J.: The KeyNote Trust Management System Version 2. Internet Society, Network Working Group, RFC 2704 (1999)
5. Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., Rivest, R.L.: Certificate chain discovery in SPKI/SDSI. J. Computer Security 9, 285–322 (2001)
6. Thompson, M.R., Essiari, A., Mudumbai, S.: Certificate-Based Authorization Policy in a PKI Environment. ACM Trans. Information and System Security 6(4), 566–588 (2003)
7. Li, N., Mitchell, J.: RT: A Role-Based Trust-Management Framework. In: 3rd DARPA Information Survivability Conference and Exposition, pp. 201–212. IEEE Computer Society Press (2003)
8. Li, N., Winsborough, W., Mitchell, J.: Distributed Credential Chain Discovery in Trust Management. J. Computer Security 1, 35–86 (2003)
9. Czenko, M., Etalle, S., Li, D., Winsborough, W.: An Introduction to the Role Based Trust Management Framework RT. In: Aldini, A., Gorrieri, R. (eds.) FOSAD 2007. LNCS, vol. 4677, pp. 246–281. Springer, Heidelberg (2007)
10. Felkner, A., Sacha, K.: The Semantics of Role-Based Trust Management Languages. In: 4th IFIP Central and East European Conference on Software Engineering Techniques, pp. 195–206 (2009)
11. Sacha, K.: Credential Chain Discovery in RT^T Trust Management Language. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2010. LNCS, vol. 6258, pp. 195–208. Springer, Heidelberg (2010)
12. Harel, D., Rumpe, B.: Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff. Weizmann Science Press of Israel, Jerusalem (2000)
13. Chapin, P., Skalka, C., Wang, X.: Authorization in Trust Management: Features and Foundations. ACM Comput. Survey 3, 1–48 (2008)
14. Ragouzis N. et al. (eds.) Security Assertion Markup Language (SAML) V2.0 Technical Overview. OASIS Committee Draft, March 2008. Document ID sstc-saml-tech-overview-2.0-cd-02 (2008),
    `http://www.oasis-open.org/committees/download.php/27819/`
15. Reith, M., Niu, J., Winsborough, W.: Engineering Trust Management into Software Models. In: International Workshop on Modeling in Software Engineering. IEEE Computer Society (2007)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press and McGraw-Hill (2001)

Ontology-Based Matching of Security Attributes for Personal Data Access in e-Health

Ioana Ciuciu[1], Brecht Claerhout[2], Louis Schilders[2], and Robert Meersman[1]

[1] Semantics Technology and Applications Research Laboratory, Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium
[2] Custodix
Kortrijksesteenweg 214 b3 Sint-Martens-Latem Belgium 9830
`{iciuciu,meersman}@vub.ac.be`
`{louis.schilders,brecht.claerhout}@custodix.com`

Abstract. This paper discusses an interoperability solution (tool) for the internal management of a policy decision engine located at the level of the authorization layer of a service oriented environment. The tool aims to support federated access control in the context of distributed architectures, in which a local authorization policy is not able to recognize all the attributes in the authorization decision requests. The approach is based on an ontology-based interoperation service (OBIS) whose role is to translate security attributes (name-value pairs) from local security vocabularies into the attributes recognized by the central (Master Policy Decision Point) vocabulary based on a security ontology and its domain-specific extensions which provides semantic reasoning services. The approach is validated in an e-Health scenario for the access of patient data for diabetes patient monitoring and disease management.

Keywords: Authentication, Authorization, Security Policies, Ontology, Ontology-based Data Matching, e-Health.

1 Introduction and Motivation

Among the challenges of the Trusted Architecture for Securely Shared Services (TAS^3)[1] project is the interoperability of different access control policies in the context of unified distributed architectures. In this setting, every stakeholder organization describes its authorization policy using its organization-specific vocabulary, and when a policy engine receives an authorization request containing unknown terms, it semantically matches these with the ones locally known by the authorization policy.

In our previous work we have proposed an ontology-based interoperation service (OBIS [1]) which calculates the matching of security concepts extracted from access requests and local authorization policies. This study extends OBIS by proposing a method for mapping the security attributes (name-value pairs specifying the subject, resource and action) corresponding to a local security vocabulary into security

[1] http://tas3.eu/

attributes recognized by the central Policy Decision Point (PDP) using a policy ontology and ontology-based matching strategies. The extension of OBIS is called OBIS Domain Mapper. The ontology is grounded in natural language, which enables individuals from different organizations to express their security policies in an intelligible way, thus enforcing the user-centricity.

The proposed method is illustrated on the ontological representation of XACML policies, but the approach applies to other policy languages.

The use case is created with one of the TAS[3] test beds, the Custodix Healthcare demonstrator [2].

The rest of the paper is organized as follows: Section II describes related work; Section III provides background information on the technology being used. The requirements and use case are presented in Section IV. Section V proposes a method for attribute mapping based on ontology-based data matching techniques. Section VI presents our conclusion and suggestions for future work.

2 Related Work

Several approaches exist which aim at resolving semantic access control.

The Semantic Access Control (SAC) Model [3] was specifically designed to enforce ABAC policies in heterogeneous and distributed environments. It maps policies to resources dynamically based on the semantics of policies and resources. The Semantic Access Control Enabler (SACE) [4] was developed to enforce Role-Based Access Control (RBAC) when accessing heterogeneous data from databases.

Verma [5] presents a semantic policy matchmaking for web service policies specified across multiple domains (e.g. security, privacy, trust). KAoS [6] is a semantic policy language and a framework for the specification, management and enforcement of policies within different security domains. A similar approach is presented in [7], concerned with the meaning of contexts to be used directly in an access control policy.

Several approaches propose [8,9,10] semantic reasoning services for policy management based on Semantic Web technologies.

Our approach is slightly different, proposing its own paradigm for semantic reasoning based on an ontology grounded in natural language and on ontology-based data matching strategies.

3 Background

In this section we provide relevant background knowledge related to our approach, namely the knowledge and constraints representation and the policy language used.

3.1 DOGMA Approach for Ontology Engineering

The common understanding of security policies in this study is based on the Delevoping Ontology Grounded Methodology and Applications (DOGMA, [11]). DOGMA is a formal ontology engineering framework applying the principles of database design methodology (NIAM/ORM2, [12]) to ontology engineering. DOGMA ontology is grounded in natural language and based on the *double articulation principle* [13], which makes the ontology two layered:

1. The *lexon* base layer, containing a set of simple binary facts, called lexons, which are expressed in semi-natural language;
2. The *commitment* layer that formally defines rules and constraints by which applications may make use of the lexons from the lexon base.

A lexon is defined as a quintuple $‹\gamma, t_1, r_1, r_2, t_2›$ representing a fact type. γ is a context identifier that points to a context where two terms, t_1, t_2 are originally defined and disambiguated. r_1, r_2 are two roles that characterize the relationship between t_1 and t_2. For example, *‹ABAC, Subject, performs, performed by, Action›* is a lexon which means "in the context of ABAC, a Subject performs and Action and an Action is performed by a Subject". Table 1 illustrates high level concepts of an ABAC (Attribute Based Access Control) policy represented with lexons.

Table 1. Lexon representation of Subject, Action and Target in the ABAC model

ABAC Policy			
Head term	**Role**	**Co-role**	**Tail term**
SecurityPolicy	controls	controlled by	Action
SecurityPolicy	has	of	Target
SecurityPolicy	written by	writes	Subject
Action	performed by	performs	Subject
Action	performed on	under	Resource

A commitment contains a constraint on a (set of) lexon(s). For instance, we can apply the cardinality constraint on the above lexon, – "only one value is allowed for the action attribute". The commitment language needs to be specified in a language such as OWL[2] or SDRule language [14].

The lexons together with the commitments can be further converted to RDF[3] and OWL in order to make the ontology processable by other applications and by widely adopted semantic reasoners (e.g. Pellet [15]).

3.2 XACML

The eXtensible Control Markup Language (XACML [16]) is an OASIS standard language and architecture for the expression and exchange of access control policies, decision requests and responses.

The XACML policy language is structured in three levels of elements: policyset, policy and rule. A policyset comprises a set of policysets and/or policies, a target, obligations and a policy combining algorithm identifier. A policy comprises a set of rules, a target, obligations and a rule combining algorithm identifier. Finally a rule comprises a

[2] http://www.w3.org/TR/owl-ref/

[3] http://www.w3.org/RDF/

condition, a target and an effect. The target component found in each element type identifies the set of *subjects*, *resources, actions* and *environments* to which it applies.

As illustrated in Table 1, a subject (e.g. physician) requests permission to perform an action (e.g. read) on a resource (e.g. medical diary). A rule is a mapping from a target to a decision, whose value can be either *Permit* or *Deny*. A rule combining algorithm is used to resolve conflicts among all the rules which are applicable and which have different effects.

An *attribute* is the basic unit of an XACML policy. Attributes are characteristics of the subject, resource, action or environment of the access request. An XACML access request therefore consists of a list of attributes-value pairs. The mappings in this study are done between attribute-value pairs in a request (at the level of the central PDP) to attribute-value pairs in the local authorization policy.

4 Requirements and Use Case

4.1 Health Information Network Requirements

The present study is done in the context of the TAS^3 authorization architecture. TAS^3 is a framework for protecting personal data in service oriented environments. It focuses on interoperability and aims to deliver a generic solution useful in a wide range of application domains. At the level of the authorization layer this translates into semantic support for different policy decision engines and policy languages.

TAS^3 primarily puts people into control over their personal data in a service oriented architecture. The e-Health pilot demonstrates how TAS^3 accomplishes this objective in the highly regulated e-health environment, where user centric personal data management translates into:

(1) The possibility for patients to adjust the default e-health domain policies determined by legislation and ethical guidelines, so that their personal data is protected according to their personal preferences on data protection;
   - For example: a patient should have the option not to disclose mental health related information from a replacement physician (out-of-office hours).
   - However, in a highly regulated environment such as healthcare, personal freedom to hide or disclose health information has its boundaries (e.g. where hiding it could result in bad treatment or damage the treating healthcare professionals). These need to be taken into account.

(2) The possibility for data users to query patients for specific (extraordinary) access requests for data processing (e-consent).
   - For example: a patient could be invited to share existing data into a clinical study.

4.2 Use Case: Federated Data Access

The demonstration environment (Fig. 1) was modeled according to the “distributed health repositories with central access” concept, which forms the basis of many e-health information sharing initiatives in the EU and the US. In particular, the use case was staged in a Belgian setting.

Central to the system is the Patient Information Location Service (PILS [2]) which is used by professionals (e.g. medical doctors, researchers) to find patient information in distributed repositories. Two types of repositories have been connected in this demonstrator: (1) hospital results servers and (2) summary record repositories, as illustrated in Fig. 1.

Multiple Identity Providers (IdP) exist in the trial, all of which are authoritative with respect to unique user identities and to unique healthcare professional identifiers (similar to the actual situation in Belgium). Finally, there is a privacy management center where patients can set access policies on their personal data.

In the privacy center, patients can specify their personal privacy preferences which are then to be enforced over the health information sharing network for health professionals. The preferences set by the individual patients are translated into XACML policies which are loaded into a central Policy Decision Point (PDP). Service providers participating in the health network are required to forward access requests to that central PDP (if they involve resources covered by the central PDP policies).

Apart from the differences in authorization frameworks, different service providers use different security vocabularies (attribute-value pairs describing the subject, resource and action). In the demonstrator, the OBIS Domain Mapper service instances are responsible for translating local security vocabularies (name value pairs) into the "domain" vocabulary, used in by the central PDP.

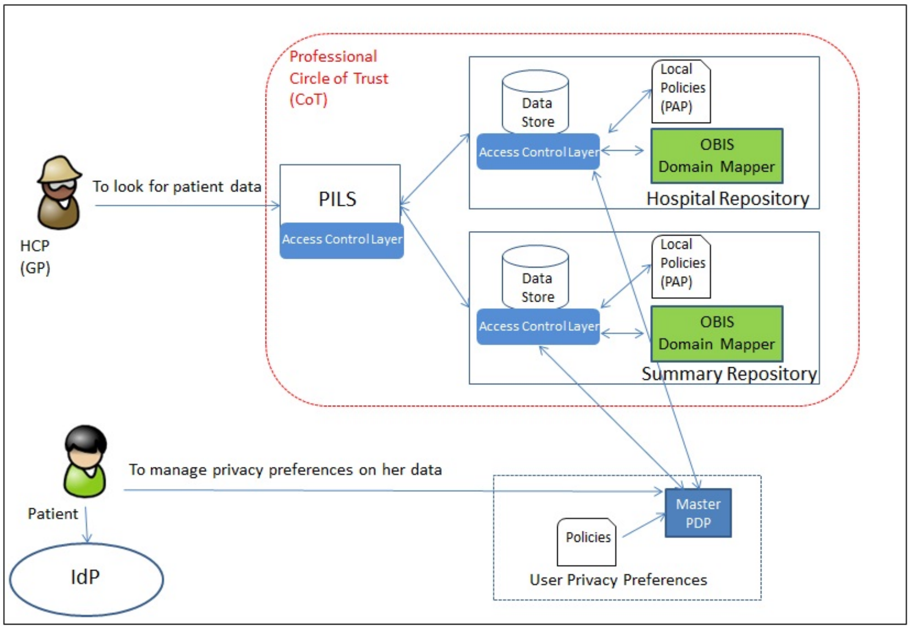


Fig. 1. Health Care use case scenario

Table 2 exemplifies request content of an access request to the central (Master) PDP in the first scenario.

Table 2. Example of content of an access request to the central PDP

Request			Decision
Subject	**Action**	**Resource**	
Subject_type = GP	Action_type = Read	Resource_type = labresult	Permit
Subject_type = GP	Action_type = Create	Resource_type = document	Permit
Any	Any	Any	Deny

5 Ontology-Based Attribute Matching for Access Control

Here we explain how we extend our previous ontology-based interoperation service to support mappings between the attributes in a decision request and the attributes from the access control rules in a local access policy. The method, the tools and the underlying technologies are presented.

5.1 Ontology-Based Interoperation Service (OBIS)

OBIS was initially designed as a web service located in the authorization architecture of TAS^3. It provides an interface to perform relation lookups between two terms represented as URIs, corresponding to the Service Requestor (SR) and to the Service Provider (SP) respectively, in order to determine the level of dominance between them.

Given e.g. a name of a resource, OBIS semantically infers the object class of the resource and computes how the authorization propagates in the (role/attribute/action) inheritance hierarchy, while enforcing the constraints in the ontological commitments.

This study proposes an extension of the OBIS service, called OBIS Domain Mapper, which translates the security attributes in the local PDPs into attributes recognized by the central (master) PDP in the TAS^3 authorization architecture. The main difference between the original OBIS service and the one proposed in this paper is that the first one only returns a code indicating the domination relation between two security concepts originating from different policy languages, while with the second approach the mapping between the two security domains (languages) is also provided. This approach is described in the next section.

5.2 Security Attributes Mapping

A method is proposed here for the mapping of security attributes using the Ω-RIDL Mapping Generator tool [17,18]. Ω-RIDL takes in input an XML file representing the access decision request and an ontology file (lexons and commitments) representing the access control policy ontology and returns a Ω-RIDL mapping file which maps the security attributes to concepts in the ontology.

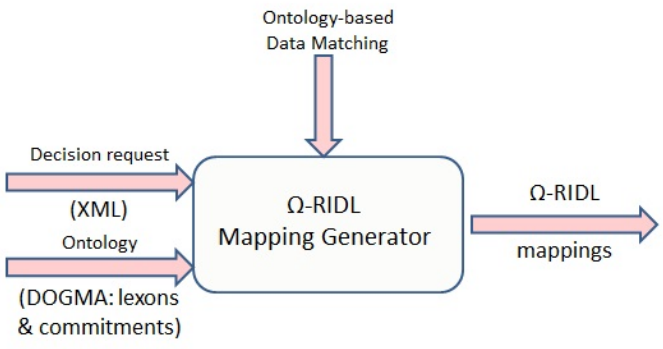

Ontology-based Data Matching
Decision request (XML)
Ontology (DOGMA: lexons & commitments)
Ω-RIDL Mapping Generator
Ω-RIDL mappings

Fig. 2. Ω-RIDL mapping generator architecture

The Ω-RIDL mappings are obtained by applying ontology-based data matching strategies at (1) string level (fuzzy literal similarity, e.g. JaroWinkler); (2) lexical level (synonymous similarity, e.g. based on WordNet[4]) and (3) ontology (lexon graph) level (semantic similarity) in this order (refer to [19] for details on ontology-based data matching strategies).

Ω-RIDL is designed as a web service which is called by the OBIS Domain Mapper service in order to infer the mapping of security attributes between two domains. OBIS Domain Mapper sends a bag of security attributes (name-value pairs representing the subject, resource and action) corresponding to a domain Dom1 in input to Ω-RIDL which performs semantic inference and ontology-based data matching operations and returns another bag of attributes corresponding to another domain, Dom2, as shown in Fig. 3. Previous to calling Ω-RIDL, OBIS performs an explicit translation (mapping) from the local terminology (Dom1, Dom2) to the core ontology (lexon graph), based on the user-defined dictionaries. Then Ω-RIDL performs semantic inference operations on the ontology graph in order to infer the mappings from Dom1 to Dom2. For the moment being we only consider one-to-one mapping of attribute-value pairs. The one-to-many and many-to-one mappings of attribute-value pairs are ongoing work.

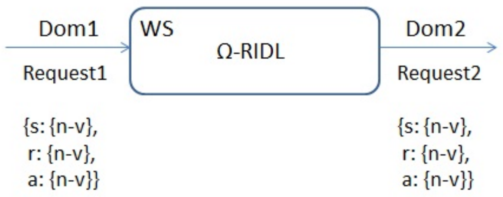


Fig. 3. Ω-RIDL invocation

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

Below we provide an example of a XACML policy rule that returns `Permit` for access requests that have value `physician` for attribute `subject`, value `read` for attribute `action` and value `summary information` for attribute `resource` (see example from Table 2).

```
<Target>
  <Resources>
    <Resource>
      <ResourceMatch
        MatchId="function:string-equal">
        <AttributeValue
          DataType="#string">summary_information
          </AttributeValue>
        <ResourceAttributeDesignator
          AttributeId="…information-class "
          DataType="#string" />
        </ResourceMatch>
      </Resource>
    </Resources>

  <Actions>
    <Action>
      <ActionMatch
        MatchId="function:string-equal">
        <AttributeValue
          DataType=="#string">read
        </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="…action"
        DataType="#string" />
      </ActionMatch>
    </Action>
  </Actions>
</Target>

<Rule RuleId="SenderIsPhysician" Effect="Permit">
  <Description>Physicians can create new documents
  </Description>
  <Condition>
    <Apply>
      FunctionId="function:string-is-in"
      <AttributeValue>
        DataType="#string">physician
      </AttributeValue>
      <SubjectAttributeDesignator
        DataType="#string"
        AttributeId="hcp-type"/>
    </Apply>
  </Condition>
</Rule>
```

When a physician tries to view a `vaccination fiche` in one of the repositories through the health information sharing infrastructure, the following happens: Inside the contacted repository an access control request for a `read` action on a `vaccination fiche` (resource) is triggered (to eventually determine if the `vaccination fiche` can be shown to the physician). This request is to be evaluated by a local access control decision engine according to locally formulated policies (which do e.g. also deal with access rules for "locally originated" requests). However, for this type of access through the health information network, also the central PDP needs to be queried (access control decisions by different engines are eventually to be combined).

The security vocabulary used in the local repository (which is typically implementation specific) is not aligned with the more generic security vocabulary used in the wider health domain (used in the policies handled by the central PDP). The local access control request can thus not be evaluated as such by the central PDP.

A generic approach to translation of access requests from one domain vocabulary to another is provided by the OBIS Domain Mapper. This component translates attributes (e.g. XACML name value pairs) from one security domain to another (here from a "local" repository into the vocabulary used as reference in the distributed environment).

More specifically, in the described example, the OBIS Domain Mapper will look into the ontology hierarchy and constraints via the Ω-RIDL mappings and will infer that a `vaccination fiche` document (as known in the repository) classifies as `summary information` according to the central policy vocabulary.

5.3 Access Control Policy Ontology

Fig. 4 illustrates the access policy ontology used in the e-Health scenario. The focus in the figure is on the "target" concept, showing its constituents hierarchically (see the circles). The semantic relations are of the type 'part-of' and 'is-a' (which grow or shrink a set), indicating the (security-specific) domination relation between the concepts. OBIS computes the domination relation between two concepts in the ontology using AND/OR graphs. The figure includes core concepts from the TAS[3] security ontology (e.g. 'subject', 'action', 'resource') linked to application-specific concepts derived from the e-Health scenario (e.g. 'patient-id', 'hcp-type').

Every component in the above described scenario commits to this ontology. Every stakeholder organization (or department), must provide a mapping file between its own terminology and the core ontology. This task is the responsibility of the security officer of every participant organization. The mapping files will serve to translate the local concepts to central ones as a preliminary step before performing the ontology-based data matching (inference) with Ω-RIDL.

Table 3 shows mappings between the local policies of the Hospital data repository and the central ontology. The first mapping concerns a subject mapping from the local hospital repository `A`, where the subject attributes are expressed as `name = employee_type` and `value = nurse`, to the central vocabulary used by the Master PDP, where `name(employee_type)` maps to `hcp-type` and `value(nurse)` maps to `nurse`.

The second row shows a resource attribute mapping from the local vocabulary of hospital repository `A`, where the resource `name = file_type` and the resource `value = vaccination type`, to the central vocabulary where `name` maps to `document-type` and `value` maps to `summary information`.

Table 3. Mappings between the local (organization-specific) terminology and the core ontology

Repository	Term	Concept in the ontology
Hosp. RepositoryA	employee_type = nurse	hcp-type = nurse
Hosp. RepositoryA	file_type = vaccination fiche	document-type = summary information
HIV Center	Doc-type = medication fiche	Sensitivity-indicator = HIV
	Location = Brussels	
	Patient-Name = Herve	

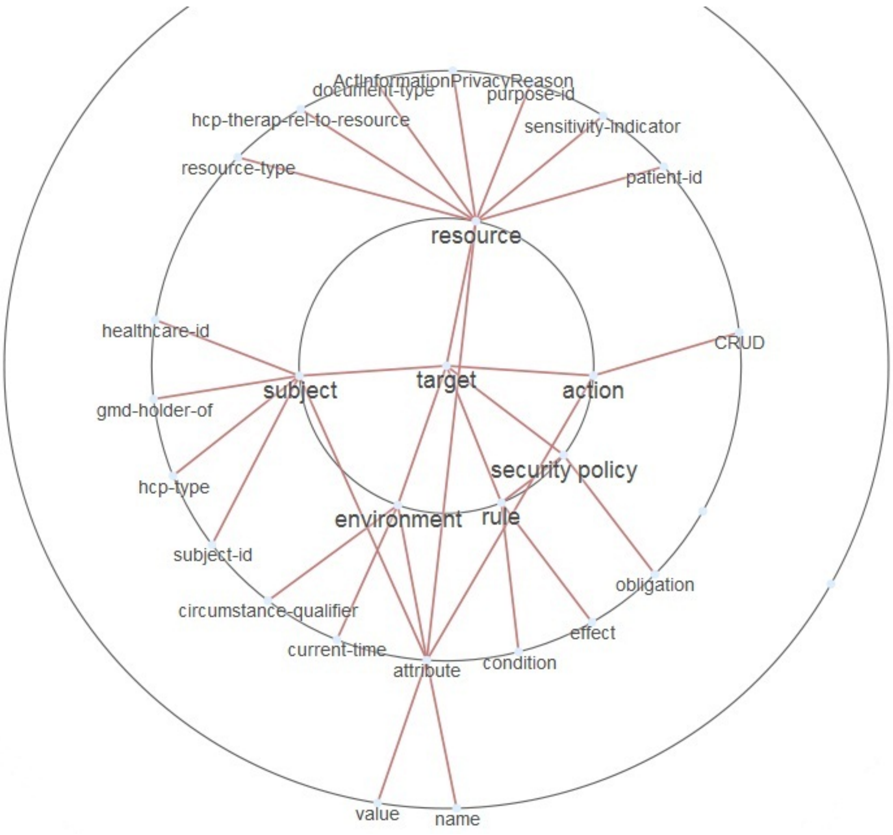


Fig. 4. The “Target” concept in the access control policy ontology

6 Conclusion and Future Work

The paper presents an extension, OBIS Domain Mapper, of a previously proposed ontology-based interoperation service which enables attribute mappings between local and central security vocabularies for the internal management of a policy decision engine in the context of service oriented architectures. The approach is based on the DOGMA ontology which enables semantic reasoning and on the Ω-RIDL mapping generator which performs the mapping based on ontology-based data matching strategies.

The OBIS service is (1) user-centric, enabling end-users to manage and protect their personal data through the creation of control access policies, without needing to know specific details about the security domains of remote service providers; (2) based on a security policy ontology grounded in natural language; (3) automated, through an integrated architecture which ensures OBIS is called by credential validation services and policy decision points; (4) autonomous, being designed as a web service to operate in an open, distributed and dynamic environment; and (5) secure, enabling query-only requests via SSL/TLS links.

Future work will involve the implementation of more sophisticated Ω-RIDL mappings by introducing additional constraints and evolving the ontology with more sophisticated authorization policies, including concepts such as obligations, delegation of authority, and separation of duty. The evaluation of the results with the ODMF (Ontology-based Data Matching Framework) evaluation benchmark is also planned as future work.

Additional functions, which extend the xacml rule engine by reasoning functions, are ongoing work.

Acknowledgments. This paper is supported by the EC FP7 TAS^3 (Trusted Architecture for Securely Shared Services) project. The authors would like to thank all TAS^3 project partners for their contribution to the research.

References

1. Ciuciu, I., Zhao, G., Chadwick, D.W., Reul, Q., Meersman, R., Vasquez, C., Hibbert, M., Winfield, S., Kirkham, T.: Ontology-based Interoperation for Securely Shared Services. In: Proc. IEEE Int. Conf. on New Technologies, Mobility and Security (NTMS 2011), Paris, France (2011)
2. Claerhout, B., Carlton, D., Kunst, C., Polman, L., Pruis, D., Schilders, L., Winfield, S.: Pilots Specifications and Use Case Scenarios, TAS^3, Deliverable D9.1, Trusted Architecture for Securely Shared Services (2010), `http://tas3.eu/`
3. Yague, M., Gallardo, M., Mana, A.: Semantic access control model: a formal specification. In: Proc. 10th European Symposium on Research in Computer Security, pp. 23–24 (2005)
4. Mitra, P., Liu, P.: Semantic access control for information interoperation. In: Proc. 11th ACM Symposium on Access Control Models and Technologies, pp. 237–246 (2006)
5. Verma, K., Akkiraju, R., Goodwin, R.: Semantic matching of web service policies. In: Proc. 2nd Int. Workshop on Semantic and Dynamic Web Processes, pp. 79–90 (2005)

6. Uszok, A., Bradshaw, J.M., Lott, J., Breedy, M.R., Bunch, L., Feltovich, P.J., Johnson, M., Jung, H.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS. In: Proc. IEEE Workshop on Policies for Distributed Systems and Networks, pp. 145–152 (2008)
7. Dersingh, A., Liscano, R., Jost, A., Finnson, J., Senthilnathan, R.: Utilizing semantic knowledge for access control in pervasive and ubiquitous systems. Mobile Netw. Appl. 15, 267–282 (2010)
8. Damiani, E., De Capitani di Vimercati, S., Fugazza, C., Samarati, P.: Extending Policy Languages to the Semantic Web. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 330–343. Springer, Heidelberg (2004)
9. Smith, M., Schain, A., Clark, K., Griffey, A., Kolovski, V.: Mother, May I? OWL-based Policy Management at NASA. In: OWLED (2007)
10. Ferrini, R., Bertino, E.: Supporting RBAC with XACML+OWL. In: SACMAT, pp. 145–154 (2009)
11. Spyns, P., Tang, Y., Meersman, R.: An Ontology Engineering Methodology for DOGMA. J. of App. Ontology 3(1-2), 13–39 (2008)
12. Halpin, T.: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. Morgan Kaufmann, San Francisco (2001)
13. Spyns, P., Meersman, R., Jarrar, M.: Data Modeling Versus Ontology Engineering. SIGMOD Record: Special Issue on Semantic Web and Data Management 31(4) (2002)
14. Tang, Y., Meersman, R.: SDRule Markup Language: Towards Modeling and Interchanging Ontological Commitments for Semantic Decision Making. In: Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. IGI Publishing, USA (2009) ISBN: 1-60566-402-2
15. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoned. J. of Web Semantics (2007)
16. OASIS "eXtensible Access Control Markup Language" (XACML) Version 2.0 OASIS Standard (2005)
17. Trog, D., Tang, Y., Meersman, R.: Towards Ontological Commitments with Ω-RIDL Markup Language. In: Ontologies, Databases and Applications of Semantics, Villamoura, Portugal (2007)
18. Verheyden, P., De Bo, J., Meersman, R.: Semantically Unlocking Database Content Through Ontology-Based Mediation. In: Bussler, C.J., Tannen, V., Fundulaki, I. (eds.) SWDB 2004. LNCS, vol. 3372, pp. 109–126. Springer, Heidelberg (2005)
19. Tang, Y., De Baer, P., Zhao, G., Meersman, R., Pudkey, K.: Towards a Pattern-Driven Topical Ontology Modeling Methodology in Elderly Care Homes. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 514–523. Springer, Heidelberg (2009)

A Unified Ontology for the Virtualization Domain*

Jacopo Silvestro, Daniele Canavese, Emanuele Cesena, and Paolo Smiraglia

Politecnico di Torino, Dip. di Automatica e Informatica, Italy
first.last@polito.it

Abstract. This paper presents an ontology of virtual appliances and networks along with an ontology-based approach for the automatic assessment of a virtualized computer network configuration. The ontology is inspired by the Libvirt XML format, based on the formal logic structures provided by the OWL language and enriched with logic rules expressed in SWRL. It can be used as a general taxonomy of the virtualized resources. We demonstrate the validity of our solution by showing the results of several analyses performed on a test system using a standard OWL-DL reasoner.

1 Introduction

The emerging paradigm of cloud computing is grounded on a massive use of virtual appliances. To automatize the management of a virtualized system it is necessary to describe it in a coherent and centralized way [1].

In the last decade, Libvirt has emerged as a hypervisor-agnostic solution for the configuration of a virtual environment. Libvirt is considered the standard de facto solution and it is used by most cloud management platforms, including OpenStack[1] and OpenNebula[2] (see also [2] for a detailed discussion).

Within Libvirt, all virtual appliance settings are described by using XML and this information is stored locally on the physical host. Whenever many hosts are in place, e.g., in a cloud data center, each host only has a partial view of the whole environment, limited to the virtual appliances it owns. Cloud management platforms usually devote a management node to keep the "big picture" and to guarantee that the configuration of each host is consistent with the others.

In this setting, a common knowledge base to collect data from all the physical hosts and to enable system verification and analysis is of primary importance. An ontology is more appropriate than XML as it allows automatic reasoning and enables reuse of virtual machine knowledge [3], so ontologies describing different virtual machines can be easily integrated.

* The extended version of the paper is available at http://security.polito.it/ontology

1 http://www.openstack.org

2 http://www.opennebula.org

Nowadays, an ontology able to describe a virtual environment is apparently not yet available. In [4], Youseff et al. proposed an ontology for the entire cloud computing environment, mentioning virtualization in their *Software Kernel* layer at an high level of abstraction, thus lacking details. Dastjerdi et al. [5] proposed an ontology based on the *Open Virtualization Format* (OVF) format [6]. The focus of their work is very similar to ours, however their description contains few details about the actual ontology and it is not publicly available.

We propose an ontology that describes the virtual appliances and the virtual networks inspired by the Libvirt XML format. The ontology concepts have been related in a coherent way, resorting to the formal logic structures provided by the OWL language and enriched with several logic rules expressed in SWRL.

The whole ontology consists of 272 classes, 74 object properties and 102 data properties. The most important class is `VirtualDomain`, which is related to other 18 classes and is referenced in 34% of the properties. The ontology is publicly available for download at `http://security.polito.it/ontology`, as well the extended version of this paper, which includes several additional scenario analyses performed by automatically importing into our ontology the Libvirt configurations through a custom tool.

While designing our ontology we considered using OVF as a basis, similarly to [5]. However, three considerations make us choose Libvirt. First, Libvirt is widely used and freely available, while OVF is only supported by few commercial products. Second, the semantics of the virtual objects is very similar in the two formats (in fact, tools exist to convert OVF into Libvirt format[3]). And finally, Libvirt, at least for our purposes, is more expressive than OVF, as it describes, for instance, CPU features. We are keeping track of several proposals to enhance the OVF standard and we will consider updating our ontology accordingly.

The remainder of the paper is structured as follows: in Section 2 we illustrate the application fields and the scope of our work, in Section 3 we provide a detailed description of our ontology, in Section 4 we propose several advanced uses of the ontology, and in Section 5 we summarize our findings and discuss future work.

2 Application Field and Scope of the Ontology

We propose a novel ontology for describing and analyzing virtualized environments, with the added benefit of using solely standard languages and reasoners.

Before starting the description of our ontology we introduce the terminology used through the remaining sections. The *hypervisor* is the core of a virtualization system; its role is to logically multiplex a number of physical resources. A hypervisor is a software running on a physical system called a *host*. A host can contain zero or more *domains*, or *virtual machines*, that are computing machines that use the virtualized components made available by a hypervisor. Within each domain, an operating system, the *guest OS*, runs.

Our ontology-driven approach is motivated by the need for a coherent description of both a virtual environment and the underlying physical system. It

[3] `http://thincrust.org/tooling-conversion.html`

allows the extraction of additional information, such as the topology of the existing virtual networks. Our ontology also enables a range of analyses, including the discovery of relationships between the physical hardware and virtualized elements. Knowing these relationships is particularly relevant for management purposes, to guarantee a proper allocation of the physical resources as well as to assess the effect of physical on the virtual world (cf., e.g., [1]).

To achieve these goals we have organized our ontology into three main categories, called *layers*. These are briefly introduced below and detailed in Section 3.

Starting from the Libvirt XML format, we have identified several classes which map the objects virtualized by a hypervisor. This brought us to the creation of the *virtual layer*, which contains the domain and virtual devices concepts.

One of the major limitations of the Libvirt XML format is the lack of a representation for the physical hardware. To bridge this gap, we have built the *physical layer* using the same base structure as its counterpart, the virtual layer. This layer is only intended to provide the minimum set of classes necessary for reasoning, and it is not a comprehensive ontology of the physical hardware.

During the ontology creation process we have frequently encountered several "concepts", e.g., bus addresses, that do not fit exactly into the virtual or physical worlds. For the sake of modularity, all these concepts were classified into an ad-hoc layer, the *logical layer*. Its heterogeneous content will be discussed further.

3 Description of the Ontology

In this section we discuss in detail the structure of our ontology and the relationships among its classes. Figure 1 presents a bird's eye view of the ontology using the UML class diagram notation.

Logical notions are represented by the descendants of the `Logical` class while the `Virtual` and `Physical` class hierarchies respectively contain all the virtual and physical related objects.

3.1 Logical Layer

Since both the virtual and the physical entities make an extensive use of logical notions, their description is provided first. The *logical layer* contains concepts used to describe several characteristics of physical and virtual resources together with elements used to establish relationships between the other two layers. This layer consists of several classes, but the most distinctive ones are discussed below.

The *actions*, represented by the `Action` class hierarchy, are entities used to define what the system must perform when a particular event occurs. On the other hand, the *mechanisms*, modeled by the `Mechanism` class hierarchy, specify how a particular goal should be accomplished. Both actions and mechanisms describe how to perform a job, but the first are event-driven, whereas the latter are event-independent.

The *features*, mapped to the `Feature` class hierarchy, are used to model the characteristics of a physical or virtual object. More specifically, features are

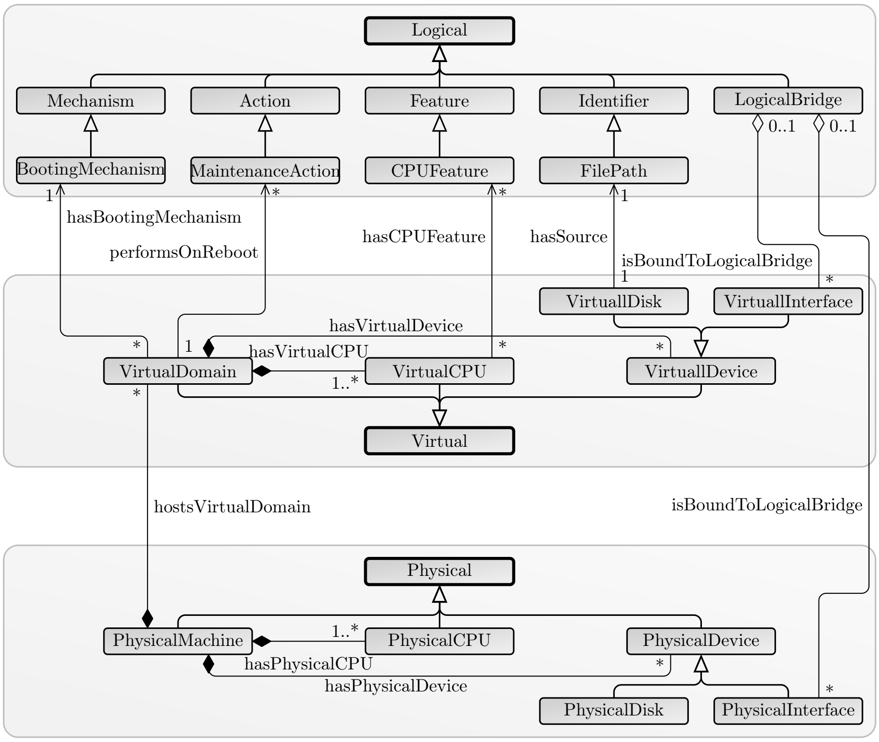


Fig. 1. The ontology UML class diagram

used to declare what a physical component supports and what a virtual entity requires. For example, the `CPUFeature` class is used to describe what a physical processor offers and, on the other hand, what a virtual processor needs.

The *identifiers*, depicted by the `Identifier` class hierarchy, are named tokens that uniquely designate an object. Figure 1 displays, as an exemplification, the `FilePath` class that models a file pathname, a concept used for several purposes, such as to specify an image file for a virtual disk.

The logical layer also consists of the `LogicalBridge` class, which represents the concept of *logical bridge*, an entity used to create virtual networks, linking physical and virtual NICs. Several objects can use the `isBoundToLogicalBridge` property to explicitly state their connection with a specific bridge.

Most of the virtual and physical classes are strongly dependent on the logical layer through several properties such as the `hasAddress` property that specifies the address owned by an entity, e.g., the IP address of a network interface. Furthermore, several virtual devices can have the `hasSource` and the `hasTarget` properties, which respectively define the data-source (e.g., a path to an image file) and the location where a guest OS will find this data (e.g., the `hda` disk).

For a more detailed discussion of the logical layer, see the extended paper available at `http://security.polito.it/ontology`.

3.2 Virtual Layer

The *virtual layer* consists of all the components that are virtualized by a generic hypervisor, i.e., the virtualized hardware and the domain concept.

One of the fundamental concepts is the notion of *domain*, that is represented by the `VirtualDomain` class. Since this class plays such a key role, a great number of properties can be attached to it, thus linking virtual machines to other virtual or physical resources, relying on the objects provided by the logical layer.

Every domain has a set of *virtual CPUs*, mapped to the `VirtualCPU` class hierarchy. Virtual CPUs can be described in detail, for instance using the property `hasCPUFeature` that uses the `CPUFeature` class individuals to define the requirements needed by a particular virtual processor.

Similarly to virtual CPUs, *virtual devices* are modeled by the `VirtualDevice` class hierarchy which represents all the virtual hardware with the exception of the processors. Virtual devices are virtual peripherals such as disks, sound and video cards, and everything that can be connected to a virtual bus (PCI, USB, and so on). The virtual device structure is vast and heterogeneous since every hardware is extremely specialized. Figure 1 displays the `VirtualInterface` class which models all the virtual NICs. It has several descendants that specialize the network interfaces, e.g., the `VirtualBridgeInterface` class represents a virtual NIC connected to a logical bridge. Figure 1 depicts also the `VirtualDisk` class which models the virtual disks, i.e., hard disks, CD-ROM and floppy drives. This class has a special property called `reachesSource` that relates a device to its real data source; in Section 4 we show how to automatically infer this property.

Another representative entity is the `VirtualPool` class, which describes the storage pools. A *storage pool* is a set of volumes, e.g., disk images, that can be used to implement several advanced storage techniques such as remote disks.

Since `VirtualDomain` is the core class of our ontology, we briefly introduce several properties that are used to connect it to other concepts. The `hasVirtualDevice` and `hasVirtualCPU` properties are used to specify a set of virtualized CPUs and devices of a domain. These properties also have a number of sub-properties like `hasVirtualInterface` and `hasVirtualDisk`, that respectively specify the virtual NICs and disks owned by a domain. Additionally, the `hasHypervisorType` property is used to refine the domain type, by defining the compatibility between a virtual machine and a specific hypervisor.

3.3 Physical Layer

The *physical layer* consists of all the tangible hardware. Libvirt does not provide a model of the real world, since this is outside its scope, but we felt that creating an ontology solely describing the virtual realm would not be very useful. In fact the virtualized hardware is strongly related to the real hardware, so the lack of a physical layer will severely limit the ontology usability. Therefore we created a hierarchical structure for the physical world, similar to the one described in Section 3.2 for the virtual layer. Since its internal architecture closely resembles the virtual layer one, we briefly discuss a selection of the most distinctive subclasses.

The PhysicalMachine class represents the *physical machines*, i.e., the hosts. It is the physical counterpart of VirtualDomain and, in a similar way, several object and data properties can be assigned to it in order to describe its configuration in detail, ranging from the owned hardware to the hosted virtual machines.

The PhysicalCPU class models the *physical CPUs*, i.e., the real processors. The supported features of the physical CPUs are specified using the property hasCPUFeature that points to the desired CPUFeature class individuals.

The PhysicalDevice class hierarchy maps all the *physical devices* apart from the processors. Figure 1 depicts the PhysicalDisk and PhysicalInterface classes which respectively represent the physical disks and NICs. The latter possesses an object property named isPhysicallyLinked that is used to represent a physical connection, e.g., a cable, between two physical network interfaces.

A physical machine can contain a set of physical processors and devices. This is described by the properties hasPhysicalCPU and hasPhysicalDevice which in turn have several sub-properties such as hasPhysicalInterface and hasPhysicalDisk, that are used to define the NICs and disks owned by a host. The physical machines have also the hostsVirtualDomain property which represents its hosted domains. This property is important because it links the physical layer to the virtual world.

4 Refinement of the Core Ontology

So far, we have discussed the structure of our ontology, the type of concepts it contains and their relations. Here we show how this ontology can be used to simplify the management of a virtual infrastructure and how the reasoning facilities provided by an OWL-DL reasoner can be exploited to this end.

The concepts defined in this section are included as children of the class ExampleViews in the ontology. All the listings use the Manchester OWL syntax.

For instance, we defined the concept MultiConnectedDomain (Listing 1.1) that gathers all the domains which have more than one network interface and the DomainWithRemoteDisk (Listing 1.2) which describes all the domains that are connected to a remote disk, i.e. a disk shared through a virtual pool.

```
VirtualDomain and (hasVirtualDevice min 2 VirtualInterface)
```

Listing 1.1. MultiConnectedDomain definition

We added a number of properties to increase the expressiveness of several concepts, such as the isLinked relation, which is as a symmetric and transitive super-property of isBoundToLogicalBridge. In this way we can assert that if a NIC is bound to a logical bridge then it is linked to that bridge and vice-versa and that all the NICs bounded to the same bridge are linked to each other.

```
VirtualDomain and (hasVirtualDevice some
    (VirtualDisk and (hasSource some
        (FilePath and ( inverse (hasTarget) some
            VirtualPool)))))
```

Listing 1.2. `DomainWithRemoteDisk` definition

By adding a property chain (Listing 1.3), we show how it is possible to infer the connection among domains starting from their description, which includes their virtual NICs, and the description of the logical bridges of a host.

```
hasVirtualInterface o isLinked o  inverse
   (hasVirtualInterface) ->
isConnectedToDomain
```

Listing 1.3. `isConnectedToDomain` property chain

Given this property chain, if there are two different domains each having a virtual NIC linked among them, then the two domains are connected. In this case we assume that each domain works as a hub, forwarding the traffic coming from each interface to all the others. Verifying such a property would require analyzing the internal network of each domain and since this task is not handled by Libvirt, this is out of the scope of this work. So we decided to be conservative from a security point of view, therefore considering this kind of connection possible.

By adding some rules in SWRL [7], it is possible to infer additional information. For instance, when considering a NAS, it is interesting to verify if the domain using it is able to access the network address to which the storage is attached. This can be achieved by defining the rule shown in Listing 1.4.

```
hostsVirtualDomain(?mac, ?dom), hasVirtualDisk(?dom,
   ?vdisk),  hasSource(?vdisk, ?file),  hasTarget(?pool,
   ?file), VirtualPool(?pool), hasSource(?pool, ?nas),
   hasAddress(?nas, ?ip), hasAddress(?pint2, ?ip),
   hasPhysicalInterface(?mac, ?pint1),
   isPhysicallyLinked(?pint1, ?pint2) ->
   reachesSource(?vdisk, ?pint2)
```

Listing 1.4. SWRL rule 2, `reachesSource` definition

The rule works in the following way. We consider a physical machine *mac* that hosts the domain *dom* which is using a remote disk *vdisk*. *vdisk* has as source the file with pathname *file* which is the target of a virtual pool *pool*. We check if *mac* is connected, through a network connection, to the machine *nas* which is the source of *pool*. We made some simplifications, since reaching the NIC of the machine to which the disk is attached is not sufficient to state that the disk is working properly, but it is a necessary condition. A more accurate analysis would require checking the internal configuration of the physical machine.

5 Conclusion and Future Work

To automatize the management of virtualized systems it is necessary to describe them in a centralized and univocal way, but such taxonomy is not yet available.

We proposed an ontology that is able to describe virtual appliances and networks inspired by the Libvirt format. The ontology concepts were related in a coherent way, resorting to the formal logic structures provided by the OWL language. We made the ontology freely available and we showed how to extend it by defining new concepts and rules, to enrich the taxonomy or perform more complex reasonings. Users can download our ontology and edit it as they like, or integrate it with any existing ontology through aligning and merging.

To demonstrate the effectiveness of our solution, we designed and implemented different scenarios related to network connectivity and remote storage. The results inferred by the reasoner match the behavior of the real system, showing that the proposed approach can effectively assist a system administrator in assessing the configuration of a virtualized environment.

In this work we made some assumptions about the physical infrastructure and the internal configuration of the domains and hosts, since this information is not handled by Libvirt. Future work will address the description of the entire physical network with an ontology and its relation to the one presented here.

References

1. Bolte, M., Sievers, M., Birkenheuer, G., Niehörster, O., Brinkmann, A.: Non-intrusive virtualization management using libvirt. In: Proc. of the Conference on Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, pp. 574–579 (2010)
2. Cerbelaud, D., Garg, S., Huylebroeck, J.: Opening the clouds: qualitative overview of the state-of-the-art open source VM-based cloud management platforms. In: Middleware 2009 Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Urbana Champaign, USA, pp. 22:1–22:8 (2009)
3. Noy, N.F., Mcguinness, D.L.: Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Sanford University (2001)
4. Youseff, L., Butrico, M., Silva, D.D.: Towards a Unified Ontology of Cloud Computing. In: Grid Computing Environments, GCE 2008, Austin, USA, pp. 1–10 (2008)
5. Dastjerdi, A.V., Tabatabaei, S.G.H., Buyya, R.: An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID 2010, Melbourne, Australia, Melbourne, Australia, pp. 104–112 (2010)
6. DMTF: Open Virtualization Format Specification. Technical report, DMTF (2010)
7. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, World Wide Web Consortium (2004)

2PSIM: Two Phase Service Identifying Method

Ali Nikravesh, Fereidoon Shams, Soodeh Farokhi, and Amir Ghaffari

Electrical and Computer Engineering Faculty
Shahid Beheshti University GC, Evin, Tehran, Iran
{a_nikravesh,f_shams}@sbu.ac.ir,
{so.farokhi,ar.ghaffari}@mail.sbu.ac.ir

Abstract. Service identification - as the first step of service-oriented modeling - holds the main emphasis on the modeling process and has a broad influence on the system development. Selecting appropriate service identification method is essential for prosperity of any service-oriented architecture project. Automation, utilizing middle-out strategy, and quality assess of services, are three important criteria in evaluation of service identification methods. Existing methods mostly ignore automation principles. Meanwhile a few automated and semi-automated methods use top-down strategy to identify services and ignore existing assets of enterprise. Moreover these methods do not take all the quality metrics into account. This paper proposes a novel semi-automated method called 2PSIM (Two-Phase Service Identification Method) which uses graph partitioning algorithm to identify services based on enterprise business processes as well business entity models. 2PSIM utilizes middle-out strategy and tries to identify reusable services with proper granularity and acceptable level of cohesion and coupling.

Keywords: Service-Oriented Architecture, Service-Oriented Modeling, Service Identification, Graph partitioning algorithms.

1 Introduction

Frequent changes in business environment and user demands are two important challenges in developing large-scale software systems. Service-Oriented Architecture (SOA) is one of the promising methods to address these challenges [1]. SOA embraces vast and enormous concepts and technologies. In this paper, we focus on the first phase of constructing service-oriented solutions, named service-oriented modeling. In the service-oriented modeling, the main emphasis is on the identification of the right architectural elements followed by their specification and realization [2].

In this phase, extant service identification methods should be evaluated and the most comprehensive one should be utilized by enterprise. Thus, the first goal of the current paper is to assess proposed academic and industrial methods for service identification and specify strength and weakness of each of them. Then, as the second goal, we will propose a novel service identification method based on the specified shortcomings.

In the current paper we have considered three important criteria to assess existing identification methods, namely: *automation*, *service quality evaluation*, and *delivery strategy*. Identification methods can be classified into three categories based on the automation point of view: prescriptive, semi-automated, and full-automated. Because SOA practically is utilized to develop large-scale software systems, hence, using prescriptive methods may lead to low quality of the identified architectural elements as well as being costly and timely. Moreover, their utilization efficiency depends on the architect's experiences [3]. On the other hand, full-automated methods suffer from lack of human supervision; therefore, they may lead to services with inappropriate granularity and low quality level.

In addition to automation issues, the proposed identification methods can be evaluated based on their delivery strategy. There are three delivery strategies to identify services: top-down, bottom-up, and middle-out [4]. In top-down strategy services are derived based on the analysis of business requirements. Bottom-up strategy focuses on derivation of services based on the existing assets of enterprise, and the middle-out strategy combines the other two strategies [5]. All of the full-automated and semi-automated methods use top-down strategy and ignore existing assets of enterprise in their identification process.

Another important criterion in assessing identification methods is quality of the identified services. Being autonomous, loose coupling, high cohesion, composability, discoverability, statelessness, and reusability are some important quality factors of an ideal business service [6], but in the service identification phase only some of them can be assessed, which are reusability, high cohesion, and low coupling [6].

Based on the mentioned points, we have set the following objective for the research reported in the present paper:

To propose a semi-automated method to identify business services which takes reusability, low coupling, and high cohesion into consideration, and utilizes middle-out strategy during identification process.

Existing methods only consider some aspects of our objective and none of them takes all of the specified criteria into account. To reach this objective, we have proposed a novel method, called 2PSIM, which uses graph partitioning algorithms to identify services. 2PSIM embraces a two phased process. In the first phase, it tries to identify services with the highest level of reusability by using business entity models of the enterprise. Because identified services in the first phase have fine granularity, in the second phase, 2PSIM tries to merge them together to identify composite services with high level of cohesion and low coupling.

The rest of this paper is organized as follows. Section two introduces some related work in the service identification area and evaluates them against our objective. In section three our method will be introduced, then it will be evaluated by performing a case study in the fourth section. Finally, the last section is dedicated to the conclusion of our work and the future works in this field.

2 Related Work

A great number of service identification methods have been proposed in the recent years. In this section we review some of the leading works in this field and analyze them against the objectives of our research, which are: *automation*, *identification strategy*, and *service quality evaluation*.

Table 1 represents the reviewed methods and their characteristics. In the following table each method has been evaluated against the mentioned criteria. Different values of the specified criteria are as follows:

- **Identification Strategy:** top-down(T), bottom-up(B), middle-out(M)
- **Quality Evaluation:** the method evaluates quality levels of the identified services (+), the method doesn't evaluates quality levels of the identified services (-)
- **Automation:** Prescriptive(-), semi-automated(+), fully-automated(++)

Table 1. Related Work

Name	Identification Strategy	Quality Evaluation	Automation
SOMA [4]	M	+	-
Kohlborn [5]	M	+	-
SOAD [7]	T	+	-
Amsden [8]	T	-	-
Ren [9]	T	-	-
MOGA-WSI [10]	T	+	+
Zhang [11]	T	+	+
Strosnider [12]	T	+	+
ASIM [13],[3]	T	+	++

3 The Proposed Method

Business service layer abstraction leads to the creation of two business service models: task-centric and entity-centric [6]. Although task-centric models are more popular than entity-centric ones, entity-centric services are more useful for creating highly reusable and business process-agnostic services [6]. On the other hand, focusing on business entities may lead to ignore semantic integrity of business elements [13]. Therefore, 2PSIM focuses on both entity-centric and task-centric models by running a two-phase process which is illustrated in Fig. 1. In the first phase, 2PSIM tries to identify entity-centric services by focusing on business entity models of the enterprise. We will refer to these services as Elementary Services (ES). The elementary services have high level of reusability as well as fine granularity.

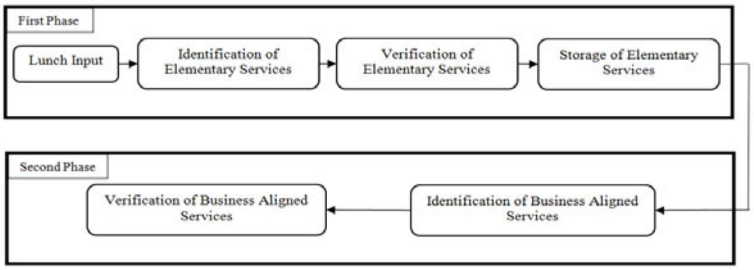

First Phase
Lunch Input
Identification of Elementary Services
Verification of Elementary Services
Storage of Elementary Services
Second Phase
Verification of Business Aligned Services
Identification of Business Aligned Services

Fig. 1. 2PSIM Process

Because fine granular services have inappropriate effects on performance and complexity of the final system, 2PSIM tries to merge elementary services together in the second phase of its process. In this phase the decision about how to compose elementary services will be made based on business process models. These composed services will be called as Business Aligned Services (BAS).

3.1 Lunch Input

2PSIM adopts business process model and business entity model as input. In order to identify entity-centric services, a model which demonstrates relationships between business entities and business activities is needed. 2PSIM uses the presented method in [14] to create the model. This method receives task-centric business processes and business entities as input and creates entity-centric business process model. After creating entity-centric business process a bipartite graph will be created which its vertices are business entities and business activities and each edge indicates a relationship between a business entity and a business activity. Fig 2 demonstrates this bipartite graph.

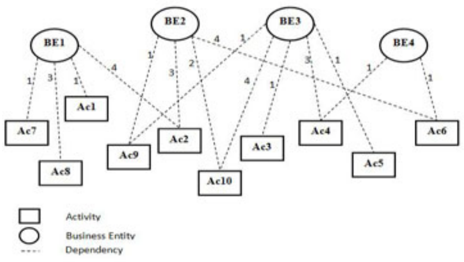


Fig. 2. The Bipartite Graph

3.2 Identification of Elementary Services

In order to identify entity-centric services, each business entity and its related activities should be exposed as an elementary service. Because elementary services should not be overlapped, each business activity should be only in one of the elementary services. Thus, we have assigned a weight ranging from one to four to each edge of the graph and edges with higher weights demonstrate more intense relationships between nodes of the graph. After specifying weight of each edge, 2PSIM identifies elementary services by running a partitioning algorithm on the bipartite graph. Fig 3 depicts the identified elementary services of the Fig 2. Details of elementary service identification algorithms fall beyond the scope of the current paper.

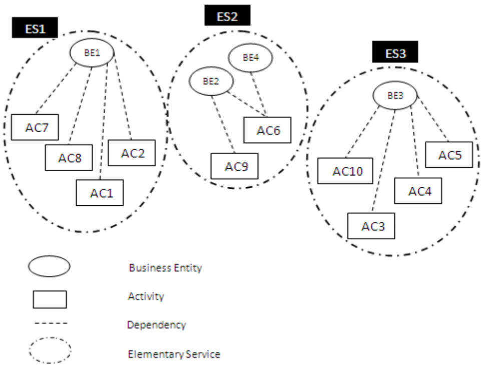


Fig. 3. Identified Elementary Services

3.3 Verification of the Elementary Services

After identifying elementary services, business analyst verifies each elementary service and assesses its business value. In addition, software architect evaluates each elementary service and verifies if the service can be mapped to existing assets of the enterprise.

3.4 Storage of Elementary Services

In the fourth and the final step of the first phase, identified elementary services will be added to a central service repository. Performing this step helps software architects to discover and reuse these services in the future service oriented projects.

3.5 Identification of Business Aligned Services

Since elementary services have fine granularity, utilizing them to develop software systems will have inappropriate effects on the performance and complexity of the final system. Therefore, in the second phase of 2PSIM, elementary services should be composed together. This composition is performed based on the enterprise business processes and the results are called “Business Aligned Services” (BAS). In order to identify BASs, a graph called “elementary service graph” is created. Its nodes represent elementary services that were identified in the first phase, and its edges demonstrate relationships between two elementary services. To discover relationships between elementary services, a new format of business process model is created, in which each business activity is substituted with its containing elementary service. This new format of business process model illustrates relationships between subsequent elementary services. Fig. 4 depicts two business processes and the new format of them.

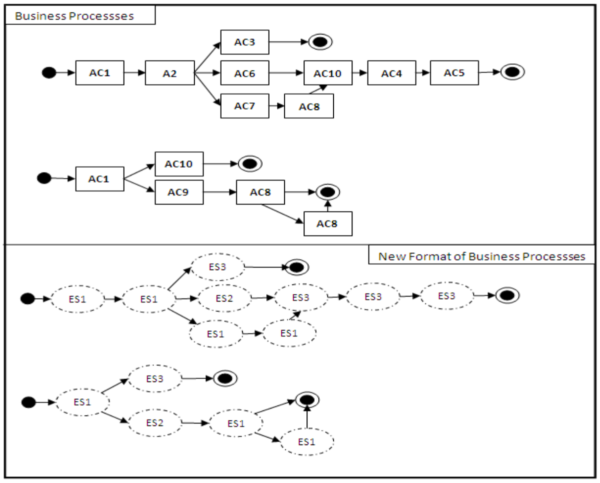


Fig. 4. New format of business process

It should be considered that relationships between elementary services in the new format of business process model do not have equal weights. We have considered the following formula to calculate cohesion of elementary services [15]:

a. If A and B be two consecutive elementary services then the cohesion value between them equals one unit.
b. If A and B are two elementary services that are connected via a condition in the business process model, then the cohesion value between them equals 1/(number of branches) unit.

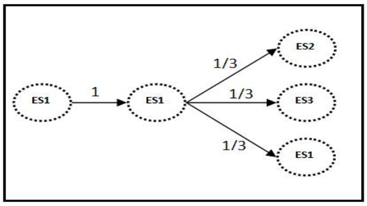


Fig. 5. Elementary services cohesion calculation

For example, in the depicted process of the Fig. 5,

a. ES_1 is connected to ES_2 via a condition with three branches, therefore the cohesion value between ES_1 and ES_2 is 1/3

b. ES_1 is connected to ES_3 via a condition with three branches, therefore the cohesion value between ES_1 and ES_3 is 1/3
c. ES_1 is connected to itself via a condition with three branches and a direct connection, therefore the cohesion value between ES_1 and ES_1 is 1+1/3= 4/3

By summing up all the cohesion values between elementary services, the total cohesion value between each pair of elementary services is assigned to the relevant edge of the elementary service graph. A sample elementary service graph is depicted in Fig. 6.

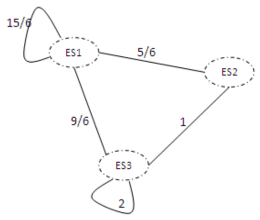


Fig. 6. A sample elementary service graph

Finally, 2PSIM partitions elementary service graph and promotes each partition as a business aligned service. 2PSIM tries to maximize the sum of intra-partition weights as well as minimizing the sum of inter-partition weights. Fig7 illustrates the identified business aligned services of the mentioned case study. The details of graph partitioning algorithm fall beyond the scope of this paper.

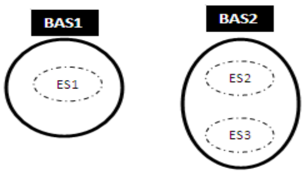


Fig. 7. Identified Business Aligned Services

3.6 Verification of the Business Aligned Services

After identification of business aligned services, in the final step of 2PSIM, software architect evaluates them and removes architectural incompetency. Furthermore software architect determines about how to fit each BAS to the existing enterprise applications.

4 Evaluation

In this section the users' evaluation of our method is presented. To this end we have taken advantage of Sol [16] methodology framework. Because the focus of this paper is on service-oriented modeling phase, the framework is used only for capability classification of different aspects of our proposed process, and then used these capabilities to design questionnaires for conducting users' evaluation of our process [3].

The users' evaluation of our method was collected through a survey with the aim of gathering independent evaluation of the method from its future potential users. The survey participants were asked about general capabilities of our proposed method.

We had 9 participants in our survey and all of them had rich experience in software development as well as suitable knowledge of service-oriented modeling. The questions had been organized into three categories regarding to inputs, outputs, and our identification process. The participants could answer a question based on a four-point scale ranging from (1) disagree, (2) neutral, (3) agree, to (4) strongly agree. In the following table, np expresses number of positive responses, i.e. responses 3 or 4, m represents average of the given grades and sd denotes the standard deviation.

Table 2. Experts' evaluation of the method

Scope of capabilities	Capabilities	1	2	3	4	np	M	sd
Input	Preparation of the inputs didn't have any difficulties	0	1	5	3	8	3.2	0.6
	The inputs of 2PSIM was enough for service identification	0	0	4	5	9	3.5	0.5
Process	2PSIM was practical in the scope of enterprise	1	2	4	2	6	2.6	0.7
	2PSIM was simple	0	3	3	3	6	3	0.8
	2PSIM was flexible	0	2	6	1	7	2.8	0.4
	Using the method facilitated service identification	0	0	3	6	9	3.6	0.4
	The method had a suitable level of automation	0	2	4	3	7	3.1	0.7
Result	The identified services were acceptable in the cohesion point of view	0	0	4	5	9	3.5	0.5
	The identified services were acceptable in the coupling point of view	0	1	4	4	8	3.3	0.6
	The identified services were acceptable in the granularity point of view	1	3	4	1	5	2.4	0.7
	The identified services were reusable	0	2	4	3	7	3.1	0.7
	The identified services had acceptable map to existing enterprise assets	0	0	5	4	9	3.4	0.4

5 Conclusion and Future Work

In this paper a novel service identification method called 2PSIM was proposed. This method adopts business process and business entity models of an enterprise as input and identifies services by running graph partitioning algorithms. Usability of 2PSIM was evaluated through applying it to development of a real world software system. Furthermore, experts' evaluation of 2PSIM and its eminency over existing methods have been gathered through a survey.

Refining the method and strengthen it, is our main future work. We aim to excel our algorithm in order to identify services with better quality level as well as improving their granularity. Moreover, extending 2PSIM to cover service specification and realization as the succeeding steps of service modeling phase, is considered to be our future work.

Acknowledgement. This work was supported by Iran Telecommunication Research Center under Contract No. 20294/500.

References

1. Sunate, K., Minseong, K., Sooyang, P.: Service Identification Using Goal and Scenario in Service Oriented Architecture. In: 15th Asia Pacific Software Engineering Conference (2008)
2. Bieberstein, N., Laird, R.G., Jones, K., Mitra, T.: Executing SOA: a practical guide for the service-oriented architect. IBM Press (2008)
3. Jamshidi, P., Sharifi, M., Mansour, S.: To Establish Enterprise Service Model from Enterprise Business Model. In: Proc. of IEEE International Conference on Services Computing (2008)
4. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., Holley, H.: SOMA: A method for developing service-oriented solutions. IBM Systems Journal 47(3) (2008)
5. Kohlborn, T., Korthaus, A., Chan, T., Rosemann, T.: Identification and Analysis of Business and Software Services – A Consolidated Approach. IEEE Transactions on Service Computing 2(1), 50–64 (2009)
6. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design, 5th edn. Prentice Hall PTR (2005)
7. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design. IBM developerWorks (2004), `http://www.ibm.com/developerworks/webservices/library/ws-soad1/index.html`
8. Amsden, J.: Modeling SOA: Part 1. Service identification. IBM developerWorks (2007)
9. Ren, M., Wang, Y.: Rule Based Business Service Identification Using UML analysis. In: IEEE International Conference on Information Management and Engineering (ICIME 2010) (2010)
10. Jain, H., Zhao, H., Chinta, N.R.: A Spanning Tree Based Approach to Identifying Web Services. International Journal of Web Services Research 1(1), 1–20 (2004)
11. Zhang, L.J., Zhou, N., Chee, Y.M., Jalaldeen, A., Ponnalagu, K., Sindhgatta, R.R., Arsanjani, A., Bernardini, F.: SOMA-ME: A platform for the model-driven design of SOA solutions. IBM Systems Journal 47(3) (2008)
12. Strosnider, J.K., Nandi, P., Kumaran, S., Ghosh, S., Arsanjani, A.: Model-driven synthesis of SOA solutions. IBM Systems Journal 47(3) (2008)
13. Jamshidi, P., Mansour, S.: ASIM: Toward Automatic Transformation of Enterprise Business Model to Service Model. IEEE Transactions on Service Computing (under review)

14. Kumaran, S., Liu, R., Wu, F.Y.: On the Duality of Information-Centric and Activity-Centric Models of Business Processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
15. Khoshkbarforoushha, A., Jamshidi, P., Nikravesh, A., Khoshnevis, S., Shams, F.: A Metric for Measuring BPEL Process Context-Independency. In: IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2009) (2009)
16. Sol, H.G: Information System Development: A Problem Solving Approach. In: International Symposiom on System Development Methodologies (1990)

Automated Statistical Approach for Memory Leak Detection: Case Studies

Vladimir Šor[1,2], Nikita Salnikov-Tarnovski[2], and Satish Narayana Srirama[1]

[1] Institute of Computer Science, University of Tartu
J. Liivi 2, Tartu, Estonia
`{volli,srirama}@ut.ee`
[2] AS Webmedia R&D, Raatuse 20, Tartu, Estonia
`{volli,nikita}@webmedia.ee`

Abstract. Applications written in Java^{TM}language, and in other programming languages running on Java^{TM}Virtual Machine (JVM) are widely used in cloud environments. Although JVM features garbage collection, memory leaks can still happen in these applications. Current solutions for finding memory leaks have several drawbacks which become critical when deployed in distributed and dynamic environments like cloud. Statistical approach for memory leak detection gives good results in terms of false positives and we have implemented automatic statistical approach for memory leak detection in Java^{TM}applications. To test its correctness and performance we have conducted several experiments by finding memory leaks in a large web-application and searching for related bugs in open source projects from Apache Software Foundation. This paper presents the results of these experiments and concludes that automated statistical method for memory leak detection is efficient and can be used also in production systems to find hardly reproducible leaks.

Keywords: Memory leak, troubleshooting, Java^{TM}, cloud computing.

1 Introduction

Existing solutions for finding memory leaks in applications running in Java Virtual Machine (JVM) have several drawbacks even on single JVM and these drawbacks become critical when deployed in distributed and dynamic environments like cloud [7]. One of the main issues with existing solutions is the amount of qualified manual labor required to get to the right conclusions, i.e. find the real source of the leak. In cloud environment more intelligent troubleshooting tools are needed.

The statistical method for memory leak detection in garbage collected languages, e.g., Java, relies on the analysis of distribution of ages of live objects. If the number of different ages constantly grow, then these objects are created, but not reclaimed. This points to a potential memory leak. [17]

Although input data for the statistical method can be collected using a profiler (for example, NetBeans profiler [14] allows monitoring ages of objects), the analysis should be still done manually. In case of distributed or cloud applications, data has to be aggregated from different sources which further increases the complexity of the task. Moreover, the performance overhead that profilers impose does not allow using them in the production environments.

Rapid troubleshooting becomes important when transitioning from traditional approach to IT to Cloud ([11], p. 107). Dynamically scaling environments can produce vast amount of logging and monitoring data. The better tools for gathering such data get, the more data and with less overhead they will collect. But all this vast amount of monitoring and logging data has little value without proper algorithms that will help analyze the data and will aid to get faster to the problem resolution.

To use benefits of the statistical method and overcome aforementioned problems, we implemented automated statistical leak detection algorithm for Java applications using agents for JVM. Current implementation is autonomous and if memory leak is detected it just reports the results as an aggregate report.

To test correctness and performance of our tool we have conducted several experiments by finding known memory leak bugs in open source projects. Further analysis was conducted with a large web-application with a known memory leak. This paper presents short results of these experiments and concludes that automated statistical method for memory leak detection is efficient and can be used also in production systems to find hardly reproducible leaks.

The paper is organized as follows. In section 2 is given general overview of the automated statistical approach. In section 3 the general considerations of the case studies are given. Following sections (4 and 5) describe selected case studies. Section 6 discusses the related work and section 7 concludes the paper with directions for further research.

2 Automated Statistical Approach for Memory Leak Detection

Current implementation of the approach consists of two Java virtual machine agents: Java agent and native agent.

Java agent maintains required statistical metrics and performs concurrent statistical analysis after full garbage collection cycles. If statistical analysis concludes that objects of certain class are leaked, Java agent initiates further analysis to reveal reference path from leaking objects to garbage collection roots.

Native agent is required because not all required monitoring information is available through Java APIs. Thus native agent is responsible for gathering low level information, e.g., garbage collection and object freeing events, which are collected using JVM Tooling Interface (JVMTI, [15]) callbacks. Another responsibility of the native agent is to gather references between objects which are required by Java agent to find paths to garbage collection roots.

Agents operate together in following workflow:

1. track allocation of objects using byte code instrumentation;
2. analyze object age distribution according to statistical method for memory leak detection;

3. if instances of some classes reach predefined age distribution thresholds, then these classes are reported as leak candidates;
4. if previous step has identified a set of leak candidates, reference dump is captured and begins graph analysis to find shortest paths from leaking objects to garbage collection roots;
5. report is being generated.

There are certain nuances in searching for the path from leak to garbage collection roots. One of them is that objects usually leak in clusters – there is one main composite object which is actually leaking, but all object's fields are also leaked with the object. If object graph is more complex then even more classes are reported as leak suspects (which they actually are), but for the final report we must find that main object. This is the reason why objects are not reported and analyzed one by one, but are at first gathered and then analyzed at once to produce meaningful report.

It may happen that analysis and report generation must be performed in a very memory-limited environment – memory leak is eating up free heap space on one side and our analysis also requires some heap for the analysis on the other side. This may lead to a situation where JVM runs out of heap before analysis ends. To mitigate this situation the tool stores reference dump file and accompanying metadata so that analysis also can be performed offline. However, some runtime information will not be available in the report.

Such hybrid approach combines benefits of both online (e.g., profilers which monitor live data, having access to required reflection and behaviour information) and offline (heap dump analysis in separate environment) memory leak detection approaches. There is no need to wait for next leak to show itself to try to do the analysis online again, but rather do remaining analysis offline, which can be done much faster.

3 Case Studies

To evaluate the performance and precision of the implementation of automated statistical approach for memory leak detection we conducted several case study experiments. For these experiments we had to find known memory leak bugs to see if our tool could find the right reason of the leak. For that we searched the issue tracker of the Apache Software Foundation [4] for memory leak bugs in Java based projects. Among these we chose the ones that we were able to reproduce. In addition to these open source projects we also looked for known bugs in business web applications with closed source developed at AS Webmedia. Every case study followed the same workflow.

As a first step baseline was recorded – run every case study without our agents and record time to crash with OutOfMemoryError and memory usage during the course of run. Next, the case was run with troubleshooting agents attached. In addition to the baseline metrics (time to crash and memory usage) we were interested in the following information: whether our agent will find the leak, if there will be any false positive alerts, how fast will the agent be able to spot the leak and produce a report.

Experiments were conducted in the Amazon EC2 cloud [2], using Standard Large instance running 64bit Ubuntu Linux and Oracle Java^{TM}HotSpot^{TM} 1.6.0_24 64-Bit Server VM (build 19.1-b02, mixed mode). As of moment of writing, standard large instance has following characteristics: 7.5 GB memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB instance storage, 64-bit platform.

In total we have conducted 6 different experiments (bug reports) with 4 different projects from Apache Software Foundation (Velocity, Xalan, Derby and ActiveMQ) among which we have chosen ActiveMQ [1] for presentation in this article. In addition to that case study with closed source eHealth web-application are also presented in following sections.

4 Case Study: Apache ActiveMQ

Apache ActiveMQ [3] is an open source message broker which fully implements the Java Message Service 1.1 (JMS) specification. The subject of this case study was a bug, reported to Apache ActiveMQ project's issue tracker [1].

Two parties, a server, called broker, and a client, participate in communications using ActiveMQ infrastructure. Client opens a connection to the broker, and then, in violation of the ActiveMQ protocol, just drops it. This can take place either due to buggy client code or due to network problems. The result of such dropped connection is that the instance of object, representing the client who has initiated this connection, remains in the broker's memory. If many clients repeatedly create new connections to the same broker and at some point in the future just drop this connection without closing it this may result in many objects accumulating on the broker side. Connection objects are held by the broker until broker's memory is depleted which results in OutOfMemoryError.

In order to reproduce reported bug in our environment a sample client spawned a number of threads, each opening new connection to the broker and then abruptly closing this connection. This process was run repeatedly, simulating network problems over time. ActiveMQ broker was run with default settings as provided in default start-up scripts with the distribution.

For the baseline it was recorded that broker run for 1 hour and 17 minutes before the crash and its memory usage is depicted on the fig. 1.

With troubleshooting agents attached leak was spotted after 35 minutes since the application start. The agent was able to successfully find the correct source of leaking objects. No false positives were reported. Broker ran for 1 hour and 15 minutes before crash.

Comparison of memory usage graphs (fig. 1 and 2) shows that no significant memory overhead was created by troubleshooting agents. However, some slight increase in the frequency of full garbage collector runs can be seen (large drops in heap usage indicate full garbage collection).

Such memory leak can be easily exploited against the service provider who uses ActiveMQ to create Denial of Service (DoS) type attack. In case of elastic

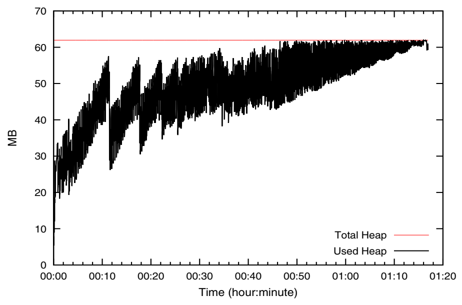


Fig. 1. ActiveMQ memory usage baseline

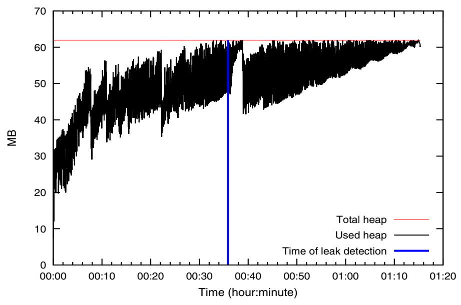


Fig. 2. ActiveMQ with agents

cloud computing environment with dynamic resource allocation provider could be attacked to allocate new instances to serve more and more non-existing clients and by that cause financial damage by forcing to pay for the allocated resources.

5 Case study: eHealth Web Application

This case study focuses on a large enterprise application (ca. 1 million lines of code) with web front-end – eHealth web-portal including patient record handling and many other health-care related functions. At some point development team has started to receive reports that application crashes in production environment every few days with `java.lang.OutOfMemoryError`. As a temporary workaround client's IT operations team had to restart application servers every other night in order to prevent crash. It must be noted that such solution is not uncommon.

At first we manually found the source of the leak by analyzing heap dumps. Then we wrote automatic test-case with Apache JMeter [5] to stress-test the memory leak in the application. Next we attached agents to application server in test environment and ran recorded stress-test until application crashed.

When the memory usage was first recorded we noted that without the agent memory consumption grows even faster than with agents (fig. 3 and 4). As memory consumption is closely related to the amount of requests performed by the application we also added number of performed requests to the picture.

As a baseline it was recorded that the application ran for 2h 30min before crash. Memory usage with number of the requests sent is depicted on the fig. 3.

With troubleshooting agents leak was spotted after 2h 30min since the application start. The agent was able to successfully find the correct source of leaking objects without false positives. Application run for about 6h before crash. Memory usage and number of requests are depicted on the fig. 4.

On both of the figures we see a rapid degradation of the application performance after about half an hour after beginning. This degradation can be easily explained by taking into account the amount of the free memory available to the application. As it decreases, JVM Garbage Collector starts to run more and more often and to take more and more time. Which in turn leaves less opportunities

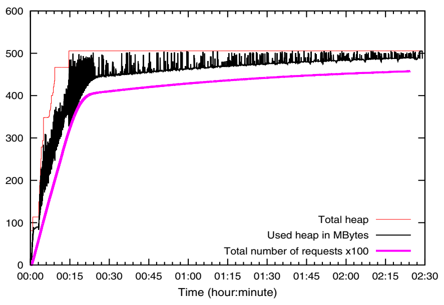


Fig. 3. eHealth application baseline

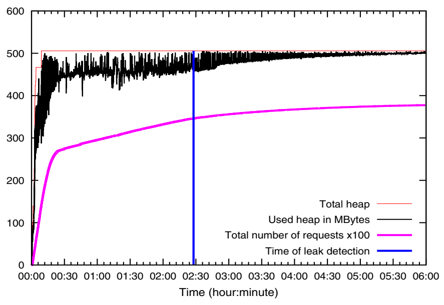


Fig. 4. eHealth application with agents

for the application to serve incoming requests. Hence performance degradation and, as a result, a slower increase in memory used. So, in case of memory leak situation, not only overall performance of the application, but the speed of the leak as well decreases significantly over time.

Comparing first 30 minutes of running the stress-test with and without the agent, we can estimate the overhead of running an application with our agent to about 35%. Which is quite large and certainly needs to be improved.

We also got one counter-intuitive result of the application running longer with the agents than without them. This can be attributed to the fact, that due to agents' performance overhead, less requests were served during this time, and as a result memory leak was consuming memory at a much slower pace.

From fig. 3 we can see that when amount of free memory in Java application reaches the limit, application is still running and garbage collector tries to free memory but the performance of the application is dramatically decreased. In current case clients would be experiencing long response times and timeout errors for 2 hours before the application would actually crash.

6 Related Work

Memory leaks have been studied in the industry and in research community extensively and currently there are several approaches for finding memory leaks in Java applications. These approaches can be divided in two large groups – offline and online analysis.

For offline analysis the heap dump is taken and then analyzed outside of the running JVM. There are several algorithms to analyze heap dumps to detect possible leaking objects. For example [13] shows usage of graph mining algorithms for this purpose. Eclipse Memory Analyzer or *MAT* [16] is an open source heap dump analysis software. Drawback of the offline analysis is the static nature of the memory dump – there is no information regarding the source of allocation of the objects, so finding the code responsible for memory leak is a separate task from just finding leaked objects.

Online analysis is done by monitoring live application. One widely used approach is to monitor certain collection classes for unlimited growth. This technique is also used in several Application Performance Monitoring (*APM*) suites. For example, CA Wily Introscope® LeakHunter$^{\text{TM}}$ [8] and AppDynamics [6]. Another similar solution is described in [18].

FindLeaks tool is using AspectJ pointcuts to analyze references between objects and find out leaking objects together with the sites of allocation [9].

Profilers are another type of tools often used in development to find memory leaks. Major disadvantage of the profilers is the need for qualified and experienced operator who can find the actual leaking code.

In addition to different instrumentation and byte code modification techniques there are several research projects applying different statistical methods for analyzing unnecessary references between objects: *Cork* [12], and stale objects: *SWAT* [10]. SWAT however, is not Java based.

7 Conclusions and Future Work

From the conducted case studies some areas which require further attention can be summarized as follows.

One of the focus areas for further development is the CPU overhead made by our agent. CPU overhead influences the number of requests application can serve per time unit. In case study of eHealth web-application (section 5) we saw a 35% degradation of the application performance measured in requests per second. This is too big overhead and certainly is unacceptable for production environments. Though it may be good enough for testing environments in order to reproduce and pin-point memory leak bug previously found in production.

No significant problems with memory overhead when using our agent were found during case studies. As graph analysis can be performed offline, as described in 2 memory overhead can be lowered even more.

It is our opinion that more effort should be put into researching if it is possible to deduct the user experience degradation due to memory problems on the server side based on the memory metrics that can be collected in the JVM. E.g., time spent in GC, GC frequency, ratio of fast GC to full GC etc. This way smarter monitoring applications and utilities can be developed, which give the application owner IT operations feedback based on the performance of the application as perceived by the clients. Combining this with the auto-scaling capabilities of cloud-based application will give the application owner finer and much more efficient control over application infrastructure without compromising users satisfaction.

To integrate with cloud infrastructure providers dashboards and other cloud monitoring software data collection and aggregation framework must be added to the memory troubleshooting tool. The preliminary idea of the agent based framework using collectD system statistic collection daemon is addressed in [17].

Acknowledgments. This research is part of the EUREKA Project 4989 (SITIO) co-funded by the European Regional Development Funds via Enterprise Estonia.

References

1. Active mq issue tracker, ticket 3021 (2011), https://issues.apache.org/jira/browse/AMQ-3021
2. Amazon Inc.: Elastic Compute Cloud (EC2) (2011), http://aws.amazon.com/ec2/
3. Apache Software Foundation: ActiveMQ (2011), http://activemq.apache.org/
4. Apache Software Foundation: Apache Software Foundation issue tracker (2011), https://issues.apache.org/bugzilla/
5. Apache Software Foundation: JMeter (2011), http://jakarta.apache.org/jmeter/
6. AppDynamics: home page (2011), http://www.appdynamics.com/
7. Armbrust, M., et al.: Above the Clouds, A Berkeley View of Cloud Computing. Technical report UCB/EECS-2009-28, University of California (February 2009)
8. CA Wily Introscope: Home page (2010), http://www.ca.com/us/application-management.aspx
9. Chen, K., Chen, J.B.: Aspect-based instrumentation for locating memory leaks in java programs. In: 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, vol. 2, pp. 23–28 (July 2007)
10. Chilimbi, T.M., Hauswirth, M.: Low-overhead memory leak detection using adaptive statistical profiling. In: 11th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 156–164 (2004)
11. Ellahi, T., Hudzia, B., Li, H., Lindner, M.A., Robinson, P.: The Enterprise Cloud Computing Paradigm. In: Cloud Computing: Principles and Paradigms. John Wiley & Sons, Inc. (2011)
12. Jump, M., McKinley, K.S.: Cork: dynamic memory leak detection for garbage-collected languages. In: 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2007), pp. 31–38. ACM (2007)
13. Maxwell, E.K.: Graph Mining Algorithms for Memory Leak Diagnosis and Biological Database Clustering. Master's thesis, Virginia Polytechnic Institute and State University (2010)
14. netbeans.org: NetBeans profiler (2011), http://profiler.netbeans.org/
15. Sun Microsystems Inc.: Jvm^{TM} tool interface (2006), http://download.oracle.com/javase/6/docs/platform/jvmti/jvmti.html
16. The Eclipse Foundation: Memory analyzer (2011), http://www.eclipse.org/mat/
17. Šor, V., Srirama, S.N.: A statistical approach for identifying memory leaks in cloud applications. In: First International Conference on Cloud Computing and Services Science (CLOSER 2011), pp. 623–628. SciTePress (May 2011)
18. Xu, G., Rountev, A.: Precise memory leak detection for java software using container profiling. In: ACM/IEEE 30th International Conference on Software Engineering, ICSE 2008, pp. 151–160 (May 2008)

ODBASE 2011 PC Co-chairs' Message

We are happy to present the papers of the 10th International Conference on Ontologies DataBases, and Applications of Semantics, ODBASE 2011, held in Heraklion, Crete (Greece), in October 2011. The ODBASE conference series provides a forum for research on the use of ontologies and data semantics in novel applications, and continues to draw a highly diverse body of researchers and practitioners by being part of the federated conferences event "On the Move to Meaningful Internet Systems (OnTheMove)" that co-locates three conferences: ODBASE, DOA-SVI (International Symposium on Secure Virtual Infrastructures), and CoopIS (International Conference on Cooperative Information Systems), and this was also done in 2011.

We received 29 paper submissions which were subjected to rigorous reviews and discussions among the PC members. We eventually accepted nine submissions as full papers, for an acceptance rate of 31%, and an additional four submissions as short papers and five papers as posters in the workshop proceedings.

ODBASE 2011 continued its traditional focus on work which bridges traditional boundaries between disciplines. We believe that this emphasis on interdisciplinarity is highly valuable for the community, and provides a necessary stimulus for the future development of Semantic Web research and practice.

August 2011

Manfred Hauswirth
Pascal Hitzler
Mukesh Mohania

RDFa Based Annotation of Web Pages through Keyphrases Extraction

Roberto De Virgilio

Dipartimento di Informatica e Automazione
Universitá Roma Tre, Rome, Italy
dvr@dia.uniroma3.it

Abstract. The goal of the Semantic Web is the creation of a linked mesh of information that is easily processable by machines, on a global scale. The process of upgrading current Web pages to machine-understandable units of information relies on semantic annotation. A typical process of semantic annotation includes three main tasks: (i) the identification of an ontology describing the domain of interest, (ii) the discovering of the concepts of the ontology in the target Web pages, and (iii) the annotations of each page with links to Web resources describing the content of the page. The goal is to support an ontology-aware agent in the interpretation of target documents. In this paper, we present an approach to the automatic annotation of Web pages. Exploiting a data reverse engineering technique, our approach is capable of: recognizing entities in Web pages, extracting keyphrases from them, and annotating such pages with RDFa tags that map discovered entities to Linked data repositories matching the extracted keyphrases. We have implemented the approach and evaluated its accuracy of on real Web sites for e-commerce.

1 Introduction

The Semantic Web [1] is an extension to the World Wide Web in which information is defined semantically in terms of concepts with meaning, with the goal of matching requests of people and machines in a more accurate way. Ever since Tim Berners Lee presented, in 2006, the design principles of Linked Open Data[1], the public availability of Semantic-Web data has grown rapidly. Today, many practitioners, organizations and universities are contributing to the this trend by publishing on the Web repositories of data expressed in RDF, the basic technology underlying the Semantic Web. In addition, ontology languages such as RDFS and OWL are also often adopted as a means to annotate Web data, since they enable various forms of reasoning. Unfortunately however, although the Semantic Web can provide exciting new functionality involving finding, sharing, and combining available information, a wide adoption of this extension of the Web is yet to be waited for. In this framework, one of the major research issue is the automatic annotation of Web pages with semantic tags.

[1] http://linkeddata.org/

A typical process of semantic annotation can be decomposed in three phases. First, an ontology describing the domain of interest is chosen. Then, instances of concepts in the ontology are identified in the target Web pages using a knowledge discovery technique. Finally, Web pages are annotated by associating Web resources with terms in the documents by means of *semantic links*. Such annotations can be then used by an ontology-aware agent to interpret the content of target documents.

Steps 2 and 3 of the process above involve two main tasks. First, the identification of candidate terms for adding links: typically they denote concepts of the ontology that can be of interest for the user. Second, the identification of a Web resource to associate with candidate terms. The first task can be done by adopting an information extraction technique. Two major sub-problems here are the decision on whether a term should be linked or not (to avoid overlinking), and the disambiguation of concepts to associate with candidate terms, since terms can have different meanings and consequently can demand different disclosures. The second task requires an external source of knowledge, which can be a predefined knowledge base (e.g. Wikipedia) or a distributed and wider data space such as Linked Open Data. The latter approach requires also a pre-selection of the sources to be considered, since only a few sources in the whole data space can contain relevant knowledge in the domain of interest.

Although several solutions are being proposed to automatically add semantic links to Web pages (e.g. [2,3,4,5,6,7]), they mainly adopt semi-automatic approaches and are ad-hoc implementations depending on the selected knowledge base (e.g. Wikipedia). Then usually such proposals do not obtain optimal values for both accuracy and efficiency.

In this paper, we propose a technique to the automatic annotation of existing Web pages with the necessary RDFa [8] attributes, a W3C Recommendation that provides a method for embedding rich metadata within HTML Web documents using a set of predefined attributes. Our approach relies on recent techniques for extracting information from the Web [9]. As in the case of most of these proposals, we start from the observation that data published in the pages of large Web sites usually (i) come from a back-end database and (ii) are embedded within shared HTML templates. Therefore the extraction process can be based on the inference of a description of the shared templates. Though this approach is applicable on Web documents, it does not exploit the hypertext structure of Web documents. Our work focuses on discovering this structure as well. Some research efforts have shown that users always expect that certain functional part of a Web page (e.g., navigational links, advertisement bar and so on) appears at certain position of a page[2]. Additionally, there exist blocks of information that involve frequent HTML elements and have a higher coherence. That it to say, in Web pages there are many unique information features that can be used to help the extraction of blocks involving homogeneous information. To this aim we have

[2] For more details see `http://www.surl.org/`

developed a data reverse engineering algorithm [10] to extract *semantic blocks* carrying information candidate for annotation. Then, a keyphrases algorithm [11] and a keyword search approach [12] allow us to extract the most meaningful key-terms from the Web page and to discover the concepts of a semantic knowledge base (e.g DBPedia) that best captures their semantics. Finally, Web pages are annotated, using the RDFa mechanism, linking terms to the selected semantic concepts.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 illustrates an architecture of reference that provides a functional global view of the approach. Section 4 describes step-by-step the annotation process. Section 5 discusses the implementation of our framework and evaluates the performance of our technique. Finally, in Section 6 we sketch conclusions and future work.

2 Related Work

Main efforts to annotation of Web pages can be classified with respect to the degree of automation of the entire process. In particular we distinguish *manual* and *automatic* approaches.

Manual Annotation. Manual annotation research focuses more on how to represent the annotation and on developing user-friendly interfaces to support the user to write down their annotation. The most representative manually annotating tool is *Annotea* [13]. Annotea describes annotations by an RDF-schema and uses XPointer to allocate them into the annotated document. Based on Annotea, there are several manual annotation tools that could potentially be used for semantically annotating the Web, which include but are not limited to: Annotator, CritLink, CoNote, ComMentor and ThirdVoice.

Automated Annotation. Research about automated annotation tools focuses more on how to create annotations according to specied domain ontologies. Most automatic semantic annotation tools exploit ontology-based Information Extraction (IE) to extract information (semantics) and compose the annotation resulting by matching the extracted information with a given pre-constructed domain ontology (i.e. in according to the Linked Data philosophy).

Ont-O-Mat [5] is the implementation of a framework (*S-CREAM*) supporting both manual and semi-automatic annotation of Web pages. Ont-O-Mat adopts automated data extraction technique from *Amilcare* [14], which is an adaptive IE system designed for supporting active annotation of documents. Ont-O-Mat implements different heuristics for post-processing and mapping of IE results to an ontology. Ont-O-Mat allows to access ontologies described in different markup formats (e.g. RDF and DAML+OIL), but the system can access to only one ontology per time. Ont-O-Mat provides a storage system for pages annotated in DAML+OIL using OntoBroker[3] as an annotation server [15]. Finally the tool

[3] `http://ontobroker.semanticWeb.org/`

exposes crawlers that can search the Web for annotated Web pages to add to its internal knowledge base.

MnM [6] is quite closed to Ont-O-Mat. It provides both automated and semi-automated annotation support. MnM implements an ontology editor, integrating it with a Web browser. Similar to Ont-O-Mat, MnM allows to access ontologies specified in different markup formats throw open APIs (e.g. OKBC[4]), that link to ontology servers and integrating IE tools. Differently from Ont-O-Mat, MnM can access multiple ontologies at the same time. Beyond this, MnM shares almost all the other features as for Ont-O-Mat. As stated by the authors, the difference between the two systems is their philosophies. While Ont-O-Mat adopts the philosophy that the markups should be included as part of the resources, MnM stores their annotations both as markups on a Web page and as items in a knowledge base.

In [16] the authors provide the *KIM* platform: it consists of a formal ontology, a knowledge base, a Server (with an API for remote access or embedding), and different user-interfaces to access to the functionality of the platform. The ontology is a light-weight upper level ontology, defining classes and relations of interest. The representation language is RDF-Schema. The knowledge base contains the entity description information for annotation purposes. During the annotation process, KIM combines NLP and IE techniques, integrating GATE[5] (General Architecture of Text Engineering), to extract, index, and annotate data instances. The KIM Server coordinates multiple units in the general platform. The annotated information is stored inside the Web pages. KIM front-ends provide a browser plug-in so that people can view graphically through different highlighted colors in regular Web browsers such as Microsoft's Internet Explorer.

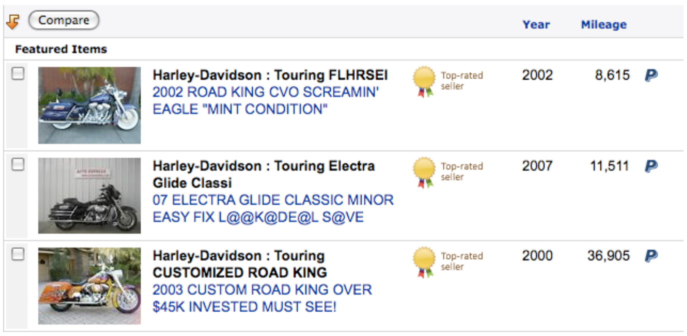
Finally *SemTag* [7] is the largest scale and most popular semantic tagging tool. SemTag uses the TAP[6] ontology to define annotation classes. The TAP ontology is very similar in size and structure to the KIM ontology and knowledge base. SemTag uses a vector-space model to assign the correct ontological class or to determine that a concept does not correspond to a class in TAP. The SemTag system is implemented on a high-performance parallel architecture: the final system results very efficient and provide high accuracy of annotation.

3 Architecture of Reference

Our main focus is to add semantic annotations to existing Web pages given as input in an automatic way. For instance let us consider an e-commerce Web site (e.g. eBay). Fig. 1 shows a snapshot of a list of Motorcycle items.

In particular the page illustrates different items of Harley-Davidson motorcycles. In the following we show the corresponding HTML code referring to the first item

[4] `http://www.ai.sri.com/~okbc/`
[5] `http://gate.ac.uk/`
[6] `http://tap.stanford.edu`



Compare			Year	Mileage	
Featured Items					
	Harley-Davidson : Touring FLHRSEI 2002 ROAD KING CVO SCREAMIN' EAGLE "MINT CONDITION"	Top-rated seller	2002	8,615	P
	Harley-Davidson : Touring Electra Glide Classi 07 ELECTRA GLIDE CLASSIC MINOR EASY FIX L@@K@DE@L S@VE	Top-rated seller	2007	11,511	P
	Harley-Davidson : Touring CUSTOMIZED ROAD KING 2003 CUSTOM ROAD KING OVER $45K INVESTED MUST SEE!	Top-rated seller	2000	36,905	P

Fig. 1. An eBay example page

```
<ul>
   ...
   <li> <div class="description">
   <h1>Harley-Davidson: Touring FLHRSEI</h1>
   <a href=http://.../description>
     2002 ROAD KING CVO SCREAMIN EAGLE MINT CONDITION
   </a>
    ...
  </div>   </li>
  ...
</ul>
```

Our framework for automatically adding semantic annotations to a Web page (as the above illustrated) is composed of a number of stages. Fig. 2 sketches the main steps of our framework and their interdependencies.

The process starts selecting the parts of a Web page to investigate. Requirements to the selection are a high sensitivity (low false negative rate), good specificity (low false positive rate), and reasonable robustness (i.e. the ability to be

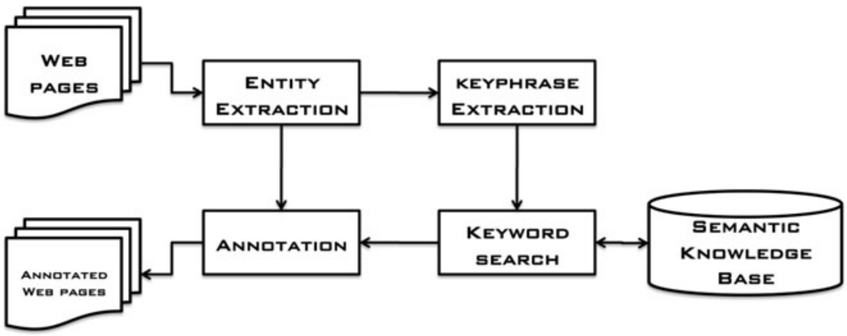


Fig. 2. Annotation process

used on a variety of Web pages that do not match the trained set). Therefore the Entity Extraction step identifies the so-called *semantic blocks* that are the areas of a page most likely to be a place of interest (e.g a document section containing information to annotate). To determine such blocks, we exploit the segmentation algorithm provided in [10] supported by a Web data model [17] to describe abstract structural features of HTML pages. Combining cognitive visual analysis and hierarchy-based algorithms, they identify blocks grouping semantically related objects occurring in Web pages, and generate a logical schema of a Web site. In the Keyphrase Extraction step, we extract keyphrases from the individuates blocks coming from the previous step. This step is supported by KEA [11], an algorithm for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary. The third step is the Keyword Search that processes a keyword search query with the keywords resulting from the Keyphrase Extraction on a Semantic Knowledge Base. This step individuates the semantic concepts best matching the query: we employ the algorithm in [12]. Finally the Annotation takes place supported by RDFa format to model the annotations in the page.

In the following section we will describe in detail the entire process.

4 Automatic Annotation of Web Pages

4.1 Semantic Block Identification

Following the approach in [10], we have to select portions of an HTML pages and map them to *constructs* of the Web Site Model (WSM) [17]. WSM defines a conceptual modeling of the main structures occurring into a Web page. More precisely, it organizes a page in a set of *meta containers* related by *links*. A metacontainer is *atomic* if it represents an atomic portion of a linked page and includes a direct reference to the elements of a content from which it takes data. It can be *basic* if it shows information about a single object (e.g. an instance of an entity) or *multi* if it shows information about a set of objects. A metacontainer is a *linker* if it contains an anchor between containers. Otherwise a metacontainer is *complex* if it is articulated in other containers (atomic, complex or linker). So a page represents a complex container. We can surf the containers through several *navigational structures* such as *Indexes* to show a list of objects without presenting the detailed information of each one, *Guided Tours* to show commands for accessing the elements of an ordered set of objects, and *Entries* to show edit field for inputting values used for searching within a set of objects meeting a condition. For instance, Fig. 3 shows a commercial Web site about several discounted items grouped in categories, represented in the WSM model. In the Figure, `Home` and `Detail` correspond to complex metacontainers. In particular `Home` describes different discounted items as basic containers (i.e. `discount`) that can be navigated by an Index structure (i.e. `Discounts`).

Implementing the segmentation algorithm of [10], we determine several blocks into a Web page identified by the HTML tags pattern to reach them. As in [10], the idea is to identify a set of *container tags* representing candidates to be

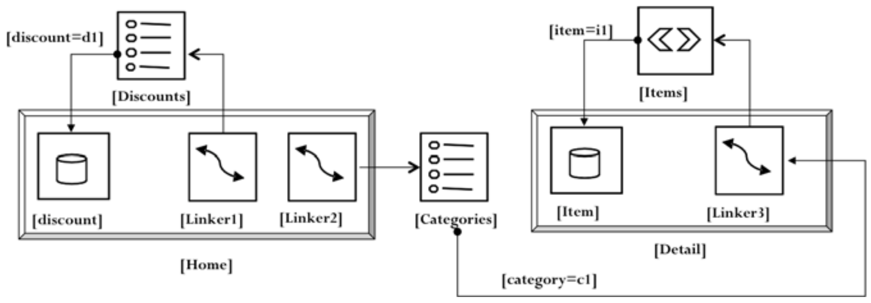


Fig. 3. The WSM representation of a commercial Web Site

mapped. In particular we refer to HTML tags that bring to information content in a Web page such as `UL, TABLE, DIV, BODY,` .... We call *non informative tags* HTML tags that don't bring to informative content (such as `SPAN, B` and so on). Each pattern rooted in a tag container will be translated into a metacontainer using a particular navigational structure (Index, Guided Tour or Entry). Let's consider to define a *target pattern t* for each construct. For instance the Index navigational structure presents `UL-LI-A` as target pattern t_{Index}. If we want to map a source pattern s to t_{Index} we have to find an alignment between s and t_{Index}. To this aim we will use the dynamic programming algorithm to calculate the optimal score and to find the optimal alignment between two patterns (i.e. alignment between two strings) [18].

First, we will compute the optimal alignment for every substring and save those scores in a matrix. For two strings, s of length m and t of length n, $D[i, j]$ is defined to be the best score of aligning the two substrings $s[1 \ldots j]$ and $t[1 \ldots i]$. A scoring matrix for scoring substitutions, matches, and gap creation is needed. The score is 2 for a match, 1 for a partial match and -1 for a mismatch. The match is based on tags comparison between patterns. A match (mismatch) is between container tags while a partial match is between a container tag and a non informative tag. Then we will consider global alignments: the conditions are set such that we compute the best score and find the best alignment of two complete strings. Therefore the best score for the alignment is precisely $D[m, n]$, the last value in the table. We will compute $D[m, n]$ by computing $D[i, j]$ for all values of i and j where i ranges from 0 to m and j ranges from 0 to n. These scores, the $D[i, j]$ are the optimal scores for aligning every substring, $s[1 \ldots j]$ and $t[1 \ldots i]$.

In general, dynamic programming has two phases: the forward phase and the backward phase. In the forward phase, we compute the optimal cost for each subproblem. In the backward phase, we reconstruct the solution that gives the optimal cost. For our string matching problem, specifically, we will: (i) use the recurrence relation to do the tabular computation of all $D[i, j]$ (forward phase), and (ii) do a traceback to find the optimal alignment.

The recurrence relation establishes a relationship between $D[i, j]$ and values of D with indices smaller than i and j. When there are no smaller values of i

and j then the values of $D[i,j]$ must be stated explicitly in the base conditions. The base conditions are used to calculate the scores in the first row and column where there are no matrix elements above and to the left. This corresponds to aligning with strings of gaps. After the base conditions have been established, the recurrence relation is used to compute all values of $D[i,j]$ in the table. The recurrence relation is:

$$D[i,j] = max\{D[i-1,j-1] + sim\_mat[s[j],t[i]], D[i-1,j] + gap\_score, D[i,j-1] + gap\_score\}$$

In other words, if you have an optimal alignment up to $D[i-1,j-1]$ there are only three possibilities of what could happen next: (1) the characters for $s[i]$ and $t[j]$ match, (2) a gap is introduced in t and (3) a gap is introduced in s. It is not possible to add a gap to both substrings. The maximum of the three scores will be chosen as the optimal score and is written in matrix element $D[i,j]$.

The second phase is to construct the optimal alignment by tracing back in the matrix any path from $D[m,n]$ to $D[0,0]$ that led to the highest score. More than one possibility can exist because it is possible that there are more than one ways of achieving the optimal score in each matrix element $D[i,j]$. In our case we have to select the target pattern that presents the best scoring matrix (i.e. the best alignment with the source pattern s).

For instance consider the pattern s, that is `UL-LI-DIV-A`. We have to map s into a metaconstruct of WSM. We have to find an alignment between s and a target pattern t. The best alignment is with the target pattern `UL-LI-A`. To this aim, we obtain the following scoring matrix.

t/s	UL	LI	DIV	A
UL	*2*	*-1*	*1*	*-1*
LI	*-1*	*2*	*1*	*-1*
A	*-1*	*-1*	*1*	*2*

The best alignment is `UL-LI-#-A`, therefore we will map the pattern s with an Index structure of WSM.

Referring to the example of Fig. 1, we have a list of items that is represented in WSM as an Index Structure (i.e. the list of tags `<a>`) linked to a basic container (i.e. the targets of `href` in the tags `<a>`), as shown in Fig. 4.

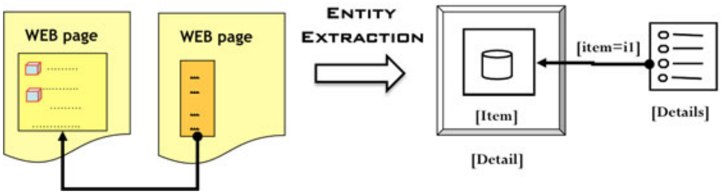


Fig. 4. The ENTITY EXTRACTION result

4.2 Keyphrase Extraction

Each semantic block has to be analyzed by the KEA algorithm to extract the keyphrases. KEA provides semantic metadata that summarize and characterize blocks. The algorithm identifies candidate keyphrases using lexical methods (i.e. TD/IDF), calculates feature values for each candidate, and uses a machine-learning approach to predict which candidates are good keyphrases. Referring to the example of Fig. 1, it is straightforward that the extracted keyphrase is `Harley Davidson`, since `Harley` and `Davidson` are the most frequent terms. In this case we have a map $\mathcal{M}$ where the keys are the keywords extracted (i.e. in this case `Harley` and `Davidson`), and the values are the semantic blocks containing such keywords (i.e. the Index Structure).

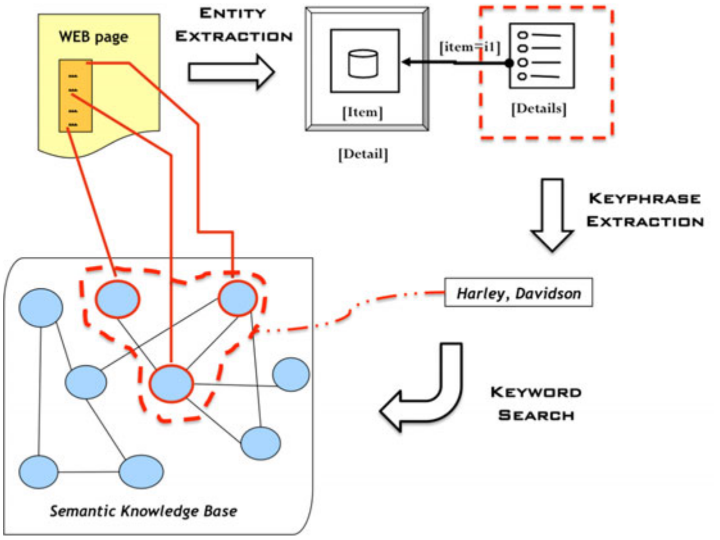


Fig. 5. Search over the Semantic Knowledge Base

4.3 Keyword Search

Once extracted the most meaningful keywords from the semantic block, we reduce the problem to submit a keyword search query to a semantic knowledge base. Fig. 5 illustrates the process, referring to our example. We employ the technique in [12], where a novel approach to keyword search over semantic data combines a solution building algorithm and a ranking technique with the goal of generating the best results in the first retrieved answers. In our case we retrieve only the top solution: the result is the RDF graph that best combines the keywords extracted by the KEA algorithm. As we will illustrate in the next section, we use DBPedia[7] as Semantic Knowledge Base. However our technique is independent from the particular semantic dataset. Once we obtained the

[7] `http://dbpedia.org/`

RDF graph, we select the nodes that exactly match the keywords and all the properties directly connected. For instance, considering again our example, given the query `Harley Davidson`, the Keyword Search returns the following RDF graph (i.e. expressed in NTRIPLE)

```
<dbp:Harley_Davidson> <rdf:type> <dbp:Motorcycle>
<dbp:Harley_Davidson> <rdfs:label> "Harley Davidson"
...
```

For the sake of simplicity we reported only the most interesting semantic information and we used the namespaces `dbp` (`http://dbpedia.org/resource/`), `rdf` (`http://www.w3.org/1999/02/22-rdf-syntax-ns#`) and `rdfs` (`http://www.w3.org/2000/01/rdf-schema#`) instead of verbose URIs.

4.4 Annotation

The final step is to generate the annotated Web pages introducing the semantic information retrieved in the previous steps. The most popular annotation styles are RDFa [8] and Microformats[8]. The simpler Microformats style uses HTML *class* attributes populated with conventional names for certain properties, which has the advantage of being usable for Web page formatting and easy to write for humans. However, these advantages are largely irrelevant for our purpose. RDFa benefits from RDF, the W3C's standard for interoperable machine-readable data. RDFa is considered more flexible and semantically rich than Microformats [19]. Therefore RDFa is a better choice for meta-data annotation than Microformats.

In addition to RDFa, it is possible to use standardized vocabulary for the specific domain to annotate. In particular we used the e-commerce as specific domain. To this aim we employ *GoodRelations*[9]. GoodRelations is a standardized vocabulary for product, price, and company data that (1) can be embedded into existing static and dynamic Web pages and that (2) can be processed by other computers. This increases the visibility of products and services in the latest generation of search engines, recommender systems, and novel mobile or social applications. Let us remind Google is now officially recommending GoodRelations for Rich Snippets[10].

Referring to our example, given the semantic information about `Harley Davidson`, we introduce them into an RDFa document, based on the GoodRelations vocabulary, as follows

```
<ul xmlns="http://www.w3.org/1999/xhtml"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

[8] `http://microformats.org/about`
[9] `http://www.heppnetz.de/projects/goodrelations/`
[10] `http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html`

```
        xmlns:gr="http://purl.org/goodrelations/v1#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
   ...
   <li> <div class="description"
            about="ProductOrServicesSomeInstancesPlaceholder_1"
            typeof="gr:ProductOrServicesSomeInstancesPlaceholder">
      <div rel="rdf:type"
           resource="http://dbpedia.org/resource/Motorcycle"/>
      <div property="rdfs:label"
           content="Harley-Davidson" xml:lang="en"/>
      <h1>Harley-Davidson: Touring FLHRSEI</h1>
      <a href=http://.../description>
        2002 ROAD KING CVO SCREAMIN EAGLE MINT CONDITION
     </a>
      ...
    </div> </li>
  ...
</ul>
```

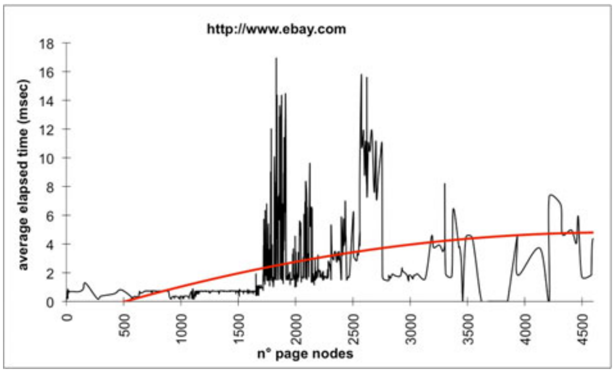
5 Experimental Results

On the basis of the methodologies and techniques above described, we have designed a Java tool to the automatic annotation of existing Web pages with the necessary RDFa attributes. Referring to the architecture illustrated in Fig. 2, each step is implemented in a separate module. The Entity Extraction module integrates an optimized version of the ReverseWeb tool illustrated in [10]. The Keyphrase Extraction module employes the Java implementation of KEA[11] while the Keyword Search module the Yaanii tool [12]. We used `DBPedia` as Semantic Knowledge Base. Finally the Annotation module takes as input the RDF graphs returned by Yaanii and produces RDFa documents, on the basis of the GoodRelations vocabulary.

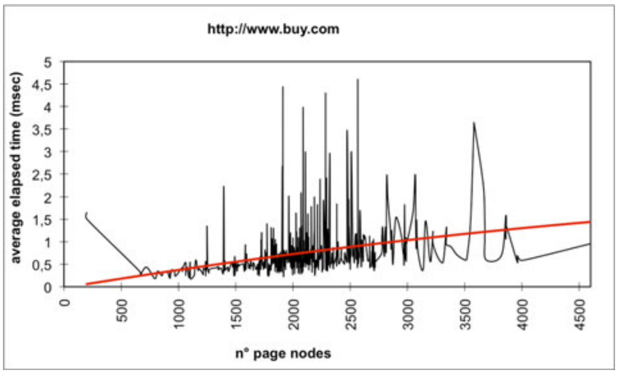
In our experiments we evaluated both the efficiency and the effectiveness. These experiments rely on crawling and annotating 100.000 pages for three popular e-commerce Web sites: *eBay*[12], *buy*[13] and *bestbuy*[14]. In particular we compared our system with SemTag [7]. In the following we omit results for bestbuy, since they are quite similar to buy.

Experiments were conducted on a dual core 2.66GHz Intel Xeon, running Linux RedHat, with 4 GB of memory, 6 MB cache, and a 2-disk 1Tbyte striped RAID array.

[11] Available at `http://www.nzdl.org/Kea/`
[12] `www.ebay.com`
[13] `www.buy.com`
[14] `www.bestbuy.com`



(a)



(b)

Fig. 6. Efficiency of our approach

5.1 Efficiency Evaluation

We measured the *average elapsed time* to produce an annotated page. Fig. 6 shows the results for our system. The diagram illustrates the average elapsed time (msec) with respect to the increasing number of DOM nodes in a Web page for the Web sites eBay (Fig. 6.(a)) and buy (Fig. 6.(b)). The Figure shows the efficient trend for both Web sites. Since eBay presents a more variable structure of pages with respect to buy, the corresponding diagram presents an increasing trend of elapsed times. A more significant result is the *speed-up* between our system and SemTag. We computed the speed-up with respect to the size (i.e. number of DOM nodes) of a page as the ratio between the average execution time (i.e. to annotate a page) of SemTag (t_{SemTag}), and that of our approach t_{own}, or briefly $\frac{t_{SemTag}}{t_{own}}$, as shown in Table 1. In general, our system performs very well with respect to SemTag in any case: of course the size of the page increases the complexity of the Entity Extraction reducing the speed-up.

Table 1. Speed-up

	500	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
eBay	20	20	18	16	12	11	9	7	6
buy	18	17	15	11	8	5	3	2	2

5.2 Effectiveness Evaluation

Then, we define the *precision* of an annotated Web page

$$precision\ (\mathcal{P}) = \frac{|SC \cap SC'|}{|SC|}$$

where SC is the set of concepts retrieved by the annotation process to annotate a Web page while SC' the set of correct concepts to insert into the annotation. It compares the number (size) of the expected (correct) semantic concepts to insert into the annotation with the real number of retrieved concepts. This coefficient is in a range [0,1]. If the precision is too close to zero, this means that the system was not able to identify significant terms in the knowledge base. Conversely, if the precision is more close to one, the system best fitted the semantic of the Web page (i.e. the *noise* is well reduced).

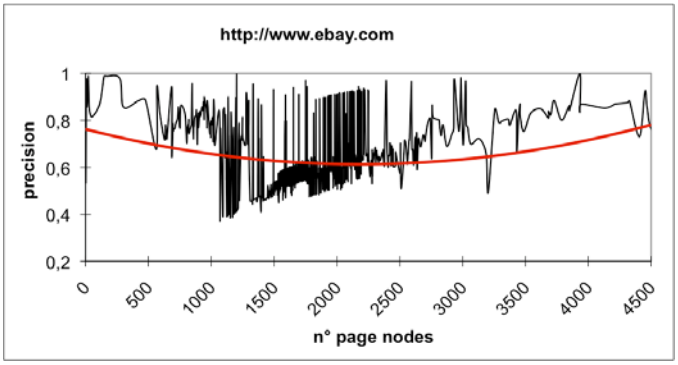
Fig. 7 illustrates the precision for eBay (Fig. 7.(a)) and buy (Fig. 7.(b)). In particular we measured the average precision of the annotation of a Web page (for each Web site) with respect to the increasing size of the page (i.e. the number of DOM nodes). Both the diagrams show a reasonable and effective quality trend. An interesting result is that the precision trend of eBay is better than that of buy. This is due to the more heterogeneity of Web content and structure of eBay with respect to buy. Such heterogeneity supports the KEA algorithm to distinguish key-terms. Moreover the more diffusion and popularity of eBay favors the *lexical matching* of key-terms with the knowledge base (i.e. `DBPedia`). As for the efficiency evaluation, we use a similar comparison with SemTag. In this case we measure the ratio between the precision of our system ($\mathcal{P}_{own}$) and that of SemTag ($\mathcal{P}_{SemTag}$) with respect to the number of DOM nodes in a page. Briefly, we have $\frac{\mathcal{P}_{own}}{\mathcal{P}_{SemTag}}$. Table 2 illustrates the final results.

Also in this case, our system is better than SemTag. In particular the precision grows with the increasing of the size of the page. While SemTag introduces noise (i.e. no meaningful concepts) due to the significant size of the page, our system improves the concept discovery due to both the KEA algorithm and the keyword search technique.

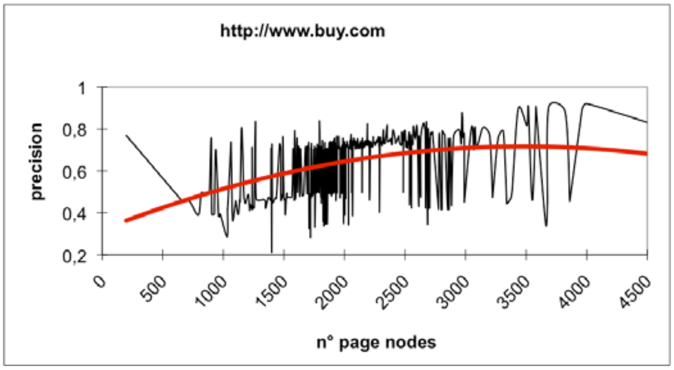
Then, we define the *recall* of an annotated Web page

$$recall\ (\mathcal{R}) = \frac{|SC \cap SC'|}{|SC'|}$$

where SC is the set of concepts retrieved by the annotation process to annotate a Web page and SC' the set of correct concepts to insert into the annotation.



(a)



(b)

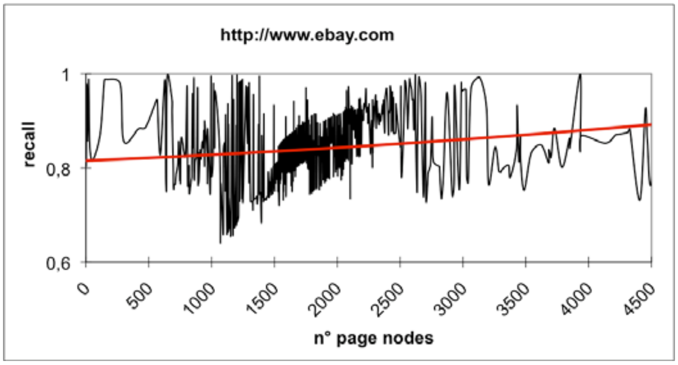
Fig. 7. Precision of our approach

It compares the number (size) of the expected (correct) semantic concepts to insert into the annotation with the real number of retrieved correct concepts. This coefficient is in a range [0,1]. As for the precision, if the recall is too close to zero, this means that the system identify false positive terms in the knowledge base. Conversely, if the recall is more close to one, the system best fitted the semantic of the Web page.

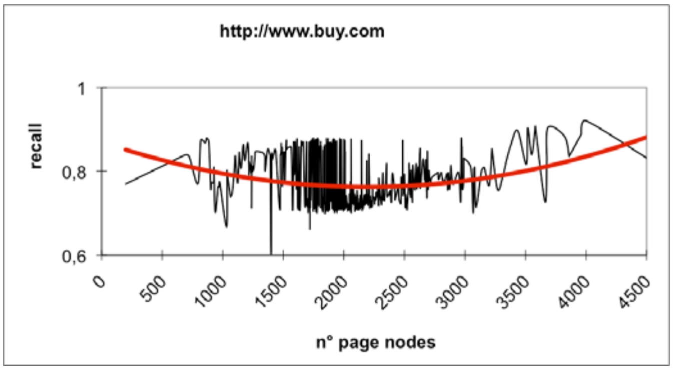
Fig. 8 illustrates the recall for eBay (Fig. 8.(a)) and buy (Fig. 8.(b)). Both the diagrams show a more reasonable and effective quality trend with respect to the precision. As for the precision, the recall trend of eBay is better than that

Table 2. Precision comparison

	500	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
eBay	1	1	1,8	1,7	1,7	2	2,5	3,2	3,2
buy	1	1,5	1,6	1,8	1,8	2,1	2,2	2,3	2,5



(a)



(b)

Fig. 8. Recall of our approach

of buy, since eBay presents a more heterogeneity of Web content and structure with respect to buy. Also in this case we compare our approach with SEMTAG. In this case we measure the ratio between the recall of our system ($\mathcal{R}_{own}$) and that of SEMTAG ($\mathcal{R}_{SemTag}$) with respect to the number of DOM nodes in a page. Briefly, we have $\frac{\mathcal{R}_{own}}{\mathcal{R}_{SemTag}}$. Table 3 illustrates the final results.

Finally, we define the *F-measure* of an annotated Web page

$$F\text{-}measure\ (\mathcal{F}) = 2 \cdot \frac{\mathcal{P} \cdot \mathcal{R}}{\mathcal{P} + \mathcal{R}}$$

Table 3. Recall comparison

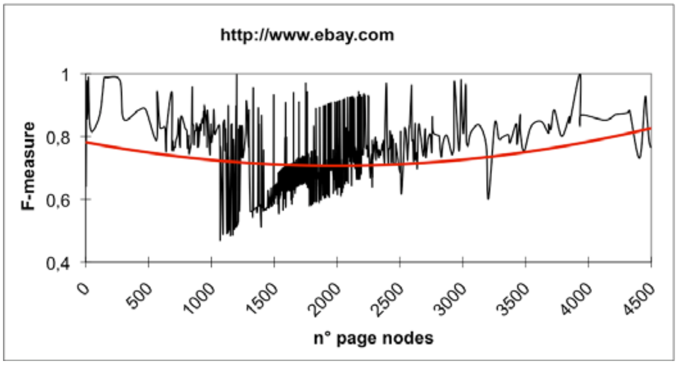
	500	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
eBay	1	1,2	1,8	2,1	2,9	3,2	3,5	3,9	4,1
buy	2	2	1,5	1,8	1,8	2,1	2,1	2,1	2,1

Table 4. F-measure comparison

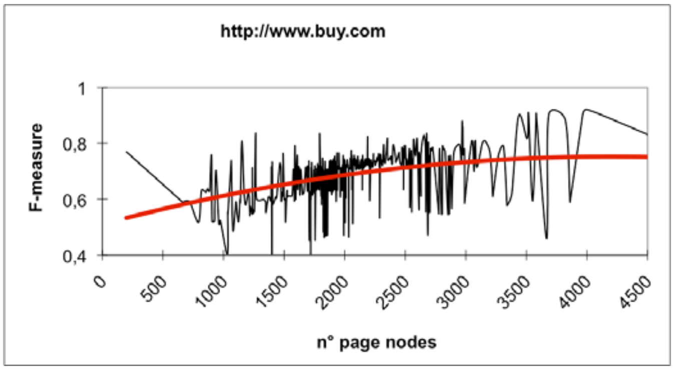
	500	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
eBay	1	1,1	1,8	1,9	2,1	2,5	2,9	3,5	3,6
buy	1,3	1,7	1,5	1,8	1,8	2,1	2,1	2,2	2,3

it is a measure of a test's accuracy. This coefficient compares the number (size) of true positives (tp) (i.e. $\mathcal{P} \cdot \mathcal{R}$) with the total (retrived and expected) number (size) of results, true positives, false positive (fp) and false negative (fn), that is $tp+fp+fn$ ((i.e. $\mathcal{P}+\mathcal{R}$)). This coefficient is in a range [0,1]. If the accuracy is too close to zero, this means that the system was completely not able to identify significant concepts, otherwise if the accuracy is more close to one, the system perfectly fitted the semantic of the Web page.

Fig. 9 illustrates the F-measure for eBay (Fig. 9.(a)) and buy (Fig. 9.(b)). Both the diagrams show a reasonable and effective quality trend with respect



(a)



(b)

Fig. 9. F-measure of our approach

to the F-measure. Also in this case we compare our approach with SemTag. In this case we measure the ratio between the recall of our system ($\mathcal{F}_{own}$) and that of SemTag ($\mathcal{F}_{SemTag}$) with respect to the number of DOM nodes in a page. Briefly, we have $\frac{\mathcal{F}_{own}}{\mathcal{F}_{SemTag}}$. Table 4 illustrates the final results.

6 Conclusion and Future Work

The Semantic Web promises exciting new functionality involving finding, sharing, and combining available information, but a wide adoption of this extension of the Web is still in its infancy. Although many practitioners, organizations and universities are publishing on the Web repositories of data expressed in ontology languages such as RDFS and OWL, there is a demanding need for techniques capable of enriching, in an automatic way, traditional Web pages with semantic information. In order to provide a contribution in this direction, we have presented in this paper a novel approach to the automatic annotation of Web pages with semantic tags. Our technique relies on a data reverse engineering technique and is capable of: (i) recognizing entities in Web pages, (ii) extracting keyphrases from them, and (iii) annotating such pages with RDFa tags that map the extracted keyphrases to entities occurring in Linked data repositories. We have implemented the approach and evaluated its accuracy of on real Web sites for e-commerce. In addition, we have compared the performance of our system with competing frameworks: it turns out that our system behaves very well with respect to them, exhibiting significant speed-up and accuracy.

Future directions includes practical and theoretical studies. From a theoretical point of view, we are investigating new algorithms to keyphrase extraction aimed at minimizing the best meaningful set of key-terms. From a practical point of view, we are developing optimization techniques of the entire process and a mechanism to automatically select the semantic knowledge bases of interest (e.g. exploiting the Linked Data technology) and to infer new concepts from them.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)
2. Mihalcea, R., Csomai, A.: Wikify!: linking documents to encyclopedic knowledge. In: CIKM, pp. 233–242 (2007)
3. Milne, D.N., Witten, I.H.: Learning to link with wikipedia. In: CIKM, pp. 509–518 (2008)
4. Gardner, J.J., Xiong, L.: Automatic link detection: a sequence labeling approach. In: CIKM, pp. 1701–1704 (2009)
5. Handschuh, S., Staab, S., Ciravegna, F.: S-CREAM – Semi-Automatic Creation of Metadata. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 358–372. Springer, Heidelberg (2002)
6. Vargas-Vera, M., Motta, E., Domingue, J., Lanzoni, M., Stutt, A., Ciravegna, F.: MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 379–391. Springer, Heidelberg (2002)

7. Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R.V., Jhingran, A., Kanungo, T., McCurley, K.S., Rajagopalan, S., Tomkins, A., Tomlin, J.A., Zien, J.Y.: A case for automated large-scale semantic annotation. J. Web Sem. 1(1), 115–132 (2003)
8. Adida, B., Birbeck M.: RDFa Primer: Bridging the Human and Data Webs (2008), http://www.w3.org/TR/xhtml-rdfa-primer/
9. Laender, A., Ribeiro-Neto, B., Silva, A.D., Teixeira, J.S.: A brief survey of web data extraction tools. ACM SIGMOD Record 31(2), 84–93 (2002)
10. De Virgilio, R., Torlone, R.: A Structured Approach to Data Reverse Engineering of Web Applications. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 91–105. Springer, Heidelberg (2009)
11. Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G.: Kea: Practical automatic keyphrase extraction. In: ACM DL, pp. 254–255 (1999)
12. De Virgilio, R., Cappellari, P., Miscione, M.: Cluster-Based Exploration for Effective Keyword Search Over Semantic Datasets. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 205–218. Springer, Heidelberg (2009)
13. Kahan, J., Koivunen, M.R., Prud'hommeaux, E., Swick, R.R.: Annotea: an open rdf infrastructure for shared web annotations. Computer Networks 39(5), 589–608 (2002)
14. Ciravegna, F., Dingli, A., Wilks, Y., Petrelli, D.: Amilcare: adaptive information extraction for document annotation. In: SIGIR, pp. 367–368 (2002)
15. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology based access to distributed and semi-structured information. In: Proceedings of the IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics- Semantic Issues in Multimedia Systems, vol. DS-8, pp. 351–369 (1998)
16. Kiryakov, A., Popov, B., Terziev, I., Manov, D., Ognyanoff, D.: Semantic annotation, indexing, and retrieval. J. Web Sem. 2(1), 49–79 (2004)
17. De Virgilio, R., Torlone, R.: A Meta-Model Approach to the Management of Hypertexts in Web Information Systems. In: Song, I.-Y., Piattini, M., Chen, Y.-P.P., Hartmann, S., Grandi, F., Trujillo, J., Opdahl, A.L., Ferri, F., Grifoni, P., Caschera, M.C., Rolland, C., Woo, C., Salinesi, C., Zimányi, E., Claramunt, C., Frasincar, F., Houben, G.-J., Thiran, P. (eds.) ER Workshops 2008. LNCS, vol. 5232, pp. 416–425. Springer, Heidelberg (2008)
18. Allison, L., Wallace, C.S., Yee, C.N.: When is a string like a string? AI & Maths (1990)
19. Tomberg, V., Laanpere, M.: RDFa versus Microformats: Exploring the Potential for Semantic Interoperability of Mash-up Personal Learning Environments. In: Second International Workshop on Mashup Personal Learning Environments, M. Jeusfeld c/o Redaktion Sun SITE, Informatik V, RWTH Aachen, pp. 102–109 (2009)

An Ontological and Terminological Resource for n-ary Relation Annotation in Web Data Tables

Rim Touhami[1,3], Patrice Buche[1,2],
Juliette Dibie-Barthélemy[3], and Liliana Ibănescu[3]

[1] INRA - UMR IATE, 2, place Pierre Viala, F-34060 Montpellier Cedex 2, France
[2] LIRMM, Montpellier, France
[3] INRA - Mét@risk & AgroParisTech, 16 rue Claude Bernard, F-75231 Paris Cedex 5, France

`Patrice.Buche@supagro.inra.fr`,
{`rim.touhami,Juliette.Dibie,Liliana.Ibanescu`}`@agroparistech.fr`

Abstract. We propose, in this paper, a model for an Ontological and Terminological Resource (OTR) dedicated to the task of n-ary relations annotation in Web data tables. This task relies on the identification of the symbolic concepts and the quantities, defined in the OTR, which are represented in the tables' columns. We propose to guide the annotation by an OTR because it allows a separation between the terminological and conceptual components and allows dealing with abbreviations and synonyms which could denote the same concept in a multilingual context. The OTR is composed of a generic part to represent the structure of the ontology dedicated to the task of n-ary relations annotation in data tables for any application and of a specific part to represent a particular domain of interest. We present the model of our OTR and its use in an existing method for semantic annotation and querying of Web tables.

Keywords: Semantic integration, semantic data model, ontology engineering.

1 Introduction

Today's Web is not only a set of semi-structured documents interconnected via hyper-links. A huge amount of technical and scientific documents, available on the Web or the hidden Web (digital libraries, ...), include data tables. They represent a very interesting potential external source for loading a data warehouse dedicated to a given domain of application. They can be used to enrich local data sources or to compare local data with external ones. In order to integrate data, a preliminary step consists in harmonizing external data with local ones, i.e. external data must be expressed with the same vocabulary, generally represented by an ontology, as the one used to index the local data. Ontology is a key notion in the Semantic Web and in data integration researches. According to [1], "Ontologies are part of the W3C standards stack for the Semantic Web, in which they are used to specify standard conceptual vocabularies in which to exchange data among systems, provide services for answering queries, publish reusable knowledge bases, and offer services to facilitate interoperability across multiple, heterogeneous systems and databases".

In [2,3,4,5,6] ontologies are associated with terminological and/or linguistic objects. In [2] authors motivate why it is crucial to associate linguistic information (part-of-speech, inflection, decomposition, etc.) with ontology elements (concepts, relations, individuals, etc.) and they introduce *LexInfo*, an ontology-lexicon model, implemented as an OWL[1] ontology. Adapting *LexInfo*, [3] presents a model called *lemon* (Lexicon Model for Ontologies) that supports the sharing of terminological and lexicon resources on the Semantic Web as well as their linking to the existing semantic representations provided by ontologies. The *CTL* model from [4] is a model for the integration of conceptual, terminological and linguistic objects in ontologies. In [5] a meta-model for ontological and terminological resources in OWL DL is presented, called an *Ontological and Terminological Resource (OTR)*, extended afterward in [7] in order to be used for ontology based information retrieval applied to automotive diagnosis.

In the same trend as in [5], we present in this paper an Ontological and Terminological Resource (OTR) dedicated to the task of data tables integration. An Ontological and Terminological Resource (OTR) [6,5] is a model allowing joint representation of an ontology and its associated terminology. According to [5], the OTR structuring can be guided by three factors: the task to realize, the domain of interest and the application. In this paper, the domain of interest is the food safety but the OTR structure we propose is generic enough to be applied to many other domains. The application is the construction of a data warehouse opened on the Web. We are interested in loading our data warehouse with data coming from external sources such as scientific papers, international reports or Web pages and in its querying.

In previous works [8,9], we proposed a data tables semantic annotation method guided by an ontology, but we did not especially pay attention to the ontology modeling and only use a preliminary version built from scratch by domain ontologists. Nevertheless, since our ontology is at the heart of our method, it appears that its modeling is essential to the sustainability of our approach and more generally to the data tables semantic annotation task. Like in [10,11], we are addressing the situation when data tables consist of a header row that represents semantic relationships between concepts which may be symbolic concepts or quantities associated with units. [10] proposes a method to discover semantic relations between concepts. Our purpose is different: the semantic annotation of a data table consists in (i) recognizing the semantic relations defined in the OTR and represented in the data table; (ii) instantiating each recognized relation in each row of the data table, that is identifying their values in each row. Our final objective is to integrate in the same 'schema', Web data tables. The work of [11] can be considered as a sub-task of ours as they focus on the recognition of quantities in columns of the tables.

[1] http://www.w3.org/TR/2004/REC-owl-features-20040210/

The model of an OTR proposed in this paper is dedicated to the task of n-ary relations annotation in Web data tables. In the OTR, a clear separation is done between conceptual aspects and terminological ones. The conceptual part represents the semantic expressed by concepts while the terminological part allows one to define the terminology and its variations (multilingual, synonyms, abbreviations) denoting the concepts. The terminological part of the OTR allows one to improve the semantic annotation of data tables in a multilingual context thanks to the synonyms and abbreviations management. Moreover, this clear distinction between conceptual and terminological aspects and the management of unit conversions allow one to improve the querying of the data warehouse. As a matter of fact, since the data are annotated with concepts of the OTR, their querying can also be performed thanks to these concepts without worrying about the terminological variations and unit conversions (see [12,13] for preliminary works).

The structure of this paper is as follows. We first present the OTR in Section 2, with its conceptual and terminological parts. Then, in Section 3 a semantic annotation method of data tables guided by this OTR is presented. We finally conclude and present our future work in Section 4.

2 Modeling of the Ontological and Terminological Resource (OTR)

Since the modeling of the OTR is dedicated to the task of n-ary relations annotation in Web data tables, we present in Figure 1 an example of a semantic annotation of a Web data table extracted from a scientific paper in food science.

Cells of a data table contain terms (e.g. *MFC film A*) denoting symbolic concepts (e.g. *Packaging Material*) or numerical values (e.g. 3), often followed by a unit of measurement (e.g. ml m-2 day-1). Usually a data table represents semantic relationships between concepts which may be symbolic concepts or quantities

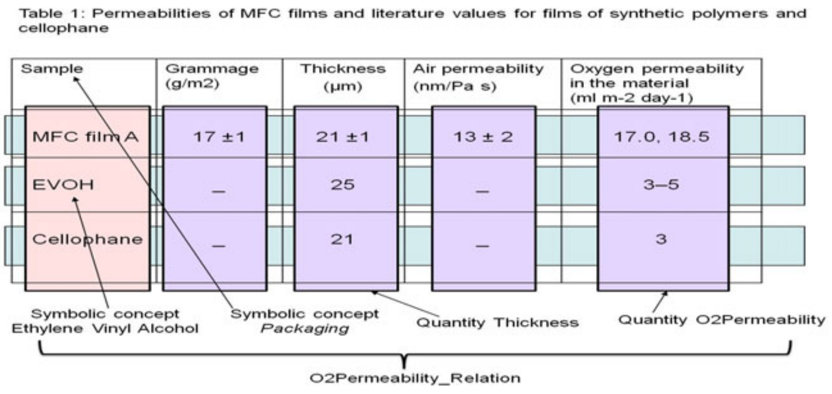


Table 1: Permeabilities of MFC films and literature values for films of synthetic polymers and cellophane

Sample	Grammage (g/m2)	Thickness (μm)	Air permeability (nm/Pa s)	Oxygen permeability in the material (ml m-2 day-1)
MFC film A	17 ±1	21 ±1	13 ± 2	17.0, 18.5
EVOH	–	25	–	3–5
Cellophane	–	21	–	3

Symbolic concept Ethylene Vinyl Alcohol
Symbolic concept *Packaging*
Quantity Thickness
Quantity O2Permeability
O2Permeability_Relation

Fig. 1. Example of a Web data table

characterized by units. The semantic annotation of a data table consists in recognizing the relations represented by the data table, which suppose to recognize symbolic concepts but also quantities and units. In the data table from Figure 1, the semantic relation *O2Permeability_Relation* which represents oxygen permeability for a food packaging material given its thickness, temperature and humidity has been partially recognized: the symbolic concept *Packaging* has been recognized in the first column, the quantity *Thickness* in the third column and the quantity *O2Permeability* in the last column, but the quantities *Temperature* and *Relative_Humidity* have not been recognized.

The OTR used for the semantic annotation of data tables should contain symbolic concepts, quantities and associated units, semantic relations linking symbolic concepts and quantities. Figure 2 presents an excerpt of our OTR in food science domain: it has a conceptual component, the ontology, and a terminological component, the terminology. We first present, in Subsection 2.1, the conceptual component of the OTR and, in Subsection 2.2, its terminological component. We modeled the conceptual and the terminological component of our OTR using the OWL2-DL[2] model.

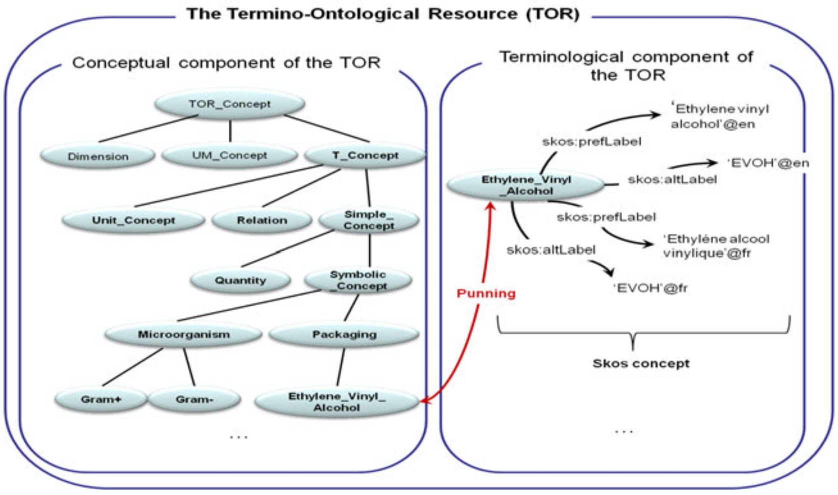


Fig. 2. An excerpt of the OTR in food science domain

2.1 Conceptual Component of the OTR

The conceptual component of the OTR is composed of two main parts: on the one hand, a generic part, commonly called core ontology, which allows the representation of the structure of the ontology and is dedicated to the n-ary relations annotation task in data tables and, on the other hand, a specific part, commonly called domain ontology, which depends on the domain of interest. Our OTR is generic because it allows n-ary relation to be instantiated in data tables for any application.

[2] http://www.w3.org/TR/owl2-overview/

Figure 3 presents the generic part of our OTR which does not depend on a domain of interest. There are three categories of generic concept: *Dimension*, *T_Concept* and *UM_Concept*. The generic concept *Dimension* represents dimensions that allow quantities and unit concepts (e.g. *Temperature*, *Length*, *Time*) to be classified. The generic concept *T_Concept* contains concepts to be recognized in data tables (in their cells, columns and rows) and are of three kinds: *Relation*, *Simple_Concept* or *Unit_Concept*. It is called *T_Concept* for *Terminological Concept*, because as detailed in Subsection 2.2, it contains concepts having one or several associated terms from the terminological component. The generic concept *UM_Concept* contains concepts which are used to manage units of measurement, especially conversions between units of measurement.

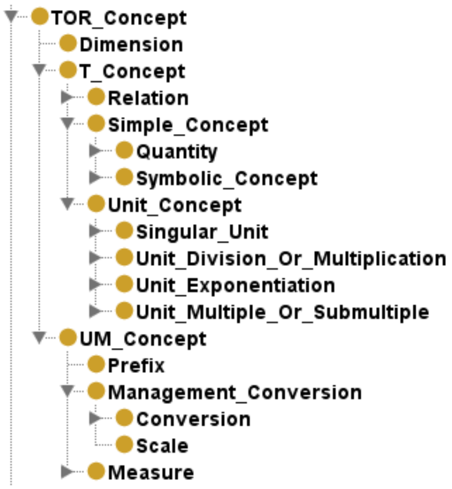


Fig. 3. The generic part of the OTR

The specific part of the OTR allows the representation of all concepts which are specific to a domain of interest. They appear in the OTR as sub concepts of the generic concepts. In OWL, all the concepts are represented by OWL classes, which are hierarchically organized by the *subClassOf* relationship and are pairwise disjoints.

We detail below the three kinds of the generic concept *T_Concept* with an example of specific sub concepts in food science domain and we present the management of conversions between units of measurement.

Generic concept *Simple_Concept*: Simple concepts include symbolic concepts (*Symbolic_Concept*) and quantities (*Quantity*).

1. *Symbolic_Concept*: A symbolic concept is characterized by its label (i.e. a term composed of one or more words), defined in the terminological part of the OTR, and by its hierarchy of possible values.

Example 1. Figure 4 presents an excerpt of the symbolic concepts hierarchy in food science domain. The specific symbolic concepts are sub-concepts of the generic concept *Symbolic_Concept*. For example, *Food_Product* and *Cereal* are two specific symbolic concepts, *Cereal* is a kind of *Food_Product*. The food science domain OTR contains 4 distinct hierarchies of specific symbolic concepts:

– *Food_Product* which has more than 500 sub concepts,
– *Microorganism* which has more than 150 sub concepts,
– *Packaging* which has more than 150 sub concepts, and
– *Response* which has three sub concepts: *growth*, *absence of growth* and *death*, which represent possible responses of a micro-organism to a treatment.

Let us notice that we could not reuse pre-existing terminologies for food products as AGROVOC[3] (from FAO - Food and Agriculture Organisation of the United Nations) or Gems-Food[4] (from WHO - World Health Organisation), because those terminologies are not specific enough compared to the one founded in our corpus in food science (respectively only 20% and 34% of common words).

2. *Quantity*: A quantity is characterized by its label, defined in the terminological part of the OTR, a set of units, which are sub concepts of the unit concept *Unit_Concept*, a dimension, which is sub concept of the dimension concept *Dimension*, and eventually a numerical range. An OWL object property *hasUnitConcept* associates a quantity with a set of unit concepts: it has for domain the generic concept *Quantity* and for range the generic concept *Unit_Concept*. An OWL object property *hasDimension* associates a quantity with a dimension: it has for domain the generic concept *Quantity* and for range the generic concept *Dimension*. We use the numerical restrictions of OWL2 (i.e. *minInclusive* and *maxInclusive*) to represent the maximal and minimal values associated with a quantity.

   Example 2. Figure 5 presents an excerpt of quantities in food science domain. The specific quantities, such as *PH*, *Permeability* or *Relative_Humidity*, are sub-concepts of the generic concept *Quantity*. The food science domain OTR contains 22 quantities. Figure 6 shows that the specific quantity *Relative_Humidity* can be expressed using the unit *Percent* or the unit *One*, which indicates dimensionless quantity, and it is restricted to the numerical range [0, 100].

Generic concept *Unit_concept*: A unit concept represents a unit of measurement. It is characterized by its label, defined in the terminological part of the OTR, a dimension and eventually by conversions. Our units classification relies on the International System of Units[5]. There exist several ontologies dedicated

[3] http://aims.fao.org/website/AGROVOC-Thesaurus
[4] http://www.who.int/foodsafety/chem/gems/en/
[5] http://www.bipm.org/en/si/

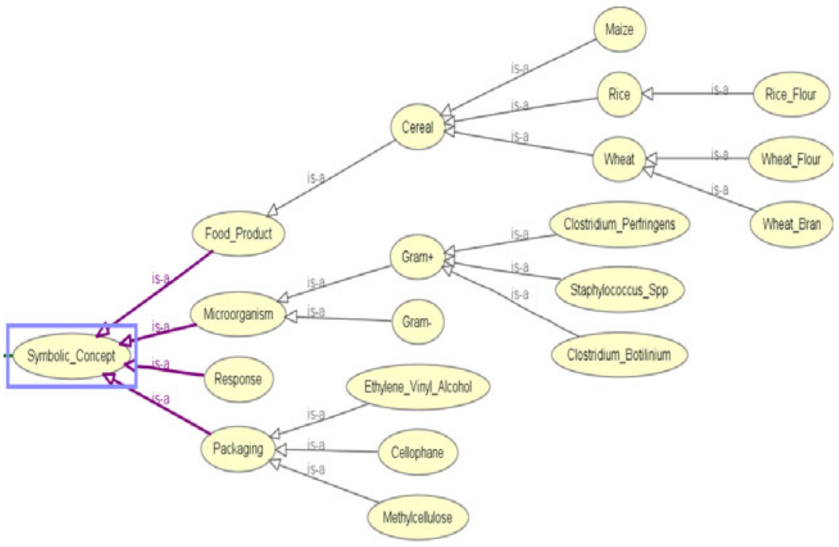


Fig. 4. An excerpt of the symbolic concepts hierarchy in food science domain

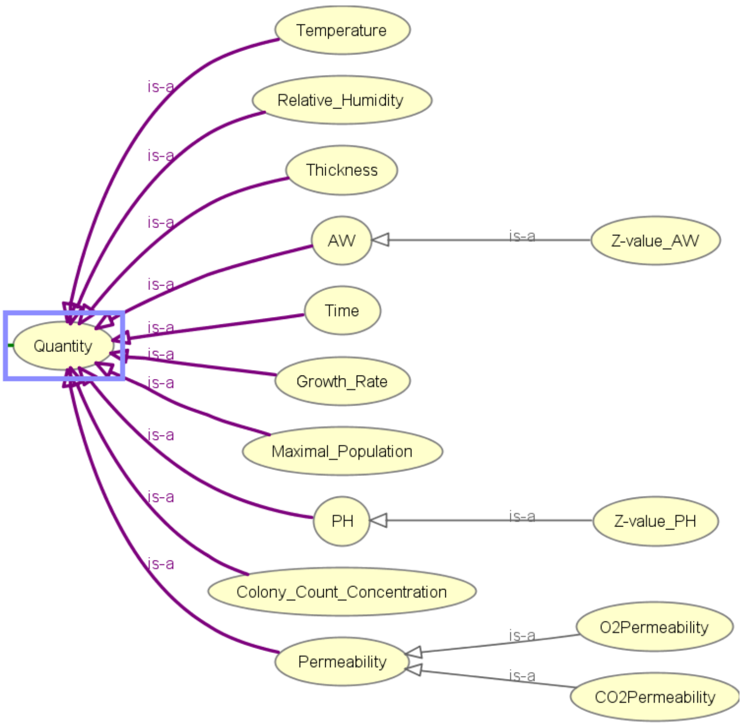
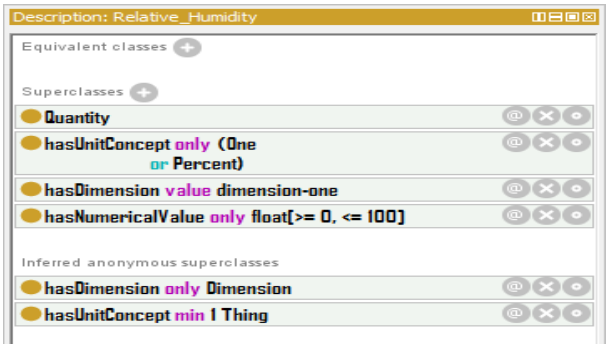


Fig. 5. An excerpt of the quantities in food science domain


Description: Relative_Humidity

Equivalent classes

Superclasses

Quantity

hasUnitConcept only (One or Percent)

hasDimension value dimension-one

hasNumericalValue only float[>= 0, <= 100]

Inferred anonymous superclasses

hasDimension only Dimension

hasUnitConcept min 1 Thing

Fig. 6. The specific quantity *Relative_Humidity*

to quantities and associated units (OM[6], OBOE[7], QUDT[8], QUOMOS, ...). We learned from these ontologies how to structure units into the ontological component of the OTR, then we added their associated terms into the terminological component, and, finally, we defined some specific units for our domain of interest. For instance, in food science domain, the ontologist has added units such as ppm[9] or CFU/g[10].

Example 3. Figure 7 presents an excerpt of the unit concepts hierarchy in food science domain. Specific concepts, such as *Day* (d), *Square_Metre* (m^2), *Micrometre* (µm) or *Cubic_Centimetre_By_25_Micrometre_Per_Square_Metre_Per_Day_Per_Atmosphere* ($cm^3 25\ \mu m/m^2/d/atm$) appear as sub concepts of one of the generic concepts *Singular_Unit*, *Unit_Exponentiation*, *Unit_Multiple_Or_Submultiple* or *Unit_Division_Or_Multiplication*. The concept *Measure* is used to represent components of units of measurement which are written in the form of a constant multiplied by a unit (e.g. 25 µm). The concept *Prefix* is used to represent constant values defined in the International System of Units (e.g. *Micro*(µ)).

Generic concept *Relation*: The concept *Relation* allows a n-ary relationship between simple concepts to be represented. A relation is characterized by its label, defined in the terminological part of the OTR, and by its signature (i.e. the set of simple concepts which are linked by the relation). The signature of a relation is defined by a domain and a range. The range is limited to only one simple concept, called *result concept*, while the domain contains one or several simple concepts, called *access concepts*. The restriction of the range to only one result concept is justified by the fact that, in a data table, a relation often represents a semantic n-ary relationship between simple concepts with only one

[6] http://www.wurvoc.org/vocabularies/om-1.8/
[7] http://marinemetadata.org/references/oboeontology
[8] http://www.qudt.org/
[9] Parts per million. ppm is a unit of concentration often used when measuring levels of pollutants in air, water, body fluids, etc.
[10] Colony-forming units per gram. Colony-forming units (CFU) is a measure of viable bacterial or fungal numbers in microbiology.

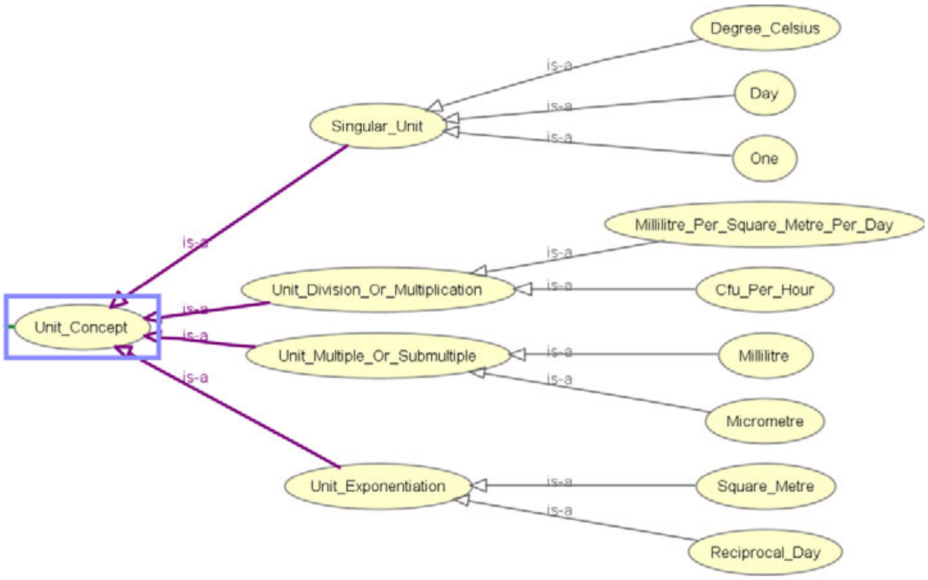


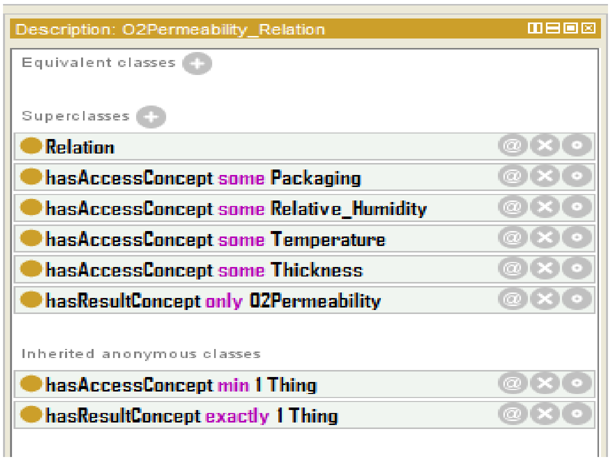
Fig. 7. An excerpt of the unit concepts hierarchy in food science domain

result, such as an experimental result with several measured parameters. If a data table contains several result columns, it is then represented by as many relations as it has results. As suggested in [14], a n-ary relation is represented in OWL by a class associated with the access concepts of its signature via the OWL object property *AccessConcept* and the result concept of its signature via the OWL functional object property *ResultConcept*.

Example 4. Figure 8 presents the specific relation *O2Permeability_Relation* which has for access concepts the specific symbolic concepts : *Packaging*, *Relative_Humidity*, *Temperature* and *Thickness* and for result concept the specific quantity *O2Permeability*. It represents oxygen permeability for a packaging material given its thickness, temperature and humidity. The food science domain OTR contains 16 relations.

Management of conversions between units of measurement : As pointed out in Section 1 the modeling of our OTR, dedicated to the task of data tables integration, has been guided by the construction of a data warehouse opened on the Web. In order to load and query the data warehouse and to be able to use data in decision models, we will need to automatically convert numerical data. We define the generic concept *Conversion*, sub concept of the generic concept *UM_Conversion* (see Figure 3), which is associated with units of measurement through the property *hasConversion*.

In this paper, we consider conversions between units of measurement which can be modeled by the following equation: $v_t = (v_s + o) * s$, where v_t is the value expressed in the target unit, v_s is a value expressed in a source unit, o is


Description: O2Permeability_Relation

Equivalent classes

Superclasses

Relation

hasAccessConcept some Packaging

hasAccessConcept some Relative_Humidity

hasAccessConcept some Temperature

hasAccessConcept some Thickness

hasResultConcept only O2Permeability

Inherited anonymous classes

hasAccessConcept min 1 Thing

hasResultConcept exactly 1 Thing

Fig. 8. The specific relation *O2Permeability_Relation*

the offset, and s is the scale. A lot of conversions between units of measurement can be done using a conversion factor (the scale) as those published by the US National Institute of Standards and Technology[11]. Conversions between units of measurement for temperatures require to introduce an additional offset (see for instance http://en.wikipedia.org/wiki/Fahrenheit).

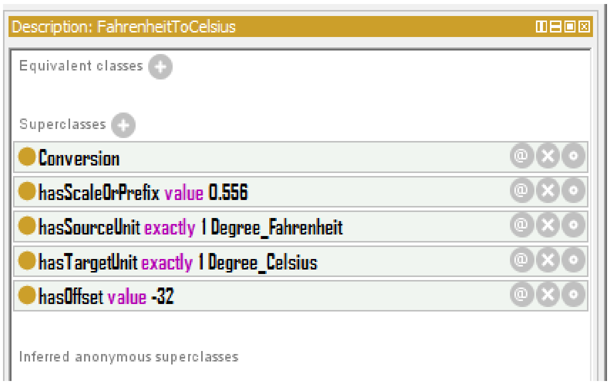
Let us illustrate the management of conversions between units of measurement through one example.

Example 5. To convert a temperature value expressed in Fahrenheit into Celsius, we use the following formula: $v_{\circ C} = (v_{\circ F} - 32) \times \frac{5}{9}$. To do this, we define the class *FahrenheitToCelsius*, detailed in Figure 9, as a subclass of the class *Conversion*, where the class *Degree_Fahrenheit* is a subclass of the generic concept *Singular_unit*.

2.2 Terminological Component of the OTR

The terminological component represents the terminology of the OTR: it contains the set of terms of the domain of interest. As mentioned in Section 2.1, at least one term of the terminological component is associated with each sub concept of the generic concept *T_Concept*; for example the term *Ethylene vinyl alcohol* is associated with the concept *Ethylene_Vinyl_Alcohol*. Each sub concept of *T_Concept* has one associated label (i.e. a sequence of words defined in a given language) called preferred label, in a given language, but it may also be characterized by alternate labels, which correspond to synonyms or abbreviations, this in different languages. Those labels associated with a given concept are used in the semantic annotation of data tables: they are compared with the terms

[11] http://ts.nist.gov/WeightsAndMeasures/Publications/appxc.cfm


Description: FahrenheitToCelsius

Equivalent classes

Superclasses

Conversion

hasScaleOrPrefix value 0.556

hasSourceUnit exactly 1 Degree_Fahrenheit

hasTargetUnit exactly 1 Degree_Celsius

hasOffset value -32

Inferred anonymous superclasses

Fig. 9. An example of conversion for temperature

present in the data tables (in their cells, columns' titles, table title) in order to be able to recognize the concepts of the OTR (more precisely the sub concepts of *T_Concept*) that the data tables represent.

We propose to associate labels with each sub concept of *T_Concept* using the labeling properties of SKOS[12] (Simple Knowledge Organization Scheme) which is a W3C recommendation and is based on RDF language. Using the meta-modeling from OWL2-DL, each sub concept of *T_Concept* is defined at the same time as an OWL class and as an instance of the class *OWL SKOS : Concept* (see the example from Figure 2). More precisely, the same identifier (URI) is associated with its OWL class representation and its individual representation, using the punning[13] metamodeling capabilities available in OWL2-DL. Therefore, each sub concept of *T_Concept* is defined, on the one hand, as an OWL class in order to be instantiated in rows of a data table and, on the other hand, as an instance in order to allow one to compare its associated labels with the terms present in the data tables.

Example 6. In food science OTR, the symbolic concept *Ethylene_Vinyl_ Alcohol* was defined both as an OWL class in order to be able to instantiate it in a data table, and as an instance of the class *OWL SKOS : Concept* allowing to represent its terminological characteristics by using the labeling properties *prefLabel* and *altLabel* of SKOS. The concept *Ethylene_Vinyl_Alcohol* is then defined as follows:

```
<owl:Class rdf:ID="Ethylene Vinyl Alcohol">
 <rdfs:subClassOf rdf:resource="#Packaging"/>
 <rdf:type rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
 <skos:prefLabel xml:lang="en">Ethylene vinyl alcohol</skos:prefLabel>
 <skos:altLabel xml:lang="en">EVOH</skos:altLabel>
 <skos:prefLabel xml:lang="fr">Ethylène alcool vinylique</skos:prefLabel>
 <skos:altLabel xml:lang="fr">EVOH</skos:altLabel>
</owl:Class>
```

[12] http://www.w3.org/TR/skos-reference/
[13] http://www.w3.org/TR/owl2-new-features/#F12:_Punning

3 Using the OTR to Annotate and Query Data Tables

We propose to illustrate the relevance of our modeling choices made in our OTR, by using it in the semantic annotation method of Web data tables proposed in [9]. We briefly present, in this section, the main steps of this method (see Figure 10) and its adaptation to our OTR through an example: the annotation of the data table from Figure 1.

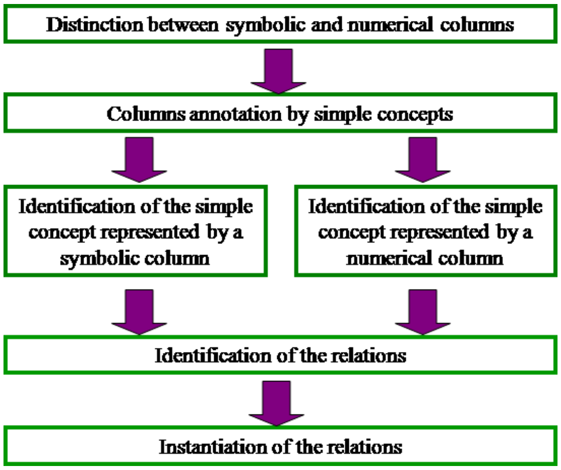

Distinction between symbolic and numerical columns
Columns annotation by simple concepts
Identification of the simple concept represented by a symbolic column
Identification of the simple concept represented by a numerical column
Identification of the relations
Instantiation of the relations

Fig. 10. The main steps of the semantic annotation method of a table driven by an OTR

Distinction between symbolic and numerical Columns. The first step of the semantic annotation method is to distinguish between symbolic and numerical columns, by counting occurrences of numerical values and terms found in each column and by using some of the knowledge described in the OTR (e.g. terms denoted unit concepts are accepted in numerical columns).

Example 7. In Table 1 from Figure 1 the first column is identified as a symbolic column: it contains only terms. The other columns are identified as numerical ones: they contain only numerical values or ranges of numerical values (e.g. 21 ± 1).

Columns annotation by simple concepts. Once a column has been classified as a symbolic column or as a numerical one, this step identifies which simple concept of the domain OTR corresponds to the column. In order to annotate a column *col* with a simple concept *c*, two scores are combined: the score of the simple concept *c* for the column *col* according to the column title, and the score of the simple concept *c* for the column *col* according to the column content. Because the main objective of the semantic annotation method is to identify which relations of the OTR are represented in a Web data table, only the simple concepts of the OTR which appear in the signatures of the relations of the OTR are considered; those simple concepts are called *simple target concepts*.

Example 8. The domain of food science is composed of four symbolic target concepts : *Food_Product*, *Microorganism*, *Packaging* and *Response* (see Figure 4), and it is composed of 22 target quantities of which an excerpt is presented in Figure 5.

Identification of the simple target concept represented by a symbolic column. The annotation of a symbolic column by a symbolic target concept relies on a comparison between the terms present in each cell of the column and the list of preferred and alternative labels associated with the concepts which belong to the hierarchy of each symbolic target concept of the OTR. We use the cosine similarity measure [15] to compare terms which have been previously transformed into a vector of lemmatized words using WordNet.

Example 9. Let us consider the first column of Table 1 which was identified as a symbolic column. The following steps allows to annotate this first column with the symbolic target concept *Packaging*.

– The second cell of the first column which contains the term *EVOH* is annotated with the symbolic target concept *Packaging* because this term is among the labels denoting the symbolic concept *Ethylene_Vinyl_Alcohol*, which is a sub concept of the symbolic target concept *Packaging* (see Example 6 and Figure 2). The score of a symbolic target concept TC for a cell $cell$ is computed as the maximum for all the cosine similarity measures between the terms t_i denoting TC or one of its sub concepts in the OTR and the term contained into the cell:

$$score_{Cell}(cell, TC) = max_i sim(t_i, content(cell)).$$

Then the score of the symbolic target concept *Packaging* for the second cell of the first column is 1: $score_{Cell}(cell_{21}, Packaging) = max_i sim(t_i, EVOH) = 1$.
– The scores of the symbolic target concept *Packaging* for the other cells (i.e. *MFC film A* and *Cellophane*) of this column are also computed and are equal to 1.
– The score of a symbolic target concept TC for a column col according to the column content is

$$score_{ContentCol}(col, TC) = \frac{\#(\text{cells of } col \text{ annotated by } TC)}{\#(\text{cells of } col)}.$$

Then the score of the symbolic target concept *Packaging* according to the content of the first column is $score_{ContentCol}(col_1, Packaging) = \frac{3}{3} = 1$.
– In the same way, the scores of the others target concepts of the food science OTR for the first column according to its content are computed and are equal to 0.

- Furthermore, the score of a symbolic target concept TC for a column *col* according to the column title is

$$score_{TitleCol}(col, TC) = max_i sim(t_i, content(title(col)))$$

where terms t_i denote TC and $content(title(col))$ is the content of the title of the column. $score_{TitleCol}(col_1, Packaging) = max_i sim(t_i, Sample) = 0$ because the symbolic target concept *Packaging* is not denoted by a label syntactically close to the term *Sample.*
- The final score of a symbolic target concept TC for a column *col* is defined by

$$score_{Col}(col, TC) =$$
$$= 1 - (1 - score_{TitleCol}(col, TC))(1 - score_{ContentCol}(col, TC)).$$

Therefore the final score of the symbolic target concept *Packaging* for the first column of Table 1 is : $score_{Col}(col_1, Packaging) = 1-(1-score_{TitleCol}(col_1, Packaging))(1 - score_{ContentCol}(col_1, Packaging)) = 1 - (1 - 0)(1 - 1) = 1$. Since all the others symbolic target concepts have a null final score for the first column, the first column of Table 1 is annotated by the symbolic target concept *Packaging.*

Identification of the simple target concept represented by a numerical column. The annotation of a numerical column *ncol* by a target quantity (called TQ in the following) relies on the units present in the column and its numerical values, which must be compatible with the numerical range of the target quantity.

Example 10. Let us consider the last column of Table 1 which was identified as a numerical column. The following steps allows to annotate this last column with the target quantity *O2Permeability*:

- First the annotation method identifies in the column the unit concept *Millilitre_Per_Square_Metre_Per_Day* because the label *ml m-2 day-1* is an alternative label for this concept. In food science OTR, this unit concept is only associated with the quantity *O2Permeability.* As the score for a unit *unit* is defined by:

$$score_{Unit}(unit) = \frac{1}{\#\{TQ|unit \in hasUnitConcept(TQ)\}}$$

then, $score_{Unit}(Millilitre\_Per\_Square\_Metre\_Per\_Day) = \frac{1}{1} = 1$.
- As *ml m-2 day-1* is the only unit in the last column and the target quantity *O2Permeability* has no numerical range defined in the OTR, then the score of the target quantity *O2Permeability* for this column according to its content is : $score_{ContentNCol}(col_5, O2Permeability) = 1$.
- In the same way, the scores of the other target quantities of the food science OTR for the column according to the column content are computed and are equal to 0.

– Furthermore, the score of a target quantity TQ for a column $ncol$ according to the column title is

$$score_{TitleNCol}(ncol, TQ) = max_i sim(t_i, content(title(ncol)))$$

where terms t_i denote TQ and $content(title(ncol))$ is the content of the title of the column.
$score_{TitleNCol}(col_5, O2Permeability)$ = $max_i sim(t_i,$ *Oxygen permeability in the material*) = *sim*(*Oxygen permeability, Oxygen permeability in the material*) = 0.816 because the target quantity *O2Permeability* is, in particular, denoted by the English preferred label *Oxygen permeability*.
Besides, the score of the target quantity *CO2Permeability* for the column according to the column title is also computed as follows: $score_{TitleNCol}(col_5,$ $CO2Permeability)$ = *sim*(*Carbon Dioxide permeability, Oxygen permeability in the material*) = 0.408.
– The final score of a target quantity TQ for a column $ncol$ is

$$score_{NCol}(ncol, TQ) =$$

$$= 1 - (1 - score_{TitleNCol}(ncol, TQ))(1 - score_{ContentNCol}(ncol, TQ)).$$

Therefore, the final scores of the target quantities *O2Permeability* and *CO2-Permeability* for the last column of Table 1 are:
$score_{NCol}(col_5, O2Permeability) = 1 - (1 - 0.816)(1 - 1) = 1,$
$score_{NCol}(col_5, CO2Permeability) = 1 - (1 - 0.408)(1 - 0) = 0.408.$
Since all the others target quantities have a null final score for the last column, the last column of Table 1 is annotated by the target quantity *O2Permeability* which has the best score.

Using the same method, we also determine that the third column of Table 1 is annotated by the target quantity *Thickness*. Furthermore, since no target quantity from the OTR has been identified to annotate the second and the fourth column of Table 1, they are annotated by the generic concept *Quantity*.

Identification of the relations. Once all the columns of a data table have been annotated by concepts of the domain OTR, the fourth step of the annotation method consists in identifying which relations of the OTR are represented in the data table. In order to annotate a data table by a relation, two scores are combined: the score of the relation for the data table according to the data table title and the score of the relation for the data table according to the data table content. This second score depends on the proportion of simple concepts in the relation's signature which were represented by columns of the data table, the result concept recognition being required. Let us notice that a data table can be annotated by several relations.

Example 11. According to Examples 9 and 10, the first column of Table 1 has been annotated by the symbolic target concept *Packaging*, the third column by the target quantity *Thickness*, the last column by the target quantity

O2Permeability and the second and fourth columns by the generic concept *Quantity*. The data table can be annotated by the relation *O2Permeability_Relation* of the OTR, which has the target quantity *O2Permeability* as result concept. The score of a relation *Rel* according to its signature is:

$$score_{Signature}(table, Rel) = \frac{\#(\text{recognized concept in } Rel \text{ signature})}{\#(\text{concepts in } Rel \text{ signature})}.$$

The score of the relation *O2Permeability_Relation* for Table 1 according to its signature (see Example 4) is: $score_{Signature}(Table_1, O2Permeability\_Relation) = \frac{3}{5} = 0.6$.

The score of a relation *Rel* for the data table *table* according to the data table title is computed as the maximum cosine similarity measure between the terms t_i denoting *Rel* in the OTR and the data table title.

$$score_{TitleTable}(table, Rel) = max_i sim(t_i, content(title(table))).$$

As the title of Table 1 is *Permeabilities of MFC films and literature values for films of synthetic polymers and cellophane*, the score of relation *O2Permeability_Relation* is: $score_{TitleTable}(Table_1, O2Permeability\_Relation) = 0.35$.

The final score of a relation *Rel* for a data table *table* is

$$score_{Table}(table, Rel) =$$

$$= 1 - (1 - score_{TitleTable}(table, Rel))(1 - score_{Signature}(table, Rel)).$$

Therefore, the final score of the relation *O2Permeability_Relation* for the data table Table 1 is: $score_{Table}(Table_1, O2Permeability\_Relation) = 1-(1-0.35)(1-0.6) = 0.74$.

Since no other relation of the OTR has the target quantity *O2Permeability* as result concept, Table 1 is annotated by the relation *O2Permeability_Relation*.

Instantiation of the relations: The fifth and last step of the annotation method (see Figure 10) is the instantiation of each identified relation for each row of the considered data table. The instantiation of a relation relies on the instantiation of the symbolic target concepts and the target quantities which belong to its signature and were represented by columns of the data table (see [9] where this step of the annotation method is detailed, but where an ad-hoc ontology is used).

Example 12. The instantiation of the relation *O2Permeability_Relation* for the second row of Table 1 is represented by the set of pairs {(original value, recognized simple target concept : (annotation values[14]))} :
{(EVOH, Packaging: (Ethylene Vinyl Alcohol)), (25 μm, Thickness: (value: 25, unit concept: Micrometre)), (3-5 ml m-2 day-1, O2Permeability : (interval of values: [3, 5], unit concept: Millilitre_Per_Square_Metre_Per_Day)) }.

[14] See [9] for more detail

4 Conclusion

We have proposed, in this paper, a model for an Ontological and Terminological Resource (OTR) dedicated to the task of n-ary relations annotation in Web data tables. In this OTR, a special effort has been made to distinguish the generic part (core ontology) dedicated to the n-ary relation annotation task for any application from the specific part dedicated to a given application domain. The OTR is implemented using the latest W3C recommendations (OWL2-DL and SKOS). We have demonstrated the relevance of this model by applying it in a semantic annotation method of Web data tables proposed in [9]. Consequently, the OTR model can be reused for any application domain, redefining only its specific part, to annotate n-ary relations from Web data tables. Since the data are annotated with concepts of the OTR, their querying can also be performed using these concepts without worrying about the terminological variations and unit conversions.

As future work we want to explore three directions: i) Concerning the implementation issue, we need first to upgrade our hole system, as shown in Section 3 for the annotation module, allowing then to perform new experimental results. It suppose: to transform our 3 ad-hoc ontologies into 3 OTR, and to add synonyms and terms in other languages into theirs terminological parts, to add new data tables in other languages into the corpora of data tables to be annotated, and to adapt the reference set for new evaluations. ii) An other short term perspective is to propose a method for the management of the OTR evolution in order to improve the quality of the annotation of Web data tables. This method should be able to take into account different types of changes: changes explicitly required by ontologists, changes due to an alignment with external ontologies, changes required after analyzing of the OTR to fulfill ontology quality assurance criteria and changes required after manual validation of new annotations. iii) A long term exciting perspective is to extend our model to be able to annotate n-ary relations not only in data tables extracted from Web documents but also using the information available in the plain text of those documents.

Acknowledgments. Financial support from the French National Research Agency (ANR) for the project Map'OPT is gratefully acknowledged.

References

1. Gruber, T.: Ontology. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1963–1965. Springer, US (2009)
2. Cimiano, P., Buitelaar, P., McCrae, J., Sintek, M.: Lexinfo: A declarative model for the lexicon-ontology interface. J. Web Sem. 9(1), 29–51 (2011)
3. McCrae, J., Spohr, D., Cimiano, P.: Linking Lexical Resources and Ontologies on the Semantic Web with Lemon. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 245–259. Springer, Heidelberg (2011)

4. Declerck, T., Lendvai, P.: Towards a standardized linguistic annotation of the textual content of labels in knowledge representation systems. In: LREC, European Language Resources Association (2010)
5. Reymonet, A., Thomas, J., Aussenac-Gilles, N.: Modelling ontological and terminological resources in OWL DL. In: OntoLex 2007, ISWC Workshop (2007)
6. Roche, C., Calberg-Challot, M., Damas, L., Rouard, P.: Ontoterminology - a new paradigm for terminology. In: Dietz, J.L.G. (ed.) KEOD, pp. 321–326. INSTICC Press (2009)
7. Reymonet, A., Thomas, J., Aussenac-Gilles, N.: Ontology based information retrieval: an application to automotive diagnosis. In: International Workshop on Principles of Diagnosis (DX 2009), pp. 9–14 (2009)
8. Hignette, G., Buche, P., Dibie-Barthélemy, J., Haemmerlé, O.: An Ontology-Driven Annotation of Data Tables. In: Weske, M., Hacid, M.-S., Godart, C. (eds.) WISE Workshops 2007. LNCS, vol. 4832, pp. 29–40. Springer, Heidelberg (2007)
9. Hignette, G., Buche, P., Dibie-Barthélemy, J., Haemmerlé, O.: Fuzzy Annotation of Web Data Tables Driven by a Domain Ontology. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 638–653. Springer, Heidelberg (2009)
10. Lynn, S., Embley, D.W.: Semantically Conceptualizing and Annotating Tables. In: Domingue, J., Anutariya, C. (eds.) ASWC 2008. LNCS, vol. 5367, pp. 345–359. Springer, Heidelberg (2008)
11. van Assem, M., Rijgersberg, H., Wigham, M., Top, J.: Converting and Annotating Quantitative Data Tables. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 16–31. Springer, Heidelberg (2010)
12. Buche, P., Haemmerlé, O.: Towards a Unified Querying System of Both Structured and Semi-Structured Imprecise Data Using Fuzzy View. In: Ganter, B., Mineau, G.W. (eds.) ICCS 2000. LNCS, vol. 1867, pp. 207–220. Springer, Heidelberg (2000)
13. Buche, P., Dibie-Barthélemy, J., Chebil, H.: Flexible Sparql Querying of Web Data Tables Driven by an Ontology. In: Andreasen, T., Yager, R.R., Bulskov, H., Christiansen, H., Larsen, H.L. (eds.) FQAS 2009. LNCS, vol. 5822, pp. 345–357. Springer, Heidelberg (2009)
14. Noy, N., Rector, A., Hayes, P., Welty, C.: Defining n-ary relations on the semantic web. W3C working group note, http://www.w3.org/TR/swbp-n-aryRelations
15. van Rijsbergen, C.J.: Information Retrieval. Butterworth (1979)

Inductive Learning of Disjointness Axioms

Daniel Fleischhacker and Johanna Völker*

KR & KM Research Group, University of Mannheim, Germany
{daniel,johanna}@informatik.uni-mannheim.de

Abstract. The tremendous amounts of linked data available on the web are a valuable resource for a variety of semantic applications. However, these applications often need to face the challenges posed by flawed or underspecified representations. The sheer size of these data sets, being one of their most appealing features, is at the same time a hurdle on the way towards more accurate data because this size and the dynamics of the data often hinder manual maintenance and quality assurance. Schemas or ontologies constraining, e.g., the possible instantiations of classes and properties, could facilitate the automated detection of undesired usage patterns or incorrect assertions, but only few knowledge repositories feature schema-level knowledge of sufficient expressivity. In this paper, we present several approaches to enriching learned or manually engineered ontologies with disjointness axioms, an important prerequisite for the applicability of logical approaches to knowledge base debugging. We describe the strengths and weaknesses of these approaches and report on a detailed evaluation based on the DBpedia dataset.

Keywords: Linked Data, Ontology Learning, OWL, Data Mining.

1 Motivation

The success of the Open Linked Data initiative and the fast growing number of knowledge repositories on the web have paved the way for the development of various semantic mashups and applications. However, these applications often need to face the challenges posed by flawed or underspecified knowledge representations. While the redundancy of structured data on the web can help to compensate for many of those problems, even applications based on lightweight knowledge representations benefit from more accurate semantic data in repositories such as DBpedia, Freebase as well as in other, domain-specific knowledge bases.

Ontologies, or generally speaking schemas, constraining the possible instantiations of classes and properties are a valuable means to improve the quality of linked data sets. They can enable the automated detection of undesired usage patterns or incorrect assertions. However, only few knowledge repositories feature schema-level knowledge of sufficient expressivity, and thus Auer and Lehmann [3] demanded that "algorithms and tools have to be developed for improving the structure, semantic richness and quality of Linked Data". In particular, disjointness axioms would be useful to enable more

* Johanna Völker is financed by a Margarete-von-Wrangell scholarship of the European Social Fund (ESF) and the Ministry of Science, Research and the Arts Baden-Württemberg.

expressive query answering as well as the detection of logical inconsistencies indicating potential modeling errors. Hitzler and van Harmelen [13] point out, for instance, that annotating linked data with ontologies that include class disjointness can help to solve the object reconciliation problem, i.e., the discovery of identical individuals across data sets. The following example taken from the DBpedia data set illustrates further benefits potentially provided by the addition of disjointness axioms:[1]

Dirk_Bouts dbo:nationality *Netherland*
Netherland rdf:type dbo:Book
dbo:nationality rdfs:domain dbo:Person

Considering these RDF triples, we find the dbo:nationality relation linking *Dirk_Bouts* to the DBpedia resource *Netherland*, which is explicit asserted to be of type dbo:Book in DBpedia. This error stems from a spelling mistake in the Wikipedia infobox of the article about Dirk_Bouts, as the value of the `nationality` property is given as `[[Netherland]]` and thus points to the wrong Wikipedia article. Note that detecting the error by logical means would not only require a properly specified range restriction of the dbo:nationality relation,[2] but it would also demand for the existence of a disjointness axiom:

dbo:nationality rdfs:range dbo:Country
dbo:Country owl:disjointWith dbo:Book

These two statements would allow us to infer that *Netherland* must be a country, and hence can *not* be a book since dbo:Book and dbo:Country are declared disjoint. This would be a logical contradiction to the previous rdf:type assertion. Since such logical contradictions can be spotted by automated means [21,18], such a manual or automatic enrichment of ontologies with further axioms can provide the maintainers of a knowledge base with valuable pointers to potential problems. In practice, of course, it will not always be clear whether the newly added or any of the existing statements are incorrect and thus should be removed – especially, if the former were generated automatically they should therefore be associated with provenance information such as certainty values to increase the efficiency of manual or automated debugging (e.g., [19]).

In this paper, we present a set of novel inductive methods for automatically enriching ontologies with disjointness axioms which could be used to extend previous approaches to inducing schema-level knowledge from knowledge bases. These methods exhibit three characteristics that we consider essential, especially for ontology generation from linked data: They are scalable enough to work on large RDF repositories such

[1] For the sake of brevity, we assume `http://dbpedia.org/resource/` to be the default namespace and use the prefix dbo: for abbreviating the URI of the DBpedia ontology (`http://dbpedia.org/ontology/`).

[2] There are several approaches to the automatic acquisition of domain or range restrictions from text or linked data including, for example, early work by Mädche and Staab [16]. Note that it is also possible to induce these types of axioms from linked data, e.g., by association rule mining [25]. For a comprehensive overview of approaches to mining structured web data, see Stumme et al. [23].

as DBpedia, and robust insofar as they can tolerate a certain number of incorrect assertions. Moreover, these methods provide users and applications with a certainty value for each of the generated disjointness axioms.

The remainder of this paper is structured as follows. After giving a brief overview of related work (cf. Section 2), we describe the three approaches that we developed in order to enrich ontologies with disjointness axioms (Section 3), including statistical correlation analysis as well as two algorithms for mining association rules. In Section 4, we report on a comparative evaluation of these approaches, before concluding with a summary and an outlook to future work (cf. Section 5).

2 Related Work

The work presented in this paper relates to previous approaches in the field of ontology learning and the automated generation of disjointness axioms. It also follows a variety of automated approaches supporting the evaluation and maintenance of knowledge bases on the web.

Early work on learning class disjointness has been done by Haase and Völker [11], who suggested an unsupervised method for mining disjointness axioms from natural language text. Their method based on lexico-syntactic patterns was later incorporated into the LeDA framework for acquiring disjointness by supervised machine learning [26]. Unlike the approaches suggested by this paper, LeDA does not crucially hinge on the existence of class membership assertions. However, the learning algorithm needs to be trained on a manually created set of disjointness axioms and requires various kinds of background knowledge, whose fit to the data set at hand can be assumed to have a huge impact on the quality of the generated disjointness axioms.

Disjointness axioms are also generated by more general ontology learning approaches. Especially, inductive methods based on inductive logic programming [14] or formal concept analysis [4] are applicable to the task at hand, but so far none of them has been evaluated with regard to the quality of acquired disjointness axioms. This also holds for methods based on association rule mining as proposed, e.g., by Völker and Niepert [25]. Their approach referred to as statistical schema induction is complemented by a simple heuristic for introducing disjointness axioms into schemas automatically generated from RDF data, that assumes non-overlapping classes with more than a hundred individuals to be disjoint – a rough rule of thumb that cannot be expected to work in the general case.

A more well-known heuristic for introducing disjointness axioms into existing ontologies has been proposed by Schlobach [20]. His approach known as *semantic clarification* aims to make logical debugging techniques applicable to lightweight ontologies. It relies on the "strong disjointness assumption" [9], which postulates disjointness among sibling classes, as well as on the pinpointing technique for discovering and fixing the causes of logical incoherence. A similar strategy was later adopted by Meilicke et al. [17], who showed that automatically generated disjointness axioms can facilitate the detection of incorrect correspondences between classes in different lightweight ontologies.

Particularly related to our approaches is recent work by Lehmann and Bühmann [15], who developed ORE, a tool for repairing different types of modeling errors in ontologies. It uses the DL-Learner framework [14], which has been shown to scale up to large knowledge bases [12], in order to enrich ontologies by class expressions automatically induced from existing instance data. Inconsistencies or incoherences resulting from this enrichment serve as indicators for modeling flaws in the knowledge base. While their approach does not focus on disjointness axioms, they emphasize the usefulness of negation in debugging knowledge bases, it would be worthwhile investigating ways to integrate our methods into ORE.

In Section 4, we will take a closer look at LeDA as well as the strong disjointness assumption and how their performance compares to the methods presented in this paper.

3 Methods for Learning Disjointness

In this section, we present three approaches to enriching the schemas of large knowledge repositories with disjointness axioms. First, after briefly introducing the syntax and semantics of disjointness axioms in the Web Ontology Language OWL, we describe an approach that is based on statistical correlation analysis (cf. Section 3.1). We then elaborate on the use of association rule mining techniques for learning disjointness, and outline the ideas underlying two alternative methods supporting the discovery of negative association rules (see Section 3.2). For a detailed comparison of these methods with state-of-the-art approaches to generating disjointness axioms, see Section 4.

Both RDF Schema[3] and the Web Ontology Language (OWL)[4] are standards proposed by the W3C for expressing schema-level knowledge on the web. RDFS allows for modeling lightweight schemas consisting of classes (or concepts), individuals (or instances), as well as properties (or relations) connecting these individuals. It also provides means to express class subsumption, domain and range restrictions of properties, and equality of individuals, for example, but the RDFS standard does not contain a negation operator or other means to model negative knowledge. Additional expressivity required, e.g., by reasoning-based applications, is offered by OWL, which extends RDFS by additional constructs, such as class and property disjointness.[5] Note that there is an RDF-based serialization for every OWL ontology. For the sake of brevity, however, we will henceforth use the description logic notation for talking about disjointness axioms.

Using class disjointness, it is possible to state that two classes *cannot* have any common individuals according to their intended interpretation, i.e., that the intersection of these classes is necessarily empty in all possible worlds. For example, the OWL axiom

$$\mathsf{Person} \sqsubseteq \neg\mathsf{Plant}$$

or equivalently, $\mathsf{Person} \sqcap \mathsf{Plant} \sqsubseteq \bot$, expresses the fact that nothing can be both a person and a plant, as the intersection of these classes is subsumed by $\bot$ and hence necessarily empty. This does *not* imply, however, that if two classes do not have any common

[3] http://www.w3.org/TR/rdf-schema/
[4] http://www.w3.org/TR/owl2-overview/
[5] Property disjointness has been added as part of OWL 2 which has become a W3C recommendation on October 27, 2009.

individuals in a particular knowledge base, they are meant to be disjoint. This is because of the Open World Assumption holding in OWL as well as in RDFS, which states that knowledge not explicitly (or implicitly) said to be true is not treated as being false, but as being unknown. For this reason, the assertions $\mathsf{Person}(Tom)$ and $\mathsf{Plant}(Tom)$ would not cause a logical contradiction, and thus would not necessarily be recognized as a modeling error by a mere reasoning-based approach, unless we add an axiom stating that Person and Plant are disjoint classes.

Ontologies using disjointness may exhibit two kinds of logical contradiction: incoherence and inconsistency. An ontology is incoherent if it contains a class C which is not satisfiable, i.e., which is empty according to every possible interpretation. An incoherent class could be introduced in an ontology by the following axioms.

$$\begin{aligned} \mathsf{Human} &\sqsubseteq \mathsf{Animal} \\ \mathsf{Human} &\equiv \mathsf{Person} \\ \mathsf{Person} &\sqsubseteq \neg\mathsf{Animal} \end{aligned}$$

Since all humans are defined to be animals and the classes Person and Human are defined to be equivalent, the disjointness between Person and Animal renders the class Person unsatisfiable. Incoherences are rarely introduced on purpose, since in most real-world application scenarios there is little reason for creating a named class that is not intended to contain individuals. When it comes to logical inference over an ontology, incoherent classes mainly have a local effect as they are subsumed by and at the same time disjoint to all other classes.

Inconsistency usually has a more significant impact on the practical usefulness of an ontology for reasoning-based applications. An ontology being inconsistent means that there is no model for this ontology which, e.g., could be caused by an individual belonging to a non-satisfiable class. Since a fact can be inferred from an ontology iff it is valid in all models of the ontology (and this trivially holds if no model exists), inconsistencies prevent most standard reasoners from performing meaningful inference. In an ontology containing the axioms from our incoherence example, the axiom

$$\mathsf{Person}(Kim)$$

would lead to an inconsistent ontology because the unsatisfiable class Person is assigned the instance Kim.

Both inconsistencies and incoherences can be detected by automated means for inconsistency diagnosis. Often, the results of a diagnosis, a set of axioms that together cause a logical contradiction, indicate some kind of modeling error in the knowledge base, that might have remained unnoticed if the ontology had not turned inconsistent. For this reason, a certain level of logical expressivity introduced, e.g., by axioms containing negation operators, is desirable as it facilitates the occurrence of logical contradictions whenever classes, individuals or properties are not used in agreement with their intended semantics. However, many of the available linked data repositories such as DBpedia only use lightweight schemas – either in formats not supporting class disjointness, like RDFS, or just not stating disjointness though possible format-wise. Thus, it is not possible to apply logical debugging methods to these semantic resources. To enable more elaborate maintenance and quality assurance on linked data, we thus explored different ways to automatically enrich lightweight schemas by disjointness axioms.

For all approaches which we present in the following, we assume the data to be represented as depicted in Table 1. In this case, we have one row per instance contained in the data set and one column per class mentioned in the dataset resp. the corresponding ontology. For each instance, all existing rdf:type assertions are marked by a 1 in the corresponding column while 0 means that the instance is not assigned to a certain class. This table is a structured representation of a so-called *transaction database* which we will introduce more formally later-on in Section 3.2.

Table 1. Excerpt from a transaction database for the DBpedia dataset

IRI	Place	City	Person	OfficeHolder
Berlin	1	1	0	0
Charles_Darwin	0	0	1	0
Eiffel_Tower	1	0	0	0
John_F._Kennedy	0	0	1	1
Golden_Gate_Bridge	1	0	0	0

The approaches that we present in the remainder of this section are based on the paradigm of statistical inductive learning, i.e., they are based on the assumption that schema-level knowledge can be derived from an analysis of existing class membership (or rdf:type) assertions – either by association rule mining or the computation of statistical correlation values. In this respect, our approaches bear some resemblance with previous work on concept learning in description logics [14]. Even several features used in the LeDA framework, including the taxonomic overlap in terms of existing or automatically acquired individuals, can be considered inductive or extensional. In Section 4, we will take a closer look at LeDA as the only existing framework for learning disjointness, and how it compares to the new, purely inductive methods.

3.1 Correlation

The first approach we applied for generating disjointness axioms is measuring the correlation between class rdf:type assertions. Correlation coefficients are commonly used to rate the strength of linear relationships between two value sequences. One widely known correlation coefficient is Pearson's correlation coefficient which is also used in a similar fashion by Antonie and Zaïane [2] who combine it with association rules and use it for filtering.

For our experiments, the values we consider for computing the correlation coefficient are the values stating which instances belong to a given class. Given two classes C_1 and C_2, we take the sequences formed by the appropriate columns of our transaction database and compute the correlation between these two sequences. For each instance, there exist four possibilities of class combinations which are shown in the following table, e.g., n_{10} is the number of transactions containing class C_1 but not class C_2.

	C_1	$\neg C_1$	
C_2	n_{11}	n_{01}	n_{*1}
$\neg C_2$	n_{10}	n_{00}	n_{*0}
	n_{1*}	n_{0*}	

For this specific variant, the Pearson correlation coefficient can be reduced to the so-called ϕ-coefficient given by

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1*}n_{0*}n_{*0}n_{*1}}}$$

Given the resulting correlation coefficient, we can assess the strength of the correlation between the occurrences of the classes. According to Cohen [8] the results of the Pearson correlation coefficient can be coarsely divided into the categories of strong correlation for absolute values larger than 0.5, medium correlation for absolute values in the range from 0.3 to 0.5 and small correlation for absolute values from 0.1 to 0.3. Absolute values of less than 0.1 are inexpressive.

Since disjointness of two classes means that both classes must not have any common instantiations, classes being clearly disjoint would lead to $\phi = -1.0$. In contrast, a ϕ-value of 1.0 would show two perfectly equivalent classes based on the set of instances. Thus, negative correlation values having a high absolute value give the most evidence for both classes being disjoint and can be considered as a confidence value for the validity of the corresponding disjointness axiom.

For the transaction database shown in Table 1 and the classes Place and OfficeHolder, we would get $\phi = \frac{-3}{\sqrt{24}} = -0.61$. From this strong negative correlation the correlation-based algorithm could propose a disjointness axiom between both classes using the absolute correlation value as confidence.

3.2 Association Rule Mining

The other two approaches, we evaluated for inductively learning disjointness are based on association rules. Association rules are implication patterns originally developed for large and sparse datasets such as transaction databases of international supermarket chains. A typical dataset in such a setting can have up to 10^{10} transactions (rows) and 10^6 attributes (columns). Hence, the mining algorithms developed for these applications are also applicable to the large data repositories in the open Linked Data cloud. Formally, the transaction database $D = (t_1, t_2, \ldots, t_n)$ contains transactions $t_j \subseteq I$ where $I = \{i_1, i_2, \ldots\}$ is the set of all possible items. As already described, each individual in the data set has a corresponding transaction which contains items representing classes the individual belongs to.

To mine association rules from such a transaction database the first step is to generate frequent itemsets contained in this database. For this purpose, there are multiple algorithms, the Apriori algorithm [1] being the most commonly used one. Frequent itemsets are thereby identified by having a support value greater than a specified minimum support threshold whereas support is defined as

$$supp(X) = |\{t_i \in D : X \subseteq t_i\}|$$

In some cases, the support value is also defined relatively to the number of transactions contained in the database. Given these frequent itemsets, it is possible to generate association rules of the form $A \Rightarrow B$ where $A \subseteq I$ and $B \subseteq I$ are both itemsets. The confidence for a certain association rule is given by

$$conf(A \rightarrow B) = \frac{supp(A \cup B)}{supp(A)}$$

Thus, it shows the conditional probability of an itemset B occurring in a transaction given the occurrence of an itemset A.

In this work, we do not want to generate rules like $A \rightarrow B$, which would to a degree resemble subsumption and equivalence relations in ontologies, but negative association rules where either A or B is negated like $A \rightarrow \neg B$. It is important to note that negative association rules, despite being similar to logical implications, do not capture the same logical meaning of implication or, in our special case, of disjointness. Association rules are not definitive rules but there may be some transactions which violate their proposition. This fact is partly represented by the confidence values which incorporate the fraction of transactions transgressing the association rule.

Naïve Negative Association Rule Mining. Mining negative association rules poses different requirements to the association rule mining algorithms since typically there are many more items *not* contained in an itemset than items contained in it. In the typical problem domain of association rules, the number of possible items is too large to apply regular algorithms on the data set. However, our problem has a much more limited problem space. In our domain, we usually have to deal with a few hundreds or thousands of items (i.e., classes defined in the provided schema) whereas a typical association rule application deals with itemset sizes of up to 10^6. Therefore, we are able to apply standard algorithms to our problem of mining negative association rules sometimes referred to as the naïve approach of mining negative association rules [2,24]. To do this, for all classes the corresponding complements are also added to the transaction database and all instances not belonging to a class are marked as belonging to its respective complements. Applying this transformation to the transaction database depicted in Table 1, we get a the transaction database shown in Table 2. Using this approach, the standard association rule mining methods generate not only positive items but also negative ones and thus negative association rules. The example transaction database also illustrates one major shortcoming of this approach. Because of the addition of complement classes, we lose much of the original database's sparsity which greatly increases the space required to store such a transformed database.

Table 2. Transaction database containing materialized class complements

IRI	Place	City	Person	OfficeHolder	¬Place	¬City	¬Person	¬OfficeHolder
Berlin	1	1	0	0	0	0	1	1
Charles_Darwin	0	0	1	0	1	1	0	1
Eiffel_Tower	1	0	0	0	0	1	1	1
John_F._Kennedy	0	0	1	1	1	1	0	0
Golden_Gate_Bridge	1	0	0	0	0	1	1	1

As an example, we consider the itemset $\{\mathsf{Place}, \neg\mathsf{OfficeHolder}\}$ which reaches a support value of 3 because these items are contained in the transactions for $Berlin$, $Eiffel\_Tower$ and $Golden\_Gate\_Bridge$. For the negative association rule $\mathsf{Place} \rightarrow \neg\mathsf{OfficeHolder}$, we thus get a confidence value of

$$\frac{supp(\mathsf{Place}, \neg\mathsf{OfficeHolder})}{supp(\mathsf{Place})} = \frac{3}{3} = 1$$

Negative Association Rule Mining. Typically, the datasets association rule mining is performed on are much larger than the ones used in our case. The number of items contained in one transaction is much smaller than the number of items not contained in a transaction and thus the number of frequent itemsets gets an enormous boost by such a naïve transformation of the transaction database. Since this greatly reduces the usefulness of standard positive association rules mining algorithms for mining negative association rules, there are several works regarding the development of special negative association rule mining algorithms [2]. In this work, we apply the negative association rule algorithm proposed by Zhang and Zhang [28]. Because of the complexity of this algorithm, we only give a short overview on it.

The negative association rule mining approach does not rely on materializing the item complements but instead searches for infrequent positive itemsets. Because of the sparsity of the original transaction database there is an almost exponential number of such infrequent positive itemsets. To reach a well enough performance for such an approach, pruning the search space is an important concern. After generating infrequent itemsets, Zhang and Zhang prune those itemsets which are not considered interesting given the minimum interest level. In this context, an itemsets is called *interesting* if its support exceeds the expected level of support. Based on the remaining negative itemsets, they define an approach to create all possible negative association rules using the probability ratio of each association rule as the corresponding confidence.

4 Experiments

The three approaches described in Section 3 are expected to perform differently on the task of generating disjointness axioms. Thus, in order to assess the quality of the produced disjointness axioms, we did an extensive comparison. As a state of the art approach for creating this type of axioms, we also included the LeDA framework [26] into our comparison which implements a supervised machine learning approach to acquiring disjointness axioms. Additionally, we compared the results to the heuristic proposed by Schlobach [20] (cf. Section 2). All of these methods have been tested against a gold standard of manually created disjointness axioms.

4.1 Setting

For our experiments,[6] we used the data set provided by the DBpedia project [5]. We applied the aforementioned approaches to a set of transaction tables containing the

[6] All data used in our experiments is available from `http://code.google.com/p/gold-miner/`.

DBpedia data as of December 2010. In addition, we used the DBpedia ontology[7] version 3.5.1.

Implementation. The values for the *correlation-based approach* were computed by our own implementation that determines the Pearson correlation coefficient as described in Section 3.1 for each pair of two classes that are stated to be of the types of some resources in DBpedia. The resulting correlation coefficients were used as confidence values for the corresponding disjointness axioms.

For mining the *association rules*, we used the Apriori miner system version 5.39 by Borgelt and Kruse [6], while the transaction tables for the naïve way of mining were generated by our own implementation. It is worthwhile mentioning that the materialization of the transaction database with representations for the respective complements of all contained classes, increased the size of the transaction data file from 13 MB to about 1.7 GB. On this materialized data file, we did multiple runs of Apriori miner with different settings for minimum support, minimum confidence and minimum interest. In addition, to actually make the amount of data manageable for the mining system, we had to limit the computation of frequent itemsets to sets consisting of two elements – no real limitation as we anyway only consider disjointness axioms relating two atomic classes.

Since we were unable to find a publicly available implementation of *negative association rule mining* suitable for our experiments, we implemented the algorithm described by Zhang and Zhang [28] ourselves and used it in the experiments described in the following.[8] This mining approach is also influenced by the parameters minimum support, minimum confidence and minimum interest.

Configuration of LeDA. In order to evaluate our methods against LeDA [26], a state-of-the-art framework for learning disjointness axioms from heterogeneous sources of evidence, we updated the latest LeDA release as follows. We replaced the original KAON2-based implementation of the ontology backend by a version that uses Pellet and the Manchester OWL API. For our experiments, we used the set of features that performed well in recent experiments by Meilicke et al. [17]. These features are listed in Table 3. The naive bayes classifier of LeDA was trained on the upper module of the PROTON ontology[9] using the partial disjointness gold standard created by Völker et al. After this training phase, we applied the resulting classifier to the DBpedia ontology.

The background ontologies for the respective features have been automatically generated by using the Text2Onto tool [7]. We extracted these ontologies from a corpus of Wikipedia articles which we automatically assembled by downloading the articles corresponding to the class names contained in the ontologies. During this process only an automatic transformation from the camel-case naming style used in the ontologies to a sequence of single words has been done but no further disambiguation steps were applied. Ontologies automatically generated by Text2Onto contain instances and thus provide LeDA with a limited amount of instance data, e.g., for determining the in-

[7] http://dbpedia.org/Ontology

[8] The negative association rule mining system is integrated into the gold-miner system available from http://code.google.com/p/gold-miner/.

[9] http://proton.semanticweb.org/

stance overlap of two classes. Since the PROTON ontology used during training does not contain instance data, the instance-based features of LeDA could only be applied to the background ontologies but not to the original ontologies, which also resembles the setup employed by Meilicke et al.

Table 3. Features of the classification model used by LeDA [26]

Feature	Description
f_{doc}	Lexical context similarity (Wikipedia)
$f_{jaro-winkler}$	Label similarity (JaroWinkler)
$f_{levenshtein}$	Label similarity (Levenshtein)
$f_{overlap_c}$	Taxonomic overlap wrt. subclasses
$f^b_{overlap_c}$	Taxonomic overlap wrt. subclasses (learned ontology)
$f^b_{overlap_i}$	Taxonomic overlap wrt. instances (learned ontology)
f_{path}	Semantic distance
f^b_{path}	Semantic distance (learned ontology)
f_{prop}	Object properties
f_{qgrams}	Label similarity (QGrams)
f_{sub}	Subsumption
f^b_{sub}	Subsumption (learned ontology)
f_{wn_1}	WordNet similarity (Patwardhan-Pedersen v1)
f_{wn_2}	WordNet similarity (Patwardhan-Pedersen v2)

Thresholds. For the association rule mining approaches, we chose an absolute support value of 10 transactions and a confidence value of 0.8 for both approaches. We did not set an interest threshold, i.e., our minimum interest value was 0. For LeDA we just used the default thresholds given by the type of classifier and thus considered those classes as being disjoint for which the disjointness has been determined with a confidence of at least 0.5. On the correlation approach, we applied the threshold values 0.05 and 0.005. Even if these values are both beneath the limits for meaningful correlations we nevertheless chose these after some first experiments since the results seemed to be promising.

Baselines. In addition to these automatic approaches, we considered two more baselines and a gold standard of disjointness axioms manually added to the DBpedia ontology. For our gold standard, we asked several experienced ontology engineers to collaboratively enrich the DBpedia ontology with a full set of disjointness axioms. The first baseline approach is based on the *strong disjointness assumption* [9] used by *Schlobach*, thus it considers all siblings as being disjoint. This baseline approach reaches an accuracy of 92% with respect to our gold standard. The second baseline approach is a simple *majority* approach. Since the vast majority of all possible class pairs is considered as disjoint in the gold standard, the majority vote would be setting all pairs to disjoint. Regarding our gold standard, this method would reach an accuracy of 91%.

4.2 Handling Logical Inconsistency and Incoherence

After generating the list of disjointness axioms, we ordered the axioms by descending confidences and enriched the DBpedia ontology incrementally always adding the axiom with the highest confidence to the ontology. After each addition, the ontology is checked for coherence and, by also considering the instances contained in the DBpedia dataset, for consistency. If an incoherence or inconsistency is detected the lastly added axiom is removed from the ontology and pruned from the axiom list.

Note that checking for consistency and coherence is a non-trivial task. Due to the high number of instances contained in the DBpedia dataset it was not possible to use standard OWL reasoners like Pellet [22] which are not suited for such large repositories. Therefore, we applied a two-step approach regarding the detection of incoherence and inconsistence in the enriched ontology. Incoherence of a class means that this class is not satisfiable, i.e., it has to be empty to conform to the schema. Incoherence is not directly related to the actual existence of instances asserted to this or any other class. Thus, it is sufficient to only consider the raw schema and ignore the instance data which allows to query Pellet for satisfiability of each single class with good performance. To also preserve consistency, instances have to be considered either way. To do this, we combined results returned by the Pellet reasoner and from SPARQL queries sent to a Virtuoso RDF database containing the instances for the DBpedia ontology. We identified three different cases which he had to deal with to catch possible inconsistencies while enriching the ontology. We present these three cases in the following, all of them are checked after adding a new disjointness axiom to the ontology.

The most obvious case of inconsistency is caused by individuals assigned to the classes defined to be disjoint. This type of inconsistency is detectable by a non-empty result set for the SPARQL query[10]

```
SELECT ?x WHERE { ?x a <ClassURI1> . ?x a <ClassURI2> . }
```

Furthermore, the ontology is inconsistent if the SPARQL query

```
SELECT ?x WHERE { ?x a <ClassURI1> . ?x <PropURI> ?y . }
```

returns a non-empty result while the ontology entails $<\mathsf{ClassURI2}>\mathsf{rdf}:\mathsf{domain}<\mathsf{PropURI}>$. Eventually, the third type of inconsistencies is detected if there are individuals fulfilling the query

```
SELECT ?x WHERE { ?x a <ClassURI1> . ?y <PropURI> ?x . }
```

while the ontology entails $<\mathsf{ClassURI2}>\mathsf{rdf}:\mathsf{range}<\mathsf{PropURI}>$. It is worth noting that these patterns should be able to detect most inconsistencies which can occur in the DBpedia ontology by adding disjointness axioms. This is caused by the fact that the ontology only employs a limited set of the features provided by OWL, e.g., it does not contain cardinality restrictions.

4.3 Creation of a Gold Standard

We created a gold standard of disjointness axioms on the DBpedia ontology for our experiments. During its manual creation the human ontology engineers came across some points which led to discussions. In the following, we describe some of these problems.

[10] $<\mathsf{ClassURI1}>$ resp. $<\mathsf{ClassURI2}>$ are used as placeholders for the actual URIs of the classes the disjointness is stated for.

Table 4. Statistics for automatically generated axioms without materialization (compared to materialized gold standard)

	Total Axioms over Threshold	Correct Axioms	Precision
LeDA	62,115	57,422	0.92
Correlation (0.005)	10,218	9,562	0.94
Correlation (0.05)	424	418	0.99
Naive ARM	14,994	13,623	0.91
Negative ARM	58	58	1.00

One point which turned out to be problematic is the distinction between different functions of buildings, e.g., *shopping mall* and *airport*. Since there are several airport buildings which also include shopping malls, it would be reasonable not to set both classes as disjoint. On the other side, the actual functions of buildings are intensionally disjoint because the airport functionality has no relation to the shopping mall function. In this specific case, the way of modeling used in the DBpedia ontology favors the first interpretation because the building may at the same time serve both functions, airport and shopping mall, without the possibility to divide both parts. Thus, the subclasses of *building* have been modeled to be pairwise disjoint.

Many problems during the creation of the gold standard were similar to these ones. Another example is the differentiation between *continents* and *countries* since there is, e.g., Australia which could be considered to be both a continent and a country. In this case, we opted for a way of modeling that takes into consideration the difference between the continent being a landmass and a country being an organizational unit possibly located at a landmass.

For some classes, their intension was not clear by just using the information available from the ontology itself. In these cases, the contents of the corresponding Wikipedia articles were used to clarify the respective notions and for some cases the extension of the specific class, i.e., the DBpedia instances assigned to these classes, were considered for clarification purposes.

4.4 Results and Discussion

Analysis of Inconsistencies. After creating the gold standard, we materialized all disjointness axioms inferable from the gold standard. Based on this set of axioms, we performed an analysis of all axioms contained in the different automatically generated axiom lists. This way, we were able to compute a precision[11] regarding the actually inferable set of axioms in the gold standard. The results are shown in Table 4. For our computations, we assumed our gold standard to be complete, i.e., classes not being implicitly or explicitly stated as disjoint are considered *not* to be disjoint. Furthermore, since it is not meaningful to compute recall and accuracy on this level without any semantics of ontologies included, we left out these measures.

While enriching the DBpedia ontology with disjointness axioms our greedy approach raised various inconsistency errors caused by instances explicitly or implicitly asserted to both classes of a disjoint class pair. Usually, there are two sources of such errors. The

[11] For definitions of precision, recall and accuracy, see [27].

Table 5. Incoherences and inconsistencies detected while adding axioms

	Axioms	Incoherences	Inconsistencies: Total	Range Conflict	Domain Conflict
Gold Standard	59,914	0	1,412	1,365	47
LeDA	62,115	0	1,837	1,759	78
Correlation (0.005)	10,218	0	2,068	2,028	40
Correlation (0.05)	424	0	262	257	5
Naive ARM	14,994	230	1,025	980	45
Negative ARM	176	0	70	69	1

first one is simply that the disjointness axiom determined by the automatic learning process is incorrect, the other one are incorrect, explicit or implicit rdf:type assertions. As described in Section 4.2, we only apply a heuristic approach for inconsistency detection which checks for three different kinds of inconsistencies. Table 5 shows the number of the different contradiction types.

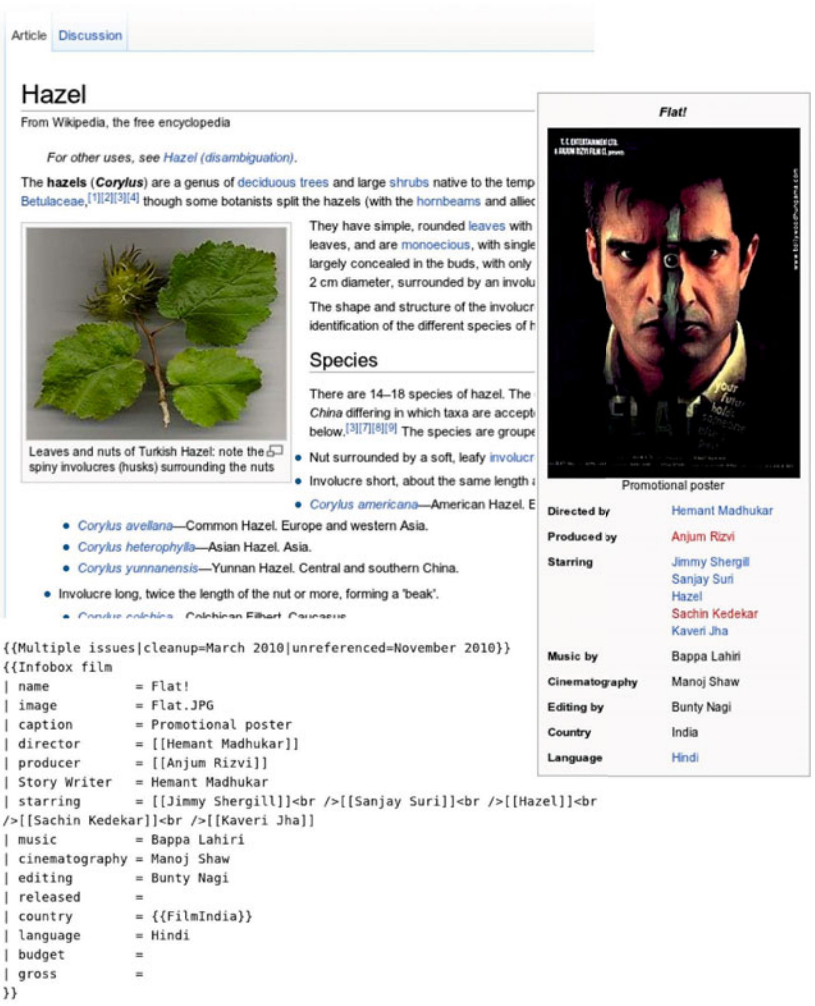
As we are able to see from the numbers, most errors are caused by range restrictions. This means that the range assertion of a specific property allows to infer a class assertion for an instance which conflicts with the generated disjointness axiom. According to our exploration of the inconsistencies, the most common error type is a *disambiguation error*. An example for this kind of error is revealed by the obviously correct disjointness axiom between the classes *Person* and *Plant* raised by the naive association rule mining approach. While adding this disjointness axiom to the DBpedia ontology, a range conflict for the property *starring* has been detected by our enriching process. A more elaborate examination of the DBpedia data showed that the range of *starring* is set to be the class *Person* but there exists a property *starring* between the instances *Flat!* (which is an Indian movie) and *Hazel* (the tree). This is caused by an incorrect cast reference in the infobox of the corresponding Wikipedia page of *Flat!*.[12] The correct reference would point to the actress *Hazel Crowney*.[13] Thus, the learned disjointness axiom helped to detect an error in Wikipedia which led to a wrong DBpedia information.

Semantically Founded Evaluation. For the automatically generated axiom lists, we not only computed the precision for the raw lists (see Table 4) but also the measures of precision, recall and accuracy on their materializations. For this purpose, we enriched the DBpedia ontology by the automatically generated disjointness axioms using our greedy approach and afterwards computed a list of all disjointness axioms inferable from the ontology. The most important results are shown in Table 6.

The best performance regarding the automatically generated disjointness axioms is achieved by LeDA using the schema-based approach. Even if it did not reach the highest precision value, the recall and accuracy values are the highest of all automatic approaches. The precision with respect to the non-materialized list is at 0.92 which is quite high, though not as high as the correlation-based approaches. The main advantage of the correlation-based approaches is their high precision for certain thresholds but they suffer from a relatively low recall. To have a more detailed insight into the good performance of LeDA in this task, we also had a look at the feature usage by means of gain ratio evaluation of Weka. According to this analysis, the classifier mostly uses the

[12] `http://en.wikipedia.org/w/index.php?title=Flat!&oldid=409238040`

[13] `http://en.wikipedia.org/wiki/Hazel_Crowney`



Article Discussion

Hazel

From Wikipedia, the free encyclopedia

For other uses, see Hazel (disambiguation).

The **hazels** (***Corylus***) are a genus of deciduous trees and large shrubs native to the temp
Betulaceae,[1][2][3][4] though some botanists split the hazels (with the hornbeams and alliec

They have simple, rounded leaves with
leaves, and are monoecious, with single
largely concealed in the buds, with only
2 cm diameter, surrounded by an involu

The shape and structure of the involucr
identification of the different species of h

Leaves and nuts of Turkish Hazel: note the spiny involucres (husks) surrounding the nuts

Species

There are 14–18 species of hazel. The
China differing in which taxa are accept
below.[3][7][8][9] The species are groupe

- Nut surrounded by a soft, leafy involucr
- Involucre short, about the same length
  - *Corylus americana*—American Hazel. E
  - *Corylus avellana*—Common Hazel. Europe and western Asia.
  - *Corylus heterophylla*—Asian Hazel. Asia.
  - *Corylus yunnanensis*—Yunnan Hazel. Central and southern China.
- Involucre long, twice the length of the nut or more, forming a 'beak'.
  - *Corylus colchica*—Colchican Filbert. Caucasus

Flat!

Promotional poster

Directed by	Hemant Madhukar
Produced by	Anjum Rizvi
Starring	Jimmy Shergill Sanjay Suri Hazel Sachin Kedekar Kaveri Jha
Music by	Bappa Lahiri
Cinematography	Manoj Shaw
Editing by	Bunty Nagi
Country	India
Language	Hindi

```
{{Multiple issues|cleanup=March 2010|unreferenced=November 2010}}
{{Infobox film
| name           = Flat!
| image          = Flat.JPG
| caption        = Promotional poster
| director       = [[Hemant Madhukar]]
| producer       = [[Anjum Rizvi]]
| Story Writer   = Hemant Madhukar
| starring       = [[Jimmy Shergill]]<br />[[Sanjay Suri]]<br />[[Hazel]]<br
/>[[Sachin Kedekar]]<br />[[Kaveri Jha]]
| music          = Bappa Lahiri
| cinematography = Manoj Shaw
| editing        = Bunty Nagi
| released       =
| country        = {{FilmIndia}}
| language       = Hindi
| budget         =
| gross          =
}}
```

Fig. 1. Disambiguation error in Wikipedia infobox for movie *Flat!*

features $f_{overlap_c}$ (gain ratio of 0.59), f_{qgrams} (0.11) and $f^{b}_{overlap_c}$ (0.11). The other activated features only gave a maximum gain ratio of 0.02 or less. The negative association rule mining approach suffers from the same problem as the correlation-based one with respect to recall but in exchange reaches 1.0 for precision which makes it very suitable if high precision is more important than high recall.

Taking into consideration the feature usage of LeDA, we can conclude that it is very dependent on the available training data and its similarity to the actual data which has to be classified. This is the main advantage of the induction-based approaches and regarding the statistics both association rule mining approaches perform well putting their specific emphasis either on precision or recall. Thus, if there is no appropriate training

Table 6. Evaluation of generated axioms[14]

	Inferable Axioms	Correct Decisions	Correct Axioms	Precision	Recall	Accuracy
Gold Standard	59,914	-	-	-	-	-
Schlobach	65,006	60,597	60,597	0.93	0.92	0.92
Majority	66,049	59,914	59,914	0.91	1.00	0.91
LeDA	60,314	57,557	55,868	0.93	0.85	0.87
Correlation (0.005)	41,786	43,049	39,350	0.94	0.60	0.65
Correlation (0.05)	3,246	9,381	3,246	1.00	0.05	0.14
Naive ARM	47,358	49,173	45,198	0.95	0.68	0.74
Negative ARM	10,790	16,925	10,790	1.00	0.16	0.26

data available, these association rule mining-based approaches should be considered and chosen depending on the desired balance between precision and recall.

Regarding the baselines it is important to mention that while delivering some of the best results regarding precision, recall and accuracy, they are not suited for being used in general. For the majority approach, one has to determine first what is more common, disjointness or non-disjointness, which means creating a disjointness gold standard for the ontology. The strong disjointness assumption proposed by Schlobach is only this successful because for the DBpedia ontology the majority of siblings is in fact disjoint but in general there might be ontologies following other characteristics, e.g., for the *Person* subtree of the DBpedia ontology the assumption does not hold.

5 Conclusion and Outlook

In this paper, we presented a set of inductive methods for generating disjointness axioms from large knowledge repositories. We performed a comparative evaluation with the most commonly used methods and heuristics for generating disjointness axioms, and discussed the respective advantages of inductive, i.e. instance-based, and schema-based methods for learning disjointness. Our experiments indicate that it is possible to induce disjointness axioms from an existing knowledge base with an accuracy that is well enough to help detecting inconsistencies in datasets. This is particularly true if there is no appropriate training data available to use tools like LeDA. As we have also shown, we were able to identify various problems in DBpedia by adding the automatically generated axioms to the DBpedia ontology.

While further experiments with other data sets will be indispensable, we are confident that our methods will facilitate the development of more efficient means to supporting users of large RDF repositories in detecting and fixing potential problems in the data sets. Future work includes the integration of our methods with existing approaches to acquiring schema-level knowledge from linked data, e.g., based on inductive logical programming [12] or association rule mining [25]. The incremental induction of schemas including disjointness axioms could facilitate, for example, an automated synchronization of the DBpedia ontology with DBpedia Live[15] and immediate diagnoses whenever changes to the underlying Wikipedia articles are submitted. Logical inconsistencies provoked by the enrichment of learned or manually engineered schemas would

[14] Correct decisions = true positives + true negatives; correct axioms = true positives.

[15] `http://live.dbpedia.org`

need to be resolved by methods for consistent ontology evolution [10]. Finally, we will investigate the applicability of our approaches to the problem of learning property disjointness.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), pp. 487–499. Morgan Kaufmann (1994)
2. Antonie, M.-L., Zaïane, O.R.: Mining Positive and Negative Association Rules: An Approach for Confined Rules. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 27–38. Springer, Heidelberg (2004)
3. Auer, S., Lehmann, J.: Creating knowledge out of interlinked data. Semantic Web 1(1-2), 97–104 (2010)
4. Baader, F., Ganter, B., Sertkaya, B., Sattler, U.: Completing description logic knowledge bases using formal concept analysis. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), pp. 230–235. AAAI Press (2007)
5. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. Web Semantics 7(3), 154–165 (2009)
6. Borgelt, C., Kruse, R.: Induction of association rules: Apriori implementation. In: Proceedings of the 15th Conference on Computational Statistics (COMPSTAT), pp. 395–400. Physica Verlag (2002)
7. Cimiano, P., Völker, J.: Text2Onto. In: Montoyo, A., Muńoz, R., Métais, E. (eds.) NLDB 2005. LNCS, vol. 3513, pp. 227–238. Springer, Heidelberg (2005)
8. Cohen, J.: Statistical power analysis for the behavioral sciences, 2nd edn. Larwence Erlbaum, New Jersey (1988)
9. Cornet, R., Abu-Hanna, A.: Usability of expressive description logics – a case study in UMLS. In: Proceedings of the AMIA Annual Symposium, pp. 180–184 (2002)
10. Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A Framework for Handling Inconsistency in Changing Ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 353–367. Springer, Heidelberg (2005)
11. Haase, P., Völker, J.: Ontology Learning and Reasoning — Dealing with Uncertainty and Inconsistency. In: da Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) URSW 2005 - 2007. LNCS (LNAI), vol. 5327, pp. 366–384. Springer, Heidelberg (2008)
12. Hellmann, S., Lehmann, J., Auer, S.: Learning of OWL class descriptions on very large knowledge bases. International Journal on Semantic Web and Information Systems 5(2), 25–48 (2009)
13. Hitzler, P., van Harmelen, F.: A reasonable semantic web. Journal of Semantic Web 1(1-2), 39–44 (2010)
14. Lehmann, J.: DL-Learner: learning concepts in description logics. Journal of Machine Learning Research (JMLR) 10, 2639–2642 (2009)
15. Lehmann, J., Bühmann, L.: ORE - A Tool for Repairing and Enriching Knowledge Bases. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part II. LNCS, vol. 6497, pp. 177–193. Springer, Heidelberg (2010)
16. Mädche, A., Staab, S.: Discovering conceptual relations from text. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI), pp. 321–325. IOS Press (2000)
17. Meilicke, C., Völker, J., Stuckenschmidt, H.: Learning Disjointness for Debugging Mappings Between Lightweight Ontologies. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 93–108. Springer, Heidelberg (2008)

18. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging owl ontologies. In: Proceedings of the 14th International Conference on World Wide Web (WWW), pp. 633–640. ACM (2005)
19. Qi, G., Haase, P., Huang, Z., Ji, Q., Pan, J.Z., Völker, J.: A Kernel Revision Operator for Terminologies — Algorithms and Evaluation. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 419–434. Springer, Heidelberg (2008)
20. Schlobach, S.: Debugging and Semantic Clarification by Pinpointing. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 226–240. Springer, Heidelberg (2005)
21. Schlobach, S.: Diagnosing terminologies. In: Proceedings of the 20th National Conference on Artificial Intelligence (NCAI), vol. 2, pp. 670–675. AAAI Press (2005)
22. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics 5, 51–53 (2007)
23. Stumme, G., Hotho, A., Berendt, B.: Semantic web mining: State of the art and future directions. Journal of Web Semantics 4(2), 124–143 (2006)
24. Teng, W.G., Hsieh, M.J., Chen, M.S.: On the mining of substitution rules for statistically dependent items. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM), pp. 442–449. IEEE Computer Society (2002)
25. Völker, J., Niepert, M.: Statistical Schema Induction. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 124–138. Springer, Heidelberg (2011)
26. Völker, J., Vrandečić, D., Sure, Y., Hotho, A.: Learning Disjointness. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 175–189. Springer, Heidelberg (2007)
27. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann (2005)
28. Zhang, C., Zhang, S.: Association rule mining: models and algorithms. Springer, Heidelberg (2002)

Breaking the Deadlock: Simultaneously Discovering Attribute Matching and Cluster Matching with Multi-Objective Simulated Annealing

Haishan Liu and Dejing Dou

Computer and Information Science Department,
University of Oregon, Eugene, USA

Abstract. In this paper, we present a data mining approach to challenges in the matching and integration of heterogeneous datasets. In particular, we propose solutions to two problems that arise in combining information from different results of scientific research. The first problem, *attribute matching*, involves discovery of correspondences among distinct numeric-typed summary features ("attributes") that are used to characterize datasets that have been collected and analyzed in different research labs. The second problem, *cluster matching*, involves discovery of matchings between patterns across datasets. We treat both of these problems together as a multi-objective optimization problem. A multi-objective simulated annealing algorithm is described to find the optimal solution. The utility of this approach is demonstrated in a series of experiments using synthetic and realistic datasets that are designed to simulate heterogeneous data from different sources.

Keywords: Multi-Objective Optimization, Cluster Matching, Attribute Matching, Simulated Annealing.

1 Introduction

The presence of heterogeneity among schemas and ontologies supporting vast amount of informational sources leads to one of the most important and toughest problems, that is, the semantic integration of heterogeneous data sources to facilitate interoperability and reuse of the information. The difficulty is especially pronounced in many scientific domains where massive amount of data are produced independently and thus each having their own data vocabulary. While manual integration is time-consuming and requires expensive specialized human capital, the development of automatic approaches becomes imminent to aid inter-institute collaboration. One purpose of the present paper is to suggest a method for solving a specific kind of ontology/schema matching problem under some severe constraints that can cause traditional methods to be ineffective. The constraints that we deal with are, namely, 1) little-to-no string-based or linguistic similarity between terminologies, and 2) all numeric typed data instances. This phenomenon is commonly seen in integrating scientific datasets which involves discovery of correspondences among distinct numeric-typed summary features ("attributes") that are used to characterize datasets that have been collected and analyzed in different research labs. We call this the *attribute matching* problem.

Another challenging task given multiple data sources is to carry out meaningful meta-analysis that combines results of several studies on different datasets to address a set of related research hypotheses. Finding correspondences among distinct patterns that are observed in different scientific datasets is an example of meta-analysis. Supposing the patterns are derived by clustering analysis, this problem can be addressed by the application of cluster comparison (or cluster matching) techniques. Clustering is an unsupervised data mining task widely used to discover patterns and relationships in a variety of fields. The clustering result provides a pattern characterization from a data-driven perspective. If similar results are obtained across multiple datasets, this leads in turn to a revision and refinement of existing domain knowledge, which is a central goal of meta-analysis. However, there are noticeably few cluster comparison methods that are able to compare two clusterings derived from different datasets. The difficulty for the comparison is further exacerbated by the fact that the datasets may be described by attributes from heterogeneous ontologies or schemas. Even those methods that are able to measure clustering similarity across different datasets (e.g., the ADCO [1] method) have to assume the homogeneous meta-data.

Given this situation, in order to carry out cluster comparison for meta-analysis, researchers often need to perform ontology or schema matching first in order to mitigate the meta-data gap. In previous work [11], we examine a practical attribute matching problem on neuroscience data where schema elements from one dataset share no lexical similarity with those from the other. Moreover, structural similarity is also limited. One can only resort to instance-based (extensional) methods. However, since all attributes are numerical, information clues available to an instance-level matcher is very restricted. Traditional instance-based matchers typically make use of constraint-based characterization, such as numerical value ranges and averages to determine correspondences. However, this is often too rough in the case of all-numerical dataset. Two attributes may have similar ranges and averages but totally different internal value distributions (an example is shown in Section 4.1). Given this, we propose to represent the attribute value distribution at a finer granularity by partitioning the values into groups. To do this, clustering is performed, and resulting clusters are then aligned across two datasets (assuming that the same pattern exists in both datasets). In this way, each attribute can be characterized by, instead of a single value, a vector of per-cluster statistical quantities (i.e., the *segmented statistical characterization*). A distance function can then be applied based on this representation. Table 1(A) shows an example distance table on the cross join of two sets of attributes. To discover attribute matching from this table can be reduced to solving a minimum assignment problem (assuming matching is bijective), which is a classical combinatory optimization problem that has a polynomial solution using the Hungarian Method [8].

Unfortunately, however, the above solution requires us to be able to align clusters across datasets, which is a difficult problem in its own right. If fully automated, as mentioned above, methods such as ADCO adopt a so called *density profile* [1] representation of clusters that requires homogeneous meta-data or a priori knowledge about the attribute matching in heterogeneous scenarios. Then the cluster matching can be carried out in a similar manner to the attribute matching by casting to the assignment problem (see Table 1(B), for example). This leads to a circular causality, or a deadlock, between the attribute matching (under the segmented statistical characterization) and cluster matching (under the density profile representation) problems—none of them can be solved automatically without the other one being solved first.

Table 1. Example distance matrices between (A) two sets of attributes and (B) two sets of clusters, respectively

	a'_1	a'_2	$\cdots$	a'_m
a_1	$d_{11'}$	$d_{12'}$	$\cdots$	$d_{1m'}$
a_2	$d_{21'}$	$d_{22'}$		$d_{2m'}$
$\vdots$			$\ddots$	
a_n	$d_{n1'}$	$d_{n2'}$		$d_{nm'}$

(A)

	c'_1	c'_2	$\cdots$	c'_n
c_1	$d_{11'}$	$d_{12'}$	$\cdots$	$d_{1n'}$
c_2	$d_{21'}$	$d_{22'}$		$d_{2n'}$
$\vdots$			$\ddots$	
c_n	$d_{n1'}$	$d_{n2'}$		$d_{nn'}$

(B)

To solve this difficulty, in the present paper, viewing the two matching problems as combinatorial optimization problems with distinct yet interrelated objective functions, we propose a novel approach using a multi-objective simulated annealing (MOSA) to discover attribute matching and cluster matching simultaneously. The objectives in the optimization are to minimize distances of attribute matching and cluster matching respectively.

The rest of this paper is organized as follows. We review the basics of multi-objective optimization and describes the relationship between various components of the proposed method and existing methods in Section 2. We present detailed description of our method for simultaneously discovering attribute matching and cluster matching in Section 3. We report experimental results in Section 4 and conclude the paper in Section 5.

2 Background and Related Work

2.1 The Multiobjective Optimization Problem and Pareto-Optimality

Multi-objective optimization problem (also called multi-criteria, multi-performance or vector optimization) can be defined mathematically as to find the vector $X = [x_1, x_2, \ldots, x_k]^T$ which satisfies the following m inequality constraints and l equality constraints:

$$g_i(X) \geq 0, i = 1, 2, \ldots, m$$
$$h_i(X) = 0, i = 1, 2, \ldots, l$$

and optimize the objective function vector

$$F(X) = [f_1(X), f_2(X), \ldots, f_N(X)]^T$$

where $X = [x_1, x_2, \ldots, x_k]^T$ is called the decision variable vector.

Real-life problems require simultaneous optimization of several incommensurable and often conflicting objectives. Usually, there is no single optimal solution, but there is a set of alternative solutions. These solutions are optimal in the sense that no other solutions in the search space are superior to each other when all the objectives are considered [16]. They are known as Pareto-optimal solutions. To define the concept of Pareto optimality, we take the example of a minimization problem with two decision vectors $a, b \in X$. Vector a is said to dominate b if

$$\begin{aligned} &\forall i = \{1, 2, \ldots, N\} \ : \ f_i(a) \leq f_i(b) \\ &\mathit{and} \\ &\exists j = \{1, 2, \ldots, N\} \ : \ f_j(a) < f_j(b) \end{aligned}$$

When the objectives associated with any pair of non-dominated solutions are compared, it is found that each solution is superior with respect to at least one objective. The set of non-dominated solutions to a multi-objective optimization problem is known as the Pareto-optimal set (Pareto front) [17].

2.2 Simulated Annealing in Multi-Objective Optimization

Simulated annealing (SA) is based on an analogy of thermodynamics with the way metals cool and anneal. It has been proved to be a compact and robust technique, which provides excellent solutions to single and multiple objective optimization problems with a substantial reduction in computation time. It is a method to obtain an optimal solution of a single objective optimization problem and to obtain a Pareto set of solutions for a multi-objective optimization problem. Simulated Annealing was started as a method or tool for solving single objective combinatorial problems, these days it has been applied to solve single as well as multiple objective optimization problems in various fields. A comprehensive survey can be found in [16].

2.3 The Schema Matching Problem

Our study of matching alternative attribute sets is closely related to the schema matching problem. According to the type of instance value, various instance-based approaches have been developed in previous research. For example, for textual attributes, a linguistic characterization based on information retrieval techniques can be applied [12]; for nominal attributes, evaluation of the degree of overlap of instance values is a preferred approach. Larson et al. [9] and Sheth et al. [14] discussed how relationships and entity sets could be integrated primarily based on their domain relationships. Similarity of partially overlapped instance set can be also calculated based on measures such as Hamming distance and Jaccard coefficient; for numeric attributes, most methods use aggregated statistics to characterize the attributes, e.g., 'SSN' and 'PhonNo' can be distinguished based on their respective patterns [12]. Hybrid systems that combine several approaches to determine matching often achieve better performance. For example, SemInt [10] is a comprehensive matching prototype exploiting up to 15 constraint-based and 5 content-based matching criteria. The LSD (Learning Source Descriptions) [4] system uses several instance-level matchers (learners) that are trained during a preprocessing step. The iMAP [2] system uses multiple basic matchers, called searches, e.g., text, numeric, category, unit conversion, each of which addresses a particular subset of the match space.

Due to the nature of many scientific datasets, we face several unique challenges. First, the data under study are semi-structured, thus invalidating those matching methods that presume a complete, known-in-advance schematic structure. In addition, totally different labels (usually acronyms or pseudowords) are widely adopted for the same or similar metrics, rendering lexical similarity-based methods unsuitable. Moreover, an important limitation of previous instance-based matching methods is their inability to handle numerical instances appropriately in certain domain applications. They use statistical characterization extracted from the numerical instances, such as range, mean and standard deviation, to determine match. However such information is too rough to capture patterns in data that are crucial in determining the correspondence.

2.4 The Cluster Matching Problem

The cluster matching (cluster comparison) problem is related to the cluster validity problem, especially the technique of external/relative indexing that aims at comparing two different clustering results. Popular methods in this field, including the Rand index [13], Jaccard index [7], normalized mutual information [5], etc., are mostly based on examining membership of points to clusters. However, the basis of these methods is the comparison of different clustering schema on the same dataset.

By contrast, in the present case we are aiming to match clusters across datasets that contain non-overlapping observations. Thus, membership-based cluster validity criteria are unsuitable. A recent clustering similarity index known as ADCO (Attribute Distribution Clustering Orthogonality) proposed by Bae et al. [1] can match clusterings from non-overlapping datasets. The ADCO measure determines the similarity between two clusterings based on their *density profiles*, which incorporate distribution information of data points along each attribute. The density profile representation of clusters are defined as follows.

Density Profile: To represent clusters using density profiles, the attribute's range in each cluster is first discretized into a number of bins, and the similarity between two clusters corresponds to the number of points of each cluster falling within these bins. The formal definition for this number of points is the *density* of an attribute-bin region for cluster c_k in clustering C, denoted as $dens_C(k, i, j)$. It refers to the number of points in the region (i, j)—the j-th bin of the i-th attribute—that belongs to the cluster c_k of clustering C. For example, for clustering C in Fig. 1, $dens_C(1,1,1) = 8$, because there are 8 data points in region $(1,1)$—the first bin of the first attribute x—that belongs to the first cluster c_1.

The density profile vector V_C for a clustering C is formally defined as an ordered tuple:

$$V_C = \Big[dens_C(1,1,1),\ dens_C(1,1,2), \ldots,\ dens_C(1,1,Q),\ dens_C(1,2,1), \\ \ldots,\ dens_C(1,M,Q),\ dens_C(2,1,1), \ldots,\ dens_C(N,M,Q)\Big], \quad (1)$$

where Q is the number of bins in each of the M attributes, and K is the number of clusters in C.

The ADCO measure: After the density profile vectors of two clusterings C and C' are obtained, the degree of similarity between C and C' can be determined by calculating the dot product of the density profile vectors:

$$sim(C, C') = V_C \cdot V_{C'} .$$

Given a permutation π under which the similarity function $sim(C,\ \pi(C'))$ is maximized, an ADCO measure is calculated using a normalization factor (NF) corresponding to the maximum achievable similarity of the clusterings: $NF(C, C') = max\big[sim(C,C),\ sim(C',C')\big]$. The $ADCO(C, C')$ measure is defined as follows:

$$ADCO(C, C') = \frac{sim(C, C')}{NF(C, C')} .$$

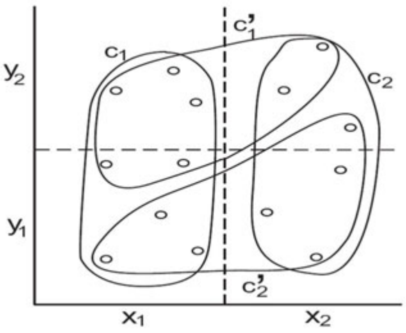


Fig. 1. Two clusterings $C = \{c_1, c_2\}$ and $C' = \{c'_1, c'_2\}$. Two attributes X (attribute 1) and Y (attribute 2) are discretized into 2 bins each. See [1] for details.

3 Method

3.1 The Multi-Objective Simulated Annealing Framework

To solve the dual matching problems, we adopt a strategy of multi-objective simulated annealing described in [15], in which the acceptance criterion in the simulated annealing process is established based on the idea of Pareto-domination based fitness. Fitness of a solution is defined as one plus the number of dominating solutions in Pareto-optimal set. The larger the value of fitness, the worse is the solution. Initially, fitness difference between the current and the generated solution is less and the temperature is high so any move is accepted due to both of them. This gives a way to explore the full solution space. As the number of iterations increases, temperature decreases and fitness difference between the current and generated solutions may increase. Both of them make the acceptance move more selective and it results in a well-diversified solution in true Pareto-optimal solutions. Details of our adaptation of the above multi-objective simulated annealing framework is outlined in Algorithm 1.

Algorithm 1. Multi-Objective Simulated Annealing

```
Input: Empty Pareto-optimal set of solutions $\mathbf{\Sigma}$
Input: Empty current decision vector $\mathbf{X} = [x_a, x_c]$
Input: Initial temperature $T$
  $count = 0$
  while $T > threshold$ do
    $initialize(\mathbf{X})$
    Put $\mathbf{X}$ in $\mathbf{\Sigma}$
    $\mathbf{X}' = generate\_solution(\mathbf{X})$
    $S_{\mathbf{X}'} = evaluate\_solution(\mathbf{X}')$
    $\Delta S = S_{\mathbf{X}'} - S_X$
    if $r = rand(0,1) < exp(\frac{-\Delta S}{T})$ then
      $\mathbf{X} = \mathbf{X}'$
      $S_X = S_{X'}$
    end if
    //Periodically restart
    if $count == restart\_limit$ then
      $\mathbf{X} = select\_random\_from\_Pareto(\mathbf{\Sigma})$
      continue
    end if
    $reduce\_temperature(T)$
  end while
```

Mathematically, the processes involved in the proposed multi-objective simulated annealing framework can be defined as follows.

$$
\begin{aligned}
X &= [x_a, x_c] \\
F &= [f_a, f_c] \\
P_a([x_a^{(n-1)}, x_c^{(n-1)}]) &= [x_a^{(n)}, x_c^{(n-1)}] \\
P_c([x_a^{(n-1)}, x_c^{(n-1)}]) &= [x_a^{(n-1)}, x_c^{(n)}] \\
G_{c|a}([x_a^{(n)}, x_c^{(n-1)}]) &= [x_a^{(n)}, x_c^{(n)}] \\
G_{a|c}([x_a^{(n-1)}, x_c^{(n)}]) &= [x_a^{(n)}, x_c^{(n)}] \\
G \circ P([x_a^{(n-1)}, x_c^{(n-1)}]) &= [x_a^{(n)}, x_c^{(n)}]
\end{aligned}
$$

X is the decision vector that contains two variables for attribute matching, x_a, and cluster matching, x_c, respectively (details in Section 3.2). F is the objective function vector that contains two criterion functions (f_a and f_c) to evaluate attribute matching and cluster matching decisions (details in Section 3.4). P is the random perturbation function that takes a decision vector in the $(n-1)$th iteration and partially advances it to the nth iteration (we use P_a or P_c to distinguish between the random selections). The partial candidate decision generation function G takes the output of P and fully generate a decision vector for the nth iteration (by advancing the left-out variable in P to its nth iteration). Thus, the compound function $G \circ P$ fulfils the task of generating an nth-iteration candidate decision vector given the $(n-1)$th one (details in Section 3.5).

3.2 Decision Variable

The domains of the decision variables in the matching problems take values on a permutation space. In other word, by formalizing the problem of finding correspondent elements of two sets S and S' of cardinality n as an optimization problem, the solution is completely specified by determining an optimal permutation of $1, \ldots, n$. For instance, for two sets of three elements, their indexes range over $\{0, 1, 2\}$. Applying a permutation $\pi = \{2, 0, 1\} \in S_3$ on S' can be viewed as creating a mapping (bijection) from elements on the new positions of S' to elements on the corresponding positions in S. In this example, the permutation π on S' specifies the following correspondences: $S_0 \leftrightarrow S'_2$, $S_1 \leftrightarrow S'_0$, and $S_2 \leftrightarrow S'_1$.

Formally, let P_n ($n \in \mathbb{N}$) be the symmetric group of all permutations of the set $\{1, 2, \ldots, n\}$. Given two sets S and S' with the same cardinality of n, performing identity permutation on one set and an arbitrary permutation $\pi \in S_n$ on the other specifies a matching (or mathematically speaking, mapping) between the two sets. In the multi-objective optimization formalism for solving attribute matching and cluster matching problems, the decision vector has two variables: $X = [x_a, x_c]$. If we have M attributes and N clusters to match respectively, then $x_a \in P_M$ and $x_c \in P_N$.

3.3 Data Representation

The central objects of interest in our study, namely, the numeric-typed attributes and clusters, need to be represented in ways that meaningful quantities can be defined to measure the "goodness" of a matching decision. To this end, we propose to use the *segmented statistical characterization* to represent attributes, and the *density profiles* to represent clusters. Details of these representations are described below.

Representation of Attributes: Numeric-typed attributes can be represented by the segmented statistical characterization, in which data instances are first partitioned into groups (e.g., through unsupervised clustering) and then characterized by a vector of indicators, each denoting a statistical characterization of the corresponding group. For example, if values of an attribute A are clustered into n groups, then it can be represented by a vector of segmented statistical characterization as follows:

$$V_A = \left[\mu_1, \mu_2, \ldots, \mu_n\right],$$

where we choose the mean value μ_i for cluster i as the statistical indicator in our implementation.

Representation of Clusters: Clusters can be represented using density profiles [1] as described in Section 2. The attribute's range in each cluster is first discretized into a number of bins, and the similarity between two clusters corresponds to the number of points (i.e. *density*) of each cluster falling within these bins. Given this, density profile vector V_C for a clustering C is formally defined as an ordered tuple by Equation 1 and is repeated here:

$$V_C = \Big[dens_C(1,1,1),\ dens_C(1,1,2), \ldots,\ dens_C(1,1,Q),\ dens_C(1,2,1),$$
$$\ldots,\ dens_C(1,M,Q),\ dens_C(2,1,1), \ldots,\ dens_C(N,M,Q)\Big],$$

where Q is the number of bins in each of the M attributes, N is the number of clusters in C, and $dens_C(k,i,j)$ refers to the number of points in the region (i,j)—the j-th bin of the i-th attribute—that belongs to the cluster c_k of clustering C.

3.4 Objective Functions

The objective functions in the attribute matching and cluster matching problems are criteria to evaluate the "goodness" of matchings. We use the sum of pair-wise distances between matched elements (see Figure 1 for example) as the objective function. Given this, to determine the form of objective functions amounts to defining proper pair-wise distance measures for the attribute and cluster matching problems respectively, as detailed in the following.

Distance function between two attributes. The pairwise distance between two attributes are defined as the Euclidean distance between their segmented statistical characterization vectors, and f_a calculates the sum of pair-wise distances under the attribute matching specified by x_a:

$$f_a(x_a) = \sum_{k=1}^{M} \mathcal{L}\Big((V_a)^k,\ (V_a')^{x_a(k)} \Big) = \sum_{k=1}^{M} \sqrt{\sum_{i=1}^{N} \Big(\mu_i^k - (\mu')_i^{x_a(k)} \Big)^2}\ , \tag{2}$$

where $x_a \in P_M$.

Distance function between two clusters. The ADCO similarity described in Section 2.4 can be transformed to a distance defined as follows [1]:

$$D_{ADCO}(C, C') = \begin{cases} 2 - ADCO(C, C')\ , & if\ C \neq C'\ (V_C \neq V_{C'}) \\ 0 & , otherwise \end{cases} \tag{3}$$

We use D_{ADCO} as the pair-wise distance between two clusters under the density profile representation, and f_c calculates the sum of pair-wise distances under the cluster matching specified by x_c

$$f_c(x_c) = \sum_{k=1}^{N} D_{ADCO}\Big((V_c)^k,\ (V_c')^{x_c(k)} \Big) = \sum_{k=1}^{N} \left(2 - \frac{\sum_{i=1}^{M}\sum_{j=1}^{Q} \Big(dens(k,i,j) \times dens(x_c(k),i,j) \Big)}{\max\left[\sum_{i=1}^{M}\sum_{j=1}^{Q} dens(k,i,j)^2\ ,\ \sum_{i=1}^{M}\sum_{j=1}^{Q} dens(x_c(k),i,j)^2 \right]} \right), \tag{4}$$

where $x_c \in P_N$.

3.5 Generation of New Solution

In each iteration of the simulated annealing process, we randomly generate candidate decision in the neighborhood of the last-iteration decision by applying two consecutive processes, namely, the random perturbation and the partial candidate decision generation, as described below.

Random Perturbation: In each iteration, we select at random one variable (either x_a or x_c) in the decision vector and perturb it by randomly swapping two positions in the selected variable. This advances that variable from $(n-1)$th iteration to nth iteration. Then the following partial candidate generation process is carried out to bring the other variable also to nth iteration.

Partial candidate decision generation
Given $x_c^{(n)}$, derive $x_a^{(n)}$:

$$x_a^n = \arg\min_{\pi} f_a(\pi, x_c^{(n)}) = \arg\min_{\pi} \sum_{k=1}^{M} \mathcal{L}\Big((V_a)^k,\ (V_a')^{\pi(k)} \Big)$$
$$= \arg\min_{\pi} \sum_{k=1}^{M} \sqrt{\sum_{i=1}^{N} \Big(\mu_i^k - (\mu')^{\pi(k)}_{x_c^{(n)}(i)} \Big)^2} \tag{5}$$

Given $x_a^{(n)}$, derive $x_c^{(n)}$:

$$x_c^n = \arg\min_{\pi} f_c(\pi, x_a^{(n)}) = \arg\min_{\pi} \sum_{k=1}^{N} D_{ADCO}\Big((V_c)^k,\ (V_c')^{\pi(k)} \Big)$$

$$= \arg\min_{\pi} \sum_{k=1}^{N} \left(2 - \frac{\sum_{i=1}^{M}\sum_{j=1}^{Q} \Big(dens(k,i,j) \times dens(\pi(k), x_a^{(n)}(i), j) \Big)}{\max\left[\sum_{i=1}^{M}\sum_{j=1}^{Q} dens(k,i,j)^2\ ,\ \sum_{i=1}^{M}\sum_{j=1}^{Q} dens(\pi(k), x_a^{(n)}(i), j)^2 \right]} \right) \tag{6}$$

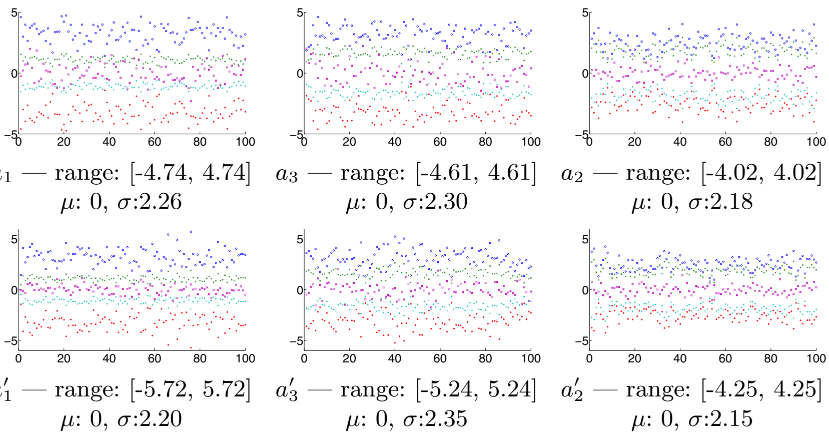
To calculate π that satisfies equations 5 and 6, rather than iterating through all possible permutations, we can consider the equation as a minimum-cost assignment problem. Table 1(A), for example, illustrates a distance table between two attribute sets A and A'. Matching of the two sets can be considered as an assignment problem where the goal is to find an assignment of elements in $\{A_i\}$ to those in $\{A_i'\}$ that yields the minimum total distance without assigning each A_i more than once. This problem can be efficiently solved by the Hungarian Method in polynomial time of $O(K_{min}^3)$ [8]. It is worth noting that by formulating the problem as the assignment problem, we assume the matching between two sets to be a one-to-one function.

4 Experiment

Because we are interested in understanding the property of the Pareto front obtained by our method, we conducted a series of experiments to highlight tradeoffs of the objectives functions. First, to illustrate the proposed method is indeed capable of determining matching between numeric-typed attributes and clusters, we synthesized a dataset simulating some extreme conditions under which previous methods are ineffective. Also, from the results obtained on the synthetic dataset, we empirically study tradeoffs between the two objective functions. Then, to evaluate the scalability of the method, we carry out a series of tests on a set of data with varied sizes. Finally, encouraged by these results, we applied our methods to actual neuroscience ERP (event-related potentials) data to highlight the applicability of our method to the neuroscience domain.

4.1 Synthetic Dataset

Data Generation: In the synthetic dataset, we generated values for each attribute in such a way that each attribute can be divided into several clusters, and each cluster corresponds to a Gaussian distribution with different mean and standard deviation, but the overall mean and standard deviation of values from all clusters in one attribute are made very close to those in other attributes. For example, Figure 2 illustrates the value distributions of three attributes (a_1, a_2, and a_3) from one dataset and their corresponding counterparts (a'_1, a'_2, and a'_3) from another. It shows that the overall means and standard deviations for these six attributes are almost indistinguishable, and their ranges are similar as well. Previous methods using these whole-attribute-wise quantities as statistical characterization of attributes would have a hard time determining the matchings. However, as mentioned above and illustrated in the figure, the individual distributions underlying clusters in these attributes are distinct and, by using the segmented statistical characterization of attributes, the difference is significant enough to differentiate and identify matchings between attributes.



a_1 — range: [-4.74, 4.74] μ: 0, σ:2.26

a_3 — range: [-4.61, 4.61] μ: 0, σ:2.30

a_2 — range: [-4.02, 4.02] μ: 0, σ:2.18

a'_1 — range: [-5.72, 5.72] μ: 0, σ:2.20

a'_3 — range: [-5.24, 5.24] μ: 0, σ:2.35

a'_2 — range: [-4.25, 4.25] μ: 0, σ:2.15

Fig. 2. Scatter plots of data instances from three sample attributes in one synthetic dataset (upper frame) and those of their corresponding attributes from another (lower frame) are illustrated to show their respective value distributions

Results: Figure 3 illustrates the Pareto front obtained from matching two synthetic datasets, each having 20 attributes and 5 clusters. Most notably, the gold standard results for both attribute matching and cluster matching are obtained from the left-most point on the Pareto front. In other words, given the decision variables (X) corresponding to that point, we obtained 100% correct matching results. We further observed that in our subsequent tests on other synthetic datasets with varied number of attributes and clusters, the derived Pareto fronts all contain gold standard result, and the point corresponding to the gold standard can always be found towards the minimum end of f_a. Given this, we propose

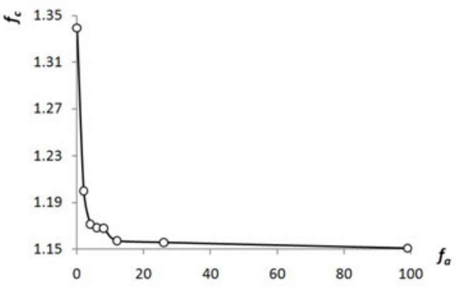


Fig. 3. An example Pareto front obtained from matching two synthetic datasets with 20 attributes and 5 clusters

the following method to reduce the Pareto-optimal set to a single point corresponding to the most favored choice (X^*) in the decision space. The idea is to find the decision with the minimum weighted sum of objective values in the obtained Pareto-optimal set, i.e., $X^* = \arg\min_X \left[\alpha f_a(X) + \beta f_c(X)\right]$, where α and β are weights. We first conducted preliminary experiments to determine the best values for α and β (0.8 and 0.2 respectively) and used them in all subsequent experiments. This method works markedly well on the synthetic datasets. For all the tests described in Table 2, 100% correct results for both attribute and cluster matchings are obtained (hence we omit the precision in the table).

Running Time: We systematically altered the number of attributes and clusters present in the data and conducted a series of tests to show the scalability of the proposed method. The running time under different configurations is reported in Table 2. The time is calculated by averaging over 5 runs of each test (on a 2.53GHz dual-core CPU with 4 gigabytes memory), each run having 1000 iterations in the simulated annealing process. The main computationally expensive part of the annealing process is the generation of new candidate solution phase (function G) in which an assignment problem is solved using the Hungarian method. The complexity of the Hungarian method is cubic and is already the most efficient algorithm for solving the assignment problem (a brute force algorithm has a factorial complexity). Fortunately, rarely is the case that the number of attributes or clusters is large in real-world scenarios where the proposed technique is needed. For reasonable configurations in most practical applications, the computation time is within a tractable range as shown in table 2.

4.2 Neuroscience Dataset

Data Acquisition: To address the problems of attribute and cluster matching in a real-world neuroscience application, we used a set of realistic simulated ERP (event-related potentials) datasets, which were designed to support evaluation of ERP analysis methods [6]. The datasets were specifically designed to simulate heterogeneous data from different groups of subjects under different conditions (via distinct simulated brain activities), as well as distinct measurement methods

Table 2. Running time of the annealing process on synthetic datasets with varied configurations of attribute and cluster sizes. The time is obtained by averaging over results of 5 runs of each test.

# attributes	# clusters	time (sec)
5	20	0.28
20	20	1.81
20	40	7.04
20	60	17.80
40	20	4.66
40	40	11.74
40	60	25.93
60	20	10.95
60	40	20.70
60	60	37.35
100	100	172.23

(spatial and temporal metrics) and distinct patterns (reflecting two different pattern decomposition techniques). Real ERP data arise from superposition of latent scalp-surface electrophysiological patterns, each reflecting the activity of a distinct cortical network that cannot be reconstructed from the scalp-measured data with any certainty. Thus, real ERP data are not appropriate for evaluation of ERP pattern mapping. By contrast, simulated ERP data are derived from known source patterns and therefore provide the necessary gold standard for evaluation of our proposed methods.

The raw data for this study consist of 80 simulated event-related potentials (ERPs), in which each ERP comprises simulated measurement data for a particular subject ($n = 40$). The 40 simulated subjects are randomly divided into two 20-subject groups, SG1 and SG2, each containing 40 ERPs (20 subjects in 2 experimental conditions). Each ERP consists of a superposition of 5 latent varying spatiotemporal patterns. These patterns were extracted from the two datasets, SG1 and SG2, using two techniques: temporal Principal Components Analysis (tPCA) and spatial Independent Components Analysis (sICA), two data decomposition techniques widely used in ERP research [3]. To quantify the spatiotemporal characteristics of the extracted patterns, two alternative metric sets, m1 and m2, were applied to the two tPCA and the two sICA derived datasets. For a complete explanation of these alternative metrics, please see Appendix in [6].

In summary, the simulated ERP data generation process yielded eight test datasets in total, reflecting a 2 (attribute sets) $\times$ 2 (subject groups) $\times$ 2 (decomposition methods) factorial design. Therefore, for each attribute sets there are 4 datasets generated from different combinations of subject groups and decomposition methods, resulting $4 \times 4 = 16$ cases for the studies of attribute matching and cluster matching. The reason to include such variabilities was to test the robustness of our matching method to different sources of heterogeneity across the different datasets. Within all test datasets, 5 major ERP spatiotemporal patterns are present. They are P100, N100, N3, MFN, and P300.

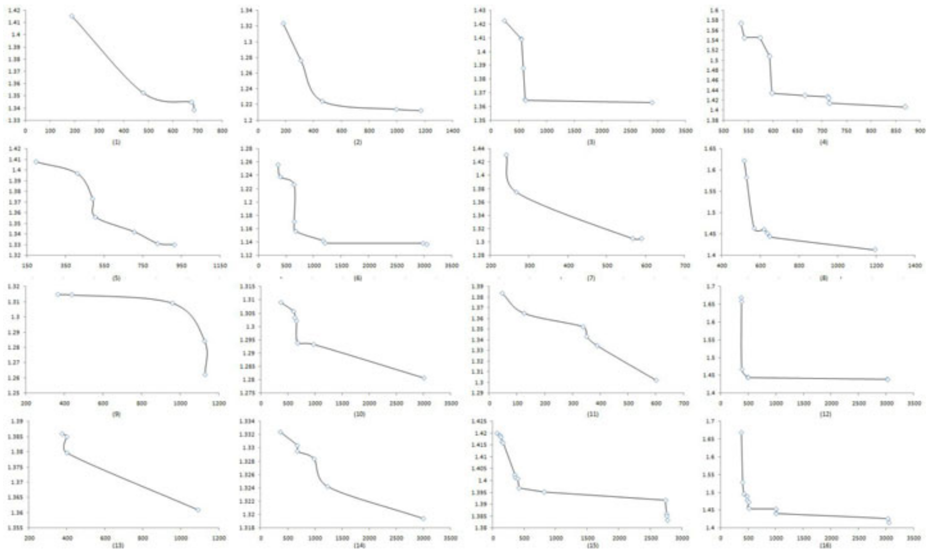


Fig. 4. Pareto fronts obtained from the 16 test cases of the neuroscience dataset

These patterns can be identified in the datasets by clustering analysis. Pretending that the latent patterns underlying discovered clusters are unknown, we hope to match clusters across datasets to recover the fact that the same patterns are present in all datasets.

Results: Figure 4 illustrates the Pareto fronts derived by the proposed method on each of the 16 test cases. We applied the weighted sum method to determine the most favored choice from the Pareto fronts using the parameters (α and β) discovered in the preliminary experiments on synthetic datasets (cf. Section 4.1). The accuracy of attribute matching and cluster matching along with the number of points in the Pareto front are listed in Table 3 (all these results are obtained by taking average from 5 runs for each test case).

It can be observed from the results in Table 3 that more different factors involved in the acquisition of the two datasets for matching can negatively affect the matching performance. For example, in test case 1, the two datasets are drawn from the same subject group (SG1) and preprocessed using the same decomposition method (sICA); whereas in test case 4, the subject groups and decomposition methods are all different, resulting in greater variability and hence the performance is less satisfactory. However, it is worth noting that our method greatly outperforms traditional whole-attribute-based statistic characterization, as is shown in Table 5. In this table we also demonstrate the accuracy of the segmented statistics characterization with expert-labeled patterns, meaning that the data is partitioned and aligned in the most accurate way, which marks the best achievable attribute matching performance. But it is not feasible because, as mentioned in Section 1, manually recognizing patterns (partitioning data) and aligning them across datasets requires

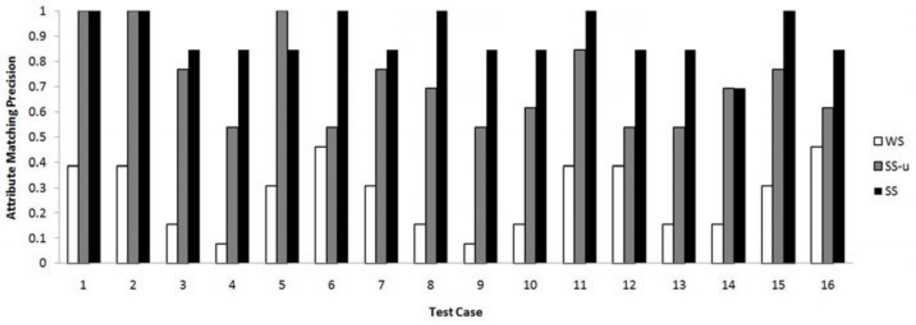


Fig. 5. A comparison of the attribute matching accuracy of three methods on the 16 test cases of the neuroscience dataset. The three methods being compared are matching based on whole-attribute statistics (WS), segmented attribute statistics without knowing a priori cluster matching (SS-u), and segmented attribute statistics with expert-aligned clusterings (SS).

a priori knowledge of attributes in the datasets which is exactly what the problem of attribute matching tries to discover (the circular causality problem). On the other hand, our method does not require human involvement (except the specification of the number of clusters (patterns) present in the data in order to run the clustering analysis) in determining both the attribute matching and cluster matching and is able to achieve close-to-optimal results.

Table 3. Matching performance of the proposed method on the 16 test cases from the neuroscience dataset. The source and target parameter configuration of the data acquisition process of each test case are shown. P_a and P_c denote the accuracy of attribute matching and cluster matching respectively. Σ is the number of points in the obtained Pareto-front. The quantities listed in the table are obtained by averaging over 5 runs of each test.

Test case	Source params	Target params	P_a	P_c	$\|\Sigma\|$
1	⟨ SG1, sICA, m1 ⟩	⟨ SG1, sICA, m2 ⟩	13/13	5/5	5
2	⟨ SG1, sICA, m1 ⟩	⟨ SG2, sICA, m2 ⟩	13/13	5/5	6
3	⟨ SG1, sICA, m1 ⟩	⟨ SG1, tPCA, m2 ⟩	10/13	5/5	6
4	⟨ SG1, sICA, m1 ⟩	⟨ SG2, tPCA, m2 ⟩	7/13	3/5	8
5	⟨ SG2, sICA, m1 ⟩	⟨ SG1, sICA, m2 ⟩	11/13	3/5	7
6	⟨ SG2, sICA, m1 ⟩	⟨ SG2, sICA, m2 ⟩	13/13	5/5	7
7	⟨ SG2, sICA, m1 ⟩	⟨ SG1, tPCA, m2 ⟩	10/13	5/5	6
8	⟨ SG2, sICA, m1 ⟩	⟨ SG2, tPCA, m2 ⟩	9/13	2/5	8
9	⟨ SG1, tPCA, m1 ⟩	⟨ SG1, sICA, m2 ⟩	7/13	5/5	4
10	⟨ SG1, tPCA, m1 ⟩	⟨ SG2, sICA, m2 ⟩	8/13	5/5	6
11	⟨ SG1, tPCA, m1 ⟩	⟨ SG1, tPCA, m2 ⟩	11/13	5/5	6
12	⟨ SG1, tPCA, m1 ⟩	⟨ SG2, tPCA, m2 ⟩	7/13	3/5	5
13	⟨ SG2, tPCA, m1 ⟩	⟨ SG1, sICA, m2 ⟩	7/13	3/5	5
14	⟨ SG2, tPCA, m1 ⟩	⟨ SG2, sICA, m2 ⟩	9/13	5/5	6
15	⟨ SG2, tPCA, m1 ⟩	⟨ SG1, tPCA, m2 ⟩	10/13	3/5	8
16	⟨ SG2, tPCA, m1 ⟩	⟨ SG2, tPCA, m2 ⟩	8/13	3/5	8

5 Conclusion

In this paper, we have presented a data mining approach to challenges in the matching and integration of heterogeneous datasets. In particular, we have proposed solutions to two problems that arise in combining information from different results of scientific research. The first problem, *attribute matching*, involves discovery of correspondences among distinct numeric-typed summary features ("attributes") that are used to characterize datasets that have been collected and analyzed in different research labs. The second problem, *cluster matching*, involves discovery of matchings between patterns across datasets.

We have treated both of these problems together as an multi-objective optimization problem. We developed a segmented statistics characterization to represent numeric-typed attributes and adapted the density profile to represent clusters. Based on these representations, we proposed objective functions that best define the criteria for selecting matching decisions. A multi-objective simulated annealing algorithm was described to find the optimal decision. The utility of this approach was demonstrated in a series of experiments using synthetic and realistic datasets that were designed to simulate heterogeneous data from different sources.

Acknowledgement. This work is supported by the NIH/NIBIB with Grant No. R01EB007684.

References

1. Bae, E., Bailey, J., Dong, G.: A Clustering Comparison Measure Using Density Profiles and Its Application to The Discovery tf Alternate Clusterings. Data Min. Knowl. Discov. 21, 427–471 (2010), http://dx.doi.org/10.1007/s10618-009-0164-z
2. Dhamankar, R., Lee, Y., Doan, A., Halevy, A., Domingos, P.: iMAP: Discovering Complex Semantic Matches between Database Schemas. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. ACM Press (2004)
3. Dien, J.: The ERP PCA Toolkit: An Open Source Program for Advanced Statistical Analysis of Event-Related Potential Data. Journal of Neuroscience Methods 187(1), 138–145 (2010), http://www.sciencedirect.com/science/article/B6T04-4Y0KWB2-4/2/3c0e7b36b475b8d0e9a72c7b868a7dcd
4. Doan, A., Domingos, P., Levy, A.Y.: Learning Source Description for Data Integration. In: WebDB (Informal Proceedings), pp. 81–86 (2000), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.9378
5. Fred, A.L., Jain, A.K.: Robust Data Clustering. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, p. 128 (2003)
6. Frishkoff, G.A., Frank, R.M., Rong, J., Dou, D., Dien, J., Halderman, L.K.: A Framework to Support Automated Classification and Labeling of Brain Electromagnetic Patterns. Computational Intelligence and Neuroscience (CIN), Special Issue, EEG/MEG Analysis and Signal Processing 7(3), 1–13 (2007)

7. Hamers, L., Hemeryck, Y., Herweyers, G., Janssen, M., Keters, H., Rousseau, R., Vanhoutte, A.: Similarity Measures In Scientometric Research: The Jaccard Index Versus Salton's Cosine Formula. Inf. Process. Manage. 25, 315–318 (1989), http://portal.acm.org/citation.cfm?id=67223.67231
8. Kuhn, H.W.: The Hungarian Method for The Assignment Problem. Naval Research Logistic Quarterly 2, 83–97 (1955)
9. Larson, J.A., Navathe, S.B., Elmasri, R.: A Theory of Attributed Equivalence in Databases with Application to Schema Integration. IEEE Trans. Softw. Eng. 15, 449–463 (1989), http://portal.acm.org/citation.cfm?id=63379.63387
10. Li, W.S., Clifton, C.: Semint: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks (2000)
11. Liu, H., Frishkoff, G., Frank, R., Dou, D.: Ontology-Based Mining of Brainwaves: A Sequence Similarity Technique for Mapping Alternative Features in Event-Related Potentials (ERP) Data. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) PAKDD 2010. LNCS, vol. 6119, pp. 43–54. Springer, Heidelberg (2010)
12. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10 (2001)
13. Rand, W.M.: Objective Criteria for the Evaluation of Clustering Methods. Journal of the American Statistical Association 66(336), 846–850 (1971), http://dx.doi.org/10.2307/2284239
14. Sheth, A.P., Larson, J.A., Cornelio, A., Navathe, S.B.: A Tool for Integrating Conceptual Schemas and User Views. In: Proceedings of the Fourth International Conference on Data Engineering, pp. 176–183. IEEE Computer Society, Washington, DC, USA (1988), http://portal.acm.org/citation.cfm?id=645473.653395
15. Suman, B.: Simulated annealing based multiobjective algorithm and their application for system reliability. Engin. Optim., 391–416 (2003)
16. Suman, B., Kumar, P.: A survey of simulated annealing as a tool for single!'/b?' and multiobjective optimization. Journal of the Operational Research Society 57, 1143–1160 (2006)
17. Zitzler, E., Thiele, L.: Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study, pp. 292–301. Springer, Heidelberg (1998)

To Cache or Not To Cache: The Effects of Warming Cache in Complex SPARQL Queries

Tomas Lampo[1], María-Esther Vidal[2], Juan Danilow[2], and Edna Ruckhaus[2]

[1] University of Maryland, College Park, USA
tlampo@cs.umd.edu
[2] Universidad Simón Bolívar, Caracas, Venezuela
{mvidal,jdanilow,ruckhaus}@ldc.usb.ve

Abstract. Existing RDF engines have developed caching techniques able to store intermediate results and reuse them in further steps of the query execution process; thus, execution time is speeded up by avoiding repeated computation of the same results. Although these techniques can be beneficial for many real-world queries, the same effects may not be observed in complex queries. Particularly, queries comprised of a large number of graph patterns that require the computation of large sets of intermediate results that cannot be reused, or queries that require complex computations to produce small amounts of data, may require further re-orderings or groupings in order to make an effective usage of the cache. In this paper, we address the problem of determining a type of SPARQL queries that can benefit from caching data during query execution or warming up cache. We report on experimental results that show that complex queries can take advantage of the cache, if they are reordered and grouped according to small-sized star-shaped groups; complex queries are not only comprised of a large number of patterns, but they may also produce a large number of intermediate results. Although the results are preliminary, they clearly show that star-shaped group queries can speed up execution time by up to three orders of magnitude when they are run in warm cache, while original queries may exhibit poor performance in warm cache.

1 Introduction

SPARQL has been defined as a standard query language for RDF and several query engines have been defined to store and retrieve RDF data [3,10,11,13,14,21,28,31]. The majority of these approaches have implemented optimization techniques and efficient physical operators able to speed up execution time [3,16,21,28]. Additionally, some of these approaches have implemented structures to efficiently store and access RDF data, and have developed execution strategies able to reuse data previously stored in the cache. In this paper we focus on the study of the benefits of this last feature supported by RDF engines such as RDF-3X[22], BitMat [4], MonetDB[12], and RDFVector [19]; we study the types of queries that can benefit from caching intermediate results as well as the benefits of maintaining them in cache, or warming up cache.

The Database community has extensively studied the benefits of caching techniques to enhance the performance of queries by using recently accessed data [6,15,17,35]. On the other hand, in the context of the Semantic Web, recent publications have reported great benefits of running queries in warm or hot caches, i.e., the effects of maintaining the cache populated with valid data previously generated[4,19,22]; also, techniques to cache the most useful previous computed results have been proposed [18,33,34]. Particularly, RDF-3X and MonetDB have been reported as engines with high performance in warm cache. The behavior of RDF-3X is due to the fact that it has developed optimization and execution techniques, which in conjunction with compressed indexed structures and caching techniques, provide the basis for efficient executions of a large set of real-world SPARQL queries. In addition, these techniques benefit an efficient usage of previously loaded intermediate results in caches. However, as we will show in this paper, there is a family of queries, comprised of small-sized star-shaped groups of graph basic patterns, where further optimization and execution techniques have to be performed in order to fully exploit the RDF-3X caching features. MonetDB[35] implements a column-based storage manager and lightweight data compression techniques which provide high performance of query-intensive workloads. Additionally, MonetDB offers sophisticated cache management techniques named *in-cache processing*, to efficiently control previously cached data. Finally, MonetDB has developed a *vectorized execution* model that contrary to the traditional pipeline iterator model, is able to pass entire vectors of values through the pipeline and exploits the properties of the query engine.

Motivated by these data management features and the results reported in the literature [4,21,35], we studied the impact of the shape of SPARQL queries on the performance of RDF-3X and MonetDB, when queries are run in both cold and warm caches, i.e., when intermediate results are maintained or not in cache. We show that if these queries are rewritten as bushy plans [1] comprised of small-sized star-shaped groups, the cold cache execution time can be reduced by up to three orders of magnitude when the query is run in warm cache, while the performance of original queries may not follow the same trend. Based on these results and the fact that the RDF-3X and MonetDB are not tailored to identify or execute bushy plans comprised of small-sized star-shaped groups, we have enabled both engines to support bushy plans. First, RDF-3X engine was modified to accept plans of any shape and assign particular operators to join small-sized sub-queries in a bushy plan; we call this new version GRDF-3X. Furthermore, we implemented a translation schema to convert SPARQL bushy plans into nested SQL queries, which enforces MonetDB to execute the plan in a bushy fashion. We will describe the optimization techniques [16] that benefit the generation of bushy plans, which exploit the usage of previously cached intermediate results.

To summarize, the main contributions of this paper are the following:

- We define a family of queries that can benefit from caching intermediate results or warming up cache. These queries reduce the number of intermediate results and CPU processing, and can be rewritten as bushy plans comprised of small-sized star-shaped groups.
- We describe GRDF-3X, an extension of the RDF-3X engine able to efficiently evaluate bushy plans comprised of small-sized star-shaped groups.

[1] A bushy plan corresponds to a query plan where operands of the join operators can be intermediate results produced by other operators of the plan.

- We explain the schema translation of bushy plans into nested SQL queries that are executed against vertical partitioned tables in MonetDB.
- We provide an empirical analysis of the performance of the RDF-3X, GRDF-3X and MonetDB engines when queries are evaluated in both cold and warm caches.

This paper is comprised of five additional sections. Section 2 describes existing state-of-the-art approaches in conjunction with an analysis of the advantages and limitations of each approach. Section 3 illustrates a motivating example; section 4 presents the main features that characterize the small-sized star-shaped group queries. Section 5 presents an experimental study where we report on the performance of the small-sized star shaped group queries. Finally, we conclude in section 6 with an outlook to future work.

2 Related Work

During the last years, several RDF stores have been developed [3,10,11,13,14,21,31]. Jena [13,32] provides a programmatic environment for SPARQL; it includes the ARQ query engine and indices, which provide efficient access to large datasets. Tuple Database or TDB [14] is a persistent graph storage layer for Jena; it works with the Jena SPARQL query engine (ARQ) to support SPARQL together with a number of extensions (e.g., property functions, aggregates, arbitrary length property paths). Sesame [31] is an open source Java framework for storing and querying RDF data; it supports SPARQL and SeRQL queries. Additionally, different storage and access structures have been proposed to efficiently retrieve RDF data [7,20,29,30]. Hexastore [30] is a main memory indexing technique that exploits the role of the arguments of an RDF triple; six indices are designed so that each one can efficiently retrieve a different access pattern; a secondary-memory-based solution for Hexastore has been presented in [29]. Fletcher et al. [7] propose indexing the universe of RDF resource identifiers, regardless of the role played by the resource. Although all of the former approaches propose different strategies to speed up the execution time of RDF queries, none of them provide techniques to manage the cache or to load RDF data into resident memory which allow the observation of differences between cold and warm cache execution times.

Additionally, MacGlothlin et al. [19] propose an index-based representation for RDF documents that materializes the results for subject-subject joins, object-object joins and subject-object joins. This approach has been implemented on top of MonetDB [12] and it can exploit the Monet DB cache management system. Abadi et al. [1,2] and Sidirourgos et al. [27] propose different RDF store schemas to implement an RDF management system on top of a relational database system. They empirically show that a physical implementation of vertical partitioned RDF tables may outperform the traditional physical schema of RDF tables. In addition, any of these solutions can exploit the properties of the database manager to efficiently manage the cache.

Recently, Atre et al. [4] proposed the BitMat approach which is supported on a fully inverted index structure that implements a compressed bit-matrix structure of the RDF data. An RDF engine has been developed on top of this bit-based structure, which exploits the properties of this structure and avoids the storage of intermediate results generated during query execution. Although BitMat does not use cache techniques, it has been shown that its performance is competitive with existing RDF engines that provide efficient cache management. Finally, RDF-3X [21] focuses on an index system, and has implemented optimization and execution techniques that support efficient and scalable execution of RDF queries. In addition, RDF-3X makes use of the Linux `mmap` system call, to load in resident memory portions of data, and thus, differences between execution time in both cold and warm caches can be observed for certain types of queries. In this paper we show how cache data management features implemented by RDF-3X and MonetDB, can be better exploited if queries are executed in a way that intermediate results are minimized.

3 Motivating Example

In this section we illustrate how the shape of a query plan can affect the performance of a SPARQL query when it is run in both cold and warm caches. SPARQL syntax resembles SQL queries where the `Where` clause is comprised of Basic Graph Patterns connected by diverse operators, e.g., join, optional, or union. Consider the RDF dataset YAGO (Yet Another Great Ontology)[2] that publishes information about people, organizations and cities around the world. Suppose a user is interested in finding groups of at most two artists who are influenced by at least one person. Figure 1 presents a SPARQL query against YAGO; the query is composed of 8 basic graph patterns connected by the join operator denoted by a "."; suppose the RDF-3X engine is used to run the query.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX yago:<http://mpii.de/yago/resource/yago>
SELECT ?A1 ?A2 WHERE
    {?A1 yago:hasFamilyName ?fn1.
    ?A1 yago:hasGivenName ?gn1 .
    ?person1 yago:influences ?A1.
    ?A2 yago:hasFamilyName ?fn2 .
    ?A2 yago:hasGivenName ?gn2 .
    ?person1 yago:influences ?A2.
    ?A1 rdf:type yago:wordnet_artist_109812338.
    ?A2 rdf:type yago:wordnet_artist_109812338.}
```

Fig. 1. SPARQL Query

Table 1 reports on the time spent by RDF-3X in a Sun Fire X4100 M2 machine with two AMD Opteron 2000 Series processors, 1MB of cache per core and 8GB RAM, running a 64-bit Linux CentOS 5.5 kernel. We can see that RDF-3X consumes 2.220 secs to evaluate the query in cold cache while the time is reduced to 0.11 secs by warming up the cache; this reduction represents 95% of the cold cache execution time, i.e., when the query is run and no data have been loaded in cache. This number is consistent with the results recently reported in the literature, where query execution time in warm cache can be reduced up to one order of magnitude with respect to cold cache execution time [4,19,22].

[2] Ontology available for download at *http://www.mpi-inf.mpg.de/yago-naga/yago/*

Table 1. Run-Time Cold and Warm Cache (secs)

Cold Cache	Warm Cache		
	Mean	Standard Deviation	Geometric Mean
2.220	0.112	0.004	0.112

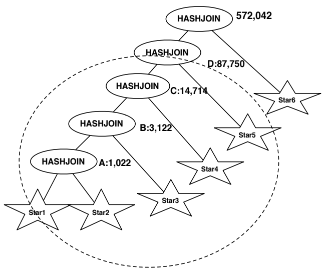
Additionally, let's consider a complex version of this query, where the user is interested in groups of at most six artists that are influenced by at least one person (Figure 2(a)). For this complex query, the behavior of the RDF-3X engine is different; the query execution time is 850.24 secs in cold cache while during warm cache, the time is reduced to 836.09 secs; thus, the savings are less than 2%. However, one can observe that almost 96% of the query execution time corresponds to optimization time in both cold and warm caches, i.e., the time to generate the plan presented in Figure 2(b).

In this plan, internal nodes correspond to the physical operator `HASH JOIN`. The leaves correspond to star-shaped groups that share exactly one variable; the `MERGE JOIN` physical operator is used to evaluate each of the stars in the query. We denote by Star$_i$ the star-shaped group comprised of the basic graph triple patterns presented in Figure 3(a); the plan produced by RDF-3X is reported in Figure 3(b). First, the merge join is used to evaluate the join between the basic graph triple patterns; the index scan is used to access RDF-3X to recover the instantiations of the variables in the query.

Clearly, if the plan is created in way that the first pattern in the star is very selective, then the number of matched triples is reduced and the chances to locate intermediate results in cache increase. RDF-3X executes this physical plan in 48.51 secs in cold cache and 40.60 secs in warm cache; thus, if only the plan execution time is measured, then the observed savings increase to 16.30%; however, this improvement is still low. The poor performance of the RDF-3X engine in this plan can be a consequence of

```
SELECT ?A1 ?A2 ?A3 ?A4 ?A5 ?A6 WHERE
{?A1 yago:hasFamilyName ?fn1. ?A1 yago:hasGivenName ?gn1 .
?person1 yago:influences ?A1.?A2 yago:hasFamilyName ?fn2 .
?A2 yago:hasGivenName ?gn2 . ?person1 yago:influences ?A2.
?A3 yago:hasFamilyName ?fn3. ?A3 yago:hasGivenName ?gn3 .
?person1 yago:influences ?A3.?A4 yago:hasFamilyName ?fn4 .
?A4 yago:hasGivenName ?gn4 . ?person1 yago:influences ?A4.
?A5 yago:hasFamilyName ?fn5. ?A5 yago:hasGivenName ?gn5 .
?person1 yago:influences ?A5.?A6 yago:hasFamilyName ?fn6 .
?A6 yago:hasGivenName ?gn6 . ?person1 yago:influences ?A6.
?A1 rdf:type yago:wordnet_artist_109812338.
?A2 rdf:type yago:wordnet_artist_109812338.
?A3 rdf:type yago:wordnet_artist_109812338.
?A4 rdf:type yago:wordnet_artist_109812338.
?A5 rdf:type yago:wordnet_artist_109812338.
?A6 rdf:type yago:wordnet_artist_109812338.}
```

(a) SPARQL Query

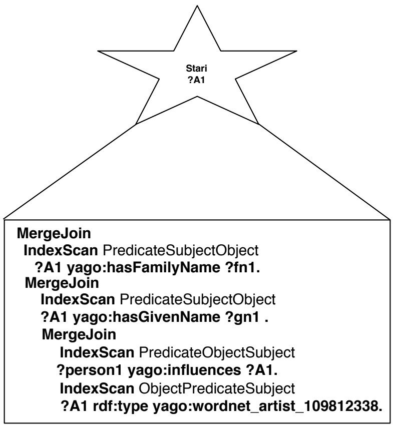


(b) RDF-3X Plan

Fig. 2. SPARQL Query and RDF-3X Plan

{?A$_i$ yago:hasFamilyName ?fn$_i$.
?A$_i$ yago:hasGivenName ?gn$_i$.
?person1 yago:influences ?A$_i$.
?A$_i$ rdf:type yago:wordnet_artist_109812338.}

(a) Star-shaped group Star$_i$



```
MergeJoin
 IndexScan PredicateSubjectObject
   ?A1 yago:hasFamilyName ?fn1.
 MergeJoin
   IndexScan PredicateSubjectObject
   ?A1 yago:hasGivenName ?gn1 .
    MergeJoin
      IndexScan PredicateObjectSubject
     ?person1 yago:influences ?A1.
      IndexScan ObjectPredicateSubject
       ?A1 rdf:type yago:wordnet_artist_109812338.
```

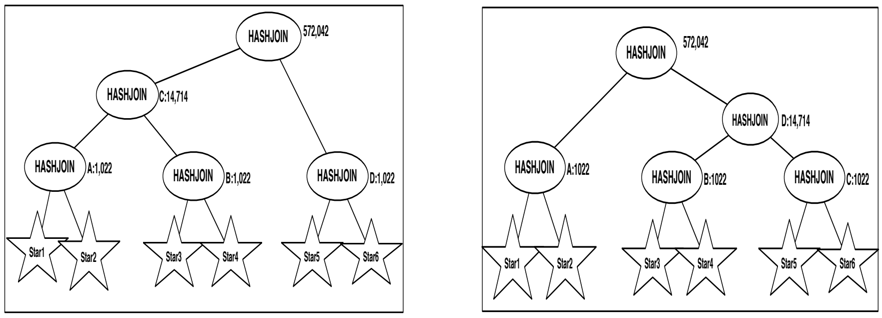
(b) RDF-3X Plan for a star-shaped group

Fig. 3. Star-Shaped Group and an RDF-3X Plan for a Star

the way this plan is evaluated. The plan is a left linear plan and generates 106,608 intermediate result triples. Particularly, the sub-plan surrounded by the circle produces 87,750 triples that may cause page faults, which degrades the performance of the RDF-3X engine.

On the other hand, if the query had been evaluated in a different fashion, the number of intermediate results could be reduced. For example, consider the plans presented in Figure 4. These plans are bushy and are comprised of star-shaped groups of small cardinality. The number of intermediate triples in any of these plans is only 17,780.

Given that any of these bushy plans maintains less triples in cache during execution time, cold cache execution time can be reduced, and the performance in warm cache is improved. The cold cache execution time for the plan in Figure 4(a) is 26.82 secs, while



(a) Bushy Plan 1

(b) Bushy Plan 2

Fig. 4. Different Bushy Plans

it consumes 14.03 secs in warm cache. Similarly, the plan in Figure 4(b) runs in 19.07 secs in cold cache and in 13.60 secs in warm cache; thus, the savings in warm cache are 47.68% and 28.68%, respectively. In this paper we illustrate the benefits of executing bushy trees in both cold and warm caches.

4 Star-Shaped Group Queries

We have developed optimization and execution techniques to support the execution of complex SPARQL queries that can be decomposed in small-sized star-shaped queries [16,28]. A star-shaped query is the join of multiple basic graph patterns that share exactly one variable or a star-shaped basic graph pattern w.r.t. ?X $(?X^* - BGP)$. This type of query is very likely to be found in real-world scenarios and can be formally defined as follows:

Definition 1 (Star-Shaped Basic Graph Pattern w.r.t. ?X, $?X^* - BGP$ **[28]).** *Each triple pattern* $\{?X\ p\ o\ \}$ *or* $\{s\ p\ ?X\ \}$ *such that* $s \neq ?X$*,* $p \neq ?X$ *and* $o \neq ?X$*, is a* star-shaped basic graph pattern w.r.t. ?X *denoted by* $?X^*$*-BGP. Let P and P′ be* $?X^*$*-BGPs such that,* $var(P) \cap var(P') = \{?X\}$ *then,* $P \cup P'$ *is star-shaped basic graph pattern w.r.t. ?X, i.e., an* $?X^*$*-BGP.*

Star-shaped basic graph patterns in Figure 3(a) correspond a $?A1^*$-BGP.

We have developed four different strategies (operators) that are used to retrieve and combine intermediate generated RDF triples of small-sized star shaped groups.

1. Index Nested-Loop Join (**njoin**): For each matching triple in the first pattern, we retrieve the matching triples in the second pattern, i.e., the join arguments[3] are instantiated in the second pattern through the sideways passing of variable bindings. When the data is indexed, the operator can exploit these indices with the instantiations of the second pattern to speed up the execution task. Time complexity in terms of I/O's of the **njoin** between sub-queries A and B on join variables JV, is expressed by the following formula:

$$Cost_{njoin}(A, B, JV) = Cost(A) + \#Instantiations(JV) \times Cost(B)$$

   The first term of the formula represents the cost of evaluating the outer sub-query of the join, while the second term counts the cost of executing B for each instantiation of the join variables JV. If tuples of B in the instantiations of JV that satisfy A can be maintained in cache, the number of I/O's will be reduced and in consequence, the cost of the operator will be reduced. So, the optimizer should try to select this operator, only when the number of instantiations of JV is small.
2. Group Join (**gjoin**): Given two groups, each of them is independently evaluated, and the results are combined to match the compatible mappings. The gjoin operator can take advantage of the cache, because intermediate results previously loaded in cache can be reused without the need to execute the operations required to compute them; however, the size of the results should be small to avoid page faults. Time

[3] The join arguments are the common variables in the two predicates that represent the patterns.

complexity in terms of I/O's of the **gjoin** between sub-queries A and B on join variables JV, is expressed by the following formula:

$$Cost_{gjoin}(A, B, JV) = Cost(A) + Cost(B) + 2 \times (Card(A) + Card(B))$$

The first and second terms of the formula represent the cost of evaluating the outer and inner sub-queries of the join. The third term counts the cost of storing and retrieving from disk the intermediate results produced by A and B, assuming the worst case when matches are done in hash tables previously flushed to secondary memory. In case cardinalities of A and B are minimized, results produced by A and B can be retained in cache, avoiding the cost of storing and retrieving intermediate results from disk.

3. Star-Shaped Group Evaluation (**sgroup**): the main idea is to evaluate the first pattern in the star-shaped group and identify the bindings/instantiations of the shared variable. These instantiations are used to bind the rest of the patterns in the group. This operator can be very efficient if the first pattern in the group is very selective, i.e., there are only few valid instantiations of the shared variable, and the rest of the patterns are indexed by these instantiations.
   Time complexity in terms of I/O's of the **sgroup** between basic graph triple patterns $A_1, A_2, A_3, ..., A_n$ on shared variable JV, is expressed by the following formula:

   $$Cost_{sgroup}(A_1, (A_2, ..., A_n), JV) = Cost(A_1) + \#Instantiations(JV) \times Cost((A_2, ..., A_n))$$

   The first term of the formula represents the cost of evaluating the first basic graph triple pattern; the second corresponds to the cost of executing the rest of the basic graph triple patterns with the instantiations produced by the execution of the first. The third term counts the cost of storing and retrieving from disk the intermediate results produced by executing each graph basic pattern, assuming the worst case when matches have been flushed to secondary memory. In case the number of instantiations of the shared variable JV is minimized, results produced by the execution of the rest of the basic graph patterns can be retained in cache, avoiding the cost of storing and retrieving intermediate results from disk.
4. Index Star-Shaped Group Evaluation (**isgroup**): In the case that all of the patterns in the group are indexed, the valid instantiations of each pattern can be independently retrieved and merged together to produce the output. For example, in the plan in Figure 4(a), `Star1` could be evaluated by searching the instantiations of the variable `?A1` that correspond to artists influenced by at least one person, the instantiations that have a given name, and the ones that have a given last name. These sets of instantiations are merged to produce the star-shaped group answer. This operator can benefit from running in warm cache if the number of valid instantiations to be merged, is small because computations of the join between the two basic graph patterns could be stored in cache, and reused to compute the join with the third basic graph pattern.

   Time complexity in terms of I/O's of the **isgroup** between basic graph triple patterns A_1, A_2, A_3, ..., A_n on shared variable JV, is expressed by the following formula:

$$Cost_{isgroup}(A_1, (A_2, ..., A_n), JV) = Cost(A_1) + Cost(A_2) + ... + Cost(A_n) +$$
$$2 \times (Card(A_1) + Card(A_2) + ... + Card(A_n))$$

The first n terms of the formula represent the cost of evaluating the basic graph triple patterns. The last term counts the cost of storing and retrieving from disk the intermediate results produced by evaluating the basic graph triple patterns, assuming the worst case when matches are flushed to secondary memory. In case cardinalities of the instantiations of $A_1, A_2, ..., A_n$ are minimized, intermediate results can be retained in cache, avoiding the cost of storing and retrieving intermediate results from disk.

We have implemented these four operators in our own RDF engine named OneQL [16]. Although execution time and memory usage are reduced, OneQL is implemented in Prolog and its performance cannot compete with state-of-the-art RDF engines such as RDF-3X. To fairly compare the performance of the plans comprised of these operators, we have extended the RDF-3X engine and called it GRDF-3X.

First, we modified the RDF-3X parser to consider all given plans; the original RDF-3X parser completely ignores groups and parentheses, and it flattens any input plan. Additionally, GRDF-3X exclusively assigns the RDF-3X `HASH JOIN` operator to evaluate **gjoin**, while **njoin**, **isgroup** and **sgroup** are evaluated with `MERGE JOIN`s; each basic graph pattern is evaluated by using `INDEX SCAN`. GRDF-3X reorders the patterns in a star-shaped group, but no star-groups are further identified. Based on these extensions, GRDF-3X can evaluate the plans presented in Figures 2(b) and 4, and it exploits the properties of these plans that were illustrated in Section 2. In case star-shaped groups are small-sized, the performance of RDF-3X in warm caches can be improved in several orders of magnitude.

Furthermore, we followed the *vertical partitioning approach* [1] to implement an RDF dataset as a relational database in MonetDB. For each property *P*, a table of two columns is defined; the first column stores the values of the subjects associated with *P*, while the second column stores the object values; indices are created on the two columns. A *dictionary encoding* is used to store integer keys instead of the string values of the subjects, properties, and objects. A table `Dictionary(ID,val)` maintains the encodings. SPARQL queries are translated into SQL by evaluating each triple pattern {`s P o`} in a SPARQL query as conditions `P.S` or `P.O` in the SQL query. These conditions can be placed in the `Where` clause or `Select` clause depending on the values of `s` or `o`. If `s` (res, `o`) is a constant, then the condition `P.S=id(s)` (res, `P.O=id(o)`) is added to the `Where` clause, where `id(s)` represents the encoding of `s`. If `s` (res, `o`) is a variable that also appears in another triple pattern, say, {`s P1 o'`}, and they are connected through the SPARQL operator `AND`, then the condition `P.S=P1.S` is added to the `Where` clause; similarly, if both patterns are connected by an `OPTIONAL`, a left outer join relates the conditions `P.S` and P1.S. Thus, the SPARQL operators `AND` and `OPTIONAL` are translated into a relational join and a left outer join, respectively. Finally, if both triple patterns are related through `UNION`, the condition `P.S` (res, `P.O`) is added to the `Select` clause; the SPARQL operator `UNION` is expressed as a SQL `union`. In case the SPARQL query corresponds to a bushy plan, we follow the same translation schema for each star-shaped group of triple patterns. For a **gjoin** between sub-queries S_1 and S_2, a SQL query is created by adding to the `FROM` clause of the query, the SQL sub-queries that result from translating S_1 and S_2 with the alias s_1 and s_2; if the **gjoin** condition `JV` is on the variables $A_1, ..., A_n$, the conditions $s_1.A_1{=}s_2.A_2$ `AND .... AND` $s_1.A_n{=}s_2.A_n$ are added to the `Where` condition of the SQL query; the **njoin** is translated as the SQL join. Figure 5 illustrates the proposed translation schema.

```
SELECT ?A1 ?A2 where
    {{?A1 yago:hasFamilyName ?fn1.
    ?A1 yago:hasGivenName ?gn1 .}
    GJOIN
    {?A2 yago:hasFamilyName ?fn2 .
    ?A2 yago:hasGivenName ?gn2 .}}
```

(a) SPARQL Bushy Plan

```
SELECT s1.A1, s2.A2
FROM
        (Select HFN.S as A1
        from hasFamilyName as HFN, hasGivenName as HGN
        where HFN.S=HGN.S ) as s1,
        (Select HFN.S as A2
        from hasFamilyName as HFN, hasGivenName as HGN
        where HFN.S=HGN.S ) as s2
WHERE s1.A1=s2.A2
```

(b) SQL Query

Fig. 5. Translation Schema-SPARQL into SQL

Finally, we have developed query optimization techniques able to identify query plans comprised of small-sized star-shaped groups. These techniques have been developed in the OneQL System on top of the following two sub-components [16,23,24,25]: (a) a hybrid cost model that estimates the cardinality and execution cost of execution plans, (b) optimization strategies to identify plans comprised of small-sized star-shaped groups. The proposed optimization techniques are based on a cost model that estimates the execution time of intermediate RDF triples generated during query execution, and are able to identify execution plans of any shape. Re-orderings and groupings of the basic graph patterns are performed to identify star-shaped groups of small size. In addition, physical operators are also assigned to each join and star-shaped group. The optimizer is implemented as a Simulated Annealing randomized algorithm which performs random walks over the search space of bushy query execution plans. Random walks are performed in stages, where each stage consists of an initial *plan generation step* followed by one or more *plan transformation steps*. An equilibrium condition or a number of iterations determines the number of transformation steps. At the beginning of each stage, a query execution plan is randomly created in the plan generation step. Then, successive *plan transformations* are applied to the query execution plan in order to obtain new plans. The probability of transforming a current plan p into a new plan p' is specified by an acceptance probability function $P(p, p', T)$ that depends on a global time-varying parameter T called the *temperature*; it reflects the number of stages to be executed. The function P may be nonzero when $cost(p') > cost(p)$, meaning that the optimizer can produce a new plan even when it is worse than the current one, i.e., it has a higher cost. This feature prevents the optimizer from becoming stuck in a local minimum. Temperature T is decreased during each stage and the optimizer concludes when $T = 0$. Transformation rules applied to the plan during the random walks correspond to the SPARQL axioms of the physical operators implemented by the query engine. The axioms state properties such as: commutativity, associativity, distributivity of gjoins over njoins, and folding and unfolding of star-shaped groups. These transformation rules are fired according to probabilities that benefit the generation of bushy plans comprised of small-sized star-shaped groups [28]. These plans usually reduce intermediate results as well as the execution time in both cold and warm caches.

5 Experimental Study

We conducted an experimental study to empirically analyze the effects of caching intermediate results during the execution of simple and complex SPARQL queries. We report on the execution time of MonetDB Apr2011 release, RDF-3X version 0.3.4 and GRDF-3X built on top of RDF-3X version 0.3.4. Particularly, we analyze the impact on the execution time performance of running bushy plans comprised of small-sized star-shaped groups in both cold and warm caches.

Benchmarking has motivated the evaluation of these query engines, and contributed to improve scalability and performance [9]. Among the most used benchmarks, we can mention: LUBM [8], the Berlin SPARQL Benchmark [5], the RDF Store Benchmarks with DBpedia[4], and the SP^{2}Bench SPARQL benchmark [26]. Similarly to existing benchmarks, we tailored a family of queries that allow us to reveal the performance of a state-of-the-art RDF engine, and we focus on illustrating the impact of the shape of query plans on the performance of the query engine in warm caches. During the definition of our benchmarks of queries, query shape, number of basic graph patterns, selectivity of the instantiations, and size of intermediate results were taken into account.

Datasets and Query Benchmark: We used the real-world ontology YAGO which is comprised of around 44,000,000 RDF triples. We developed two sets of queries[5]:
- Benchmark 1 has 9 simple queries, which are comprised of between 3 and 5 basic patterns (Figure 6(a)).
- Benchmark 2 has 9 queries, which are comprised of between 17 and 26 basic patterns (Figure 6(b)).

Additionally, we consider the LinkedCT dataset[6] which exports information of clinical trials conducted around the world; this dataset is composed of 9,809,330 RDF triples. We define a benchmark 3 comprised of 10 queries over LinkedCT; queries are composed of between 13 and 17 patterns. [7]

Evaluation Metrics: We report on runtime performance that corresponds to the *real time* produced by the *time* command of the Linux operation system. Runtime represents the elapsed time between the submission of the query and the output of the answer; optimization time just considers the time elapsed between the submission of the query and the output of the query physical plan. Experiments were run on a Sun Fire X4100 M2 machine with two AMD Opteron 2000 Series processors, 1MB of cache per core and 8GB RAM, running a 64-bit Linux CentOS 5.5 kernel. Queries in benchmark 1, 2 and 3 were run in cold cache and warm cache. To run cold cache,

[4] http://www4.wiwiss.fu-berlin.de/benchmarks-200801/
[5] *http://www.ldc.usb.ve/˜mvidal/OneQL/datasets/queries/YAGO/*
[6] *http://LinkedCT.org*
[7] *http://www.ldc.usb.ve/˜mvidal/OneQL/datasets/queries/LinkedCT/*

we cleared the cache before running each query by performing the command `sh -c "sync ; echo 3 > /proc/sys/vm/drop_caches"`. To run on warm cache, we executed the same query five times by dropping the cache just before running the first iteration of the query; thus, data temporally stored in cache during the execution of iteration i could be used in iteration $i + 1$. Additionally, the machine was dedicated exclusively to run these experiments.

query	#patterns	answer size
q1	4	10
q2	3	1
q3	3	4
q4	5	6
q5	3	2,356
q6	3	1,027
q7	3	5,683
q8	3	46
q9	3	1

(a) Benchmark 1

query	#patterns	answer size
q1	17	1,170
q2	21	4,264
q3	26	22,434
q4	17	238
q5	21	516
q6	26	1,348
q7	17	342
q8	21	1,220
q9	26	5,718

(b) Benchmark 2

Fig. 6. Query Benchmark Description

5.1 Performance of Star-Shaped Groups in Cold and Warm Cache

In attempting to identify the types of queries that can benefit from running in warm cache, we ran queries of Benchmark 1 in both cold and warm caches. These queries are comprised of a simple star-shaped group and were executed in RDF-3X and MonetDB. Table 2 reports on the execution time of cold cache, and the minimum value observed during the execution in warm cache; it also reports on the geometric means. We can see that both engines were able to improve performance if valid data is already loaded in cache. RDF-3X is able to improve cold cache execution times by a factor of 35 in the geometric mean when the queries are run in warm cache. MonetDB improves cold cache execution time by a factor of 31 in the geometric mean. Optimization time is negligible because the optimizer only has to reorder the patterns of the stars in each query. Additionally, the time to translate SPARQL queries into the MonetDB representation is not considered. In both cases, the majority of the execution time in warm cache was dominated by recovering the instantiations of shared variables of the star-shaped groups, that were maintained in memory during warm cache.

Then, we studied the performance of queries in Benchmark 2, which can be rewritten as bushy trees comprised of several small-sized star-shaped groups. Tables 3 and 4 report on cold cache execution times, the minimum value observed during the execution in warm cache, and the geometric means. Similarly to the previous experiment, queries were run in RDF-3X and MonetDB. Additionally, three optimized versions of the queries were run in GRDF-3X; one was created *by hand*, the other was generated by our OneQL query optimizer [16], and the last one was generated by RDF-3X. Furthermore, bushy plans that correspond to the hand-created plans were translated in SQL nested queries following the translation schema presented in the previous section.

Table 2. Benchmark 1-Run-Time RDF-3X and MonetDB in Cold and Warm Cache (secs)

Cold Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
RDF-3X	0.53	0.22	0.25	0.20	3.57	2.25	5.01	0.61	0.29	0.70
MonetDB	0.88	0.45	0.64	0.51	0.76	0.59	0.98	0.46	0.67	0.63
Warm Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
RDF-3X	0.012	0.042	0.012	0.015	0.095	0.058	0.150	0.012	0.011	0.02
MonetDB	0.01	0.01	0.02	0.02	0.04	0.08	0.05	0.01	0.01	0.02

For queries in Benchmark 2, we could observe that RDF-3X performs poorly in warm cache; cold cache times improve by nearly a factor of 1.2 in the geometric means. This behavior of the RDF-3X engine may be because the execution time of queries in Benchmark 2 is dominated by CPU-intensive processing that consumed up to 98% of the CPU time. Additionally, a large portion of the execution time was spent in query optimization and the generation of the physical plan. The reason for this is that the RDF-3X optimizer relies on a dynamic-based programming algorithm that is not able to efficiently scale up to complex queries. Finally, although plans generated by RDF-3X were comprised of small-sized star-shaped groups, they were shaped as left-linear plans, which generate a large number of intermediate results that may produce page faults.

On the other hand, we evaluated three optimized versions of the queries in Benchmark 2 in GRDF-3X: (1) optimal plans that were generated *by hand*; (2) OneQL plans that were produced by the OneQL optimizer; (3) plans generated by RDF-3X. The two first groups of optimized queries were shaped as bushy trees comprised of small-sized star-shaped groups, and ran in GRDF-3X in a bushy fashion such that the number of intermediate results was minimized. The plans generated by RDF-3X were also composed of small-sized star-shaped but combined in a left-linear tree fashion in which intermediate results were not minimal; execution times of these plans allow to illustrate RDF-3X execution time without considering optimization time. First, we could observe that the execution of the two first types of queries consumed up to 25% of the CPU time when they were run in GRDF-3X; the execution time in both cold and warm caches was reduced by up to five orders of magnitude. Also, the optimization time was negligible because GRDF-3X respected the groups in the input plan, and it only had to reorder the patterns of the stars in each query. Finally, because these two groups of plans were bushy trees comprised of small-sized star-shaped groups, the number of intermediate results was smaller; thus, intermediate results could be maintained in resident memory and used in further iterations. The GRDF-3X performance in warm cache was consistently good for hand-optimized queries; it could reduce the cold cache run time by a factor of 5 in the geometric means. For OneQL query plans, GRDF-3X reduced the cold cache run time by nearly a factor of 2.7 in the geometric means. Finally, RDF-3X generated plans exhibit a performance in warm cache that reduces execution time in cold cache by a factor of 1.88.

Furthermore, we can observe that MonetDB also performs poorly when the original query is executed in warm cache; cold cache times are improved by nearly a factor of 1.06 in the geometric means; Table 4 reports on MonetDB runtime. This observed

Table 3. Benchmark 2- RDF-3X Run-Time Cold and Warm Cache (secs)

Cold Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
RDF-3X	62.30	84.87	100,657.34	85.95	61.2	188,909.69	0.14	1.47	827.75	166.03
GRDF-3X (Optimal Plan)	1.60	1.80	2.34	1.22	1.38	1.36	0.99	1.05	1.75	1.45
GRDF-3X (OneQL Plan)	1.64	10.85	3.8	1.28	9.2	3.8	1.18	2.62	3,57	3.18
GRDF-3X (RDF-3X Plan)	60.92	56.16	93,010.25	60.35	59.93	183,291.76	1.34	1.7	2.64	102.68
Warm Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
RDF-3X	58.21	59.54	72,584.71	58.52	59.73	175,909.80	0.14	1.46	808.77	144.13
GRDF-3X (Optimal Plan)	0.34	0.26	0.93	0.14	0.31	0.17	0.12	0.31	0.69	0.29
GRDF-3X (OneQL Plan)	0.40	7.08	2.18	0.36	7.6	1.52	0.18	0.93	1.64	1.22
GRDF-3X (RDF-3X Plan)	54.42	55.42	71,231.74	58.52	50.33	140,822.38	0.25	0.29	0.64	54.34

behavior reinforces our assumption about the performance of RDF-3X in this set of queries, which are dominated by CPU-intensive processing and generate a large number of intermediate results that may produce page faults. Table 5 reports on the size in bytes of the intermediate results produced during the execution of these queries in MonetDB; these values were reported by the MonetDB tool `mclient`. The original version of q3 could not be executed in `mclient`, and the size of intermediate memory could not be computed. We can observe that optimized queries reduce the size of intermediate results by a factor of 81.24 (q3 is not considered). Additionally, we can observe that MonetDB performs very well executing the optimized queries; runtime was reduced by a factor of 1,196.76. The observed performance supports our hypothesis that optimized queries better exploit the features of both MonetDB and RDF-3X.

Finally, we conducted a similar experiment and ran queries in benchmark 3 against LinkedCT; as in previous experiments, we built an optimal bushy plan *by hand*, each optimal plan was comprised of small-sized sub-queries. In GDRF-3X sub-queries in the optimized plans were executed using the **gjoin** operator implemented in GRDF-

Table 4. Benchmark 2-MonetDB Run-Time Cold and Warm Cache (secs)

Cold Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
Original Query	485.30	2,993.14	3,727.33	128.57	213.03	1,751.56	576.06	3,757.61	2,622.82	1044.10
Opt Plan	0.81	0.80	1.37	0.75	1.17	0.68	0.81	0.65	1.05	0.87
Warm Caches										
	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
Orig Plan	496.71	2,593.59	3,710.13	135.74	205.25	1,536.79	461.82	4,280.02	2,027.63	978.21
Optimal Plan	0.14	0.20	0.54	0.13	0.14	0.17	0.13	0.15	0.24	0.18

Table 5. Benchmark 2-MonetDB Size of Intermediate Results (Bytes)

	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
Original Query	1.13E+11	2.66E+11	N/R	4.22E+10	5.57E+10	2.89E+11	7.12E+10	1.60E+11	3.24E+11	1.28E+11
Optimal Plan	1.38E+9	1.63E+9	1.99E+9	1.38E+9	1.63E+9	1.88E+9	1.37E+9	1.61E+9	1.87E+9	1.58E+9

Table 6. Benchmark 3-Execution RDF-3X Time Cold and Warm Caches (secs)

Cold Caches											
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	Geom. Mean
RDF-3X	6.35	3.55	4.13	1,543.82	3.71	4.36	1,381.9	2.75	3.83	0.51	10.62
GRDF-3X	0.76	0.59	0.51	0.52	0.80	0.73	0.71	0.59	0.51	0.52	0.61
Warm Caches											
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	Geom. Mean
RDF-3X	2.44	2.28	2.41	1,385.09	2.71	1.75	1,321.05	1.74	1.73	0.14	5.87
GRDF-3X	0.14	0.14	0.14	0.18	0.18	0.18	0.18	0.18	0.17	0.18	0.16

3X; SQL nested queries against vertical partitioned tables were generated to be run in MonetDB. Original queries were run in both cold and warm caches. Table 6 reports on cold cache execution times, minimum values observed during the execution in warm cache, and geometric means for RDF-3X and GRDF-3X executions; Table 7 reports on execution times for MonetDB.

We can observe that RDF-3X is able to improve cold cache execution time by a factor of 1.8 in the geometric mean when queries are run in warm cache. However, GRDF-3X performance in warm cache was consistently good for hand-optimized queries; it could reduce the cold cache run time by a factor of 3.81 in the geometric mean when the queries are run in warm cache. In addition, GRDF-3X execution times were reduced by up to four orders of magnitude in both cold and warm caches (queries $q4$ and $q7$) compared to the original query. This is because the plans were bushy trees comprised of small-sized star-shaped sub-queries, where the number of intermediate results was smaller than the original queries. Thus, intermediate results could be maintained in resident memory and used in further iterations.

Finally, we also ran these queries against MonetDB and the results are reported in Table 7; the no optimized version of q10 could not be executed because MonetDB ran out of memory. Similarly to RDF-3X, MonetDB is able to improve cold cache execution time by a factor of 1.28 in the geometric mean when the original queries are run in warm cache. However, the performance in warm cache is very good for optimized queries; the runtime is reduced by a factor of 11.65. Additionally, we can observe that optimized queries reduce runtime of original queries by a factor of 15.24.

These results provide an empirical evidence about the benefits on warm cache performance of the shape and characteristics of the plans. For simple queries, these two engines are certainly able to benefit from warming up cache; however, for queries with several star-shaped groups, the optimizers generate left-linear plans that may produce a large number of intermediate results or require CPU-intensive processing that degrades the query engine performance in both cold and warm caches. Contrary, if these queries are rewritten as bushy plans, the number of intermediate results and the CPU process-

Table 7. Benchmark 3-Execution MonetDB Time Cold and Warm Caches (secs)

Cold Caches											
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	Geom. Mean
Original Query	3.86	4.43	4.82	13.58	13.62	6.04	12.82	13.4	13.17	N/R	8.40
Optimized Plan	2.74	2.68	2.74	11.77	11.87	4.55	11.87	11.87	11.87	N/R	6.52
Warm Caches											
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	Geom. Mean
Original Query	0.66	0.57	0.52	0.49	0.58	0.52	0.59	0.5	0.55	0.5	0.55
Optimized Plan	0.04	0.04	0.04	0.05	0.05	0.06	0.05	0.05	0.05	0.04	0.047

ing can be reduced and the performance improves. Thus, the shape of a query plan can impact on the benefits of caching intermediate results.

6 Conclusions

We have reported experimental results suggesting that the benefits of running in warm cache depend on the shape of executed queries. For simple queries, RDF-3X and MonetDB are certainly able to benefit from warming up cache; however, for complex queries, some plans may produce a large number of intermediate results or require CPU-intensive processing that degrades the query engine performance in both cold and warm caches. We have presented a type of bushy queries comprised of small-sized star-groups that reduce the number of intermediate results and the CPU processing. In this type of queries the performance of RDF-3X and MonetDB is clearly better. These results encouraged us to extend RDF-3X with the functionality of evaluating bushy plans comprised of small-sized star-groups, and to define a translation schema to enforce MonetDB to execute queries in a bushy fashion. In the future, we plan to incorporate our optimization techniques in existing RDF engines.

References

1. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. VLDB Journal 18(2), 385–406 (2009)
2. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 411–422 (2007)
3. AllegroGraph (2009), http://www.franz.com/agraph/allegrograph/
4. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In: Proceedings of the WWW, pp. 41–50 (2010)
5. Bizer, C., Schultz, A.: The berlin sparql benchmark. Int. J. Semantic Web Inf. Syst. 5(2), 1–24 (2009)

6. Bornhövd, C., Altinel, M., Mohan, C., Pirahesh, H., Reinwald, B.: Adaptive database caching with dbcache. IEEE Data Eng. Bull. 27(2), 11–18 (2004)
7. Fletcher, G., Beck, P.: Scalable Indexing of RDF Graph for Efficient Join Processing. In: CIKM (2009)
8. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
9. Guo, Y., Qasem, A., Pan, Z., Heflin, J.: A requirements driven framework for benchmarking semantic web knowledge base systems. IEEE Trans. Knowl. Data Eng. 19(2), 297–309 (2007)
10. Harth, A., Umbrich, J., Hogan, A., Decker, S.: A Federated Repository for Querying Graph Structured Data from the Web. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 211–224. Springer, Heidelberg (2007)
11. Ianni, G., Krennwallner, T., Martello, A., Polleres, A.: A Rule System for Querying Persistent RDFS Data. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 857–862. Springer, Heidelberg (2009)
12. Idreos, S., Kersten, M.L., Manegold, S.: Self-organizing tuple reconstruction in column-stores. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 297–308 (2009)
13. Jena Ontology Api (2009), http://jena.sourceforge.net/ontology/index.html
14. Jena TDB (2009), http://jena.hpl.hp.com/wiki/TDB
15. Kim, S.-K., Min, S.L., Ha, R.: Efficient worst case timing analysis of data caching. In: IEEE Real Time Technology and Applications Symposium, pp. 230–240 (1996)
16. Lampo, T., Ruckhaus, E., Sierra, J., Vidal, M.-E., Martinez, A.: OneQL: An Ontology-based Architecture to Efficiently Query Resources on the Semantic Web. In: The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems at the International Semantic Web Conference, ISWC (2009)
17. Malik, T., Wang, X., Burns, R.C., Dash, D., Ailamaki, A.: Automated physical design in database caches. In: ICDE Workshops, pp. 27–34 (2008)
18. Martin, M., Unbehauen, J., Auer, S.: Improving the Performance of Semantic Web Applications with SPARQL Query Caching. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 304–318. Springer, Heidelberg (2010)
19. McGlothlin, J.: RDFVector: An Efficient and Scalable Schema for Semantic Web Knowledge Bases. In: Proceedings of the PhD Symposium ESWC (2010)
20. McGlothlin, J., Khan, L.: RDFJoin: A Scalable of Data Model for Persistence and Efficient Querying of RDF Dataasets. In: Proceedings of the International Conference on Very Large Data Bases, VLDB (2009)
21. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1), 647–659 (2008)
22. Neumann, T., Weikum, G.: Scalable join processing on very large rdf graphs. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 627–640 (2009)
23. Ruckhaus, E., Ruiz, E., Vidal, M.: Query Evaluation and Optimization in the Semantic Web. In: Proceedings ALPSWS 2006: 2nd International Workshop on Applications of Logic Programming to the Semantic Web and Semantic Web Services (2006)
24. Ruckhaus, E., Ruiz, E., Vidal, M.: OnEQL: An Ontology Efficient Query Language Engine for the Semantic Web. In: Proceedings ALPSWS (2007)

25. Ruckhaus, E., Ruiz, E., Vidal, M.: Query Evaluation and Optimization in the Semantic Web. In: TPLP (2008)
26. Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., Pinkel, C.: An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 82–97. Springer, Heidelberg (2008)
27. Sidirourgos, L., Goncalves, R., Kersten, M.L., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. PVLDB 1(2), 1553–1563 (2008)
28. Vidal, M.-E., Ruckhaus, E., Lampo, T., Martínez, A., Sierra, J., Polleres, A.: Efficiently Joining Group Patterns in SPARQL Queries. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 228–242. Springer, Heidelberg (2010)
29. Weiss, C., Bernstein, A.: On-disk storage techniques for semantic web data are b-trees always the optimal solution? In: The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems at the International Semantic Web Conference, ISWC (2009)
30. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. PVLDB 1(1), 1008–1019 (2008)
31. Wielemaker, J.: An Optimised Semantic Web Query Language Implementation in Prolog. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 128–142. Springer, Heidelberg (2005)
32. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. Exploiting Hyperlinks 349, 35–43 (2003)
33. Williams, G.T., Weaver, J.: Enabling fine-grained http caching of sparql query results. Accepted ISWC (2011)
34. Yang, M., Wu, G.: Caching intermediate result of sparql queries. In: WWW (Companion Volume), pp. 159–160 (2011)
35. Zukowski, M., Boncz, P.A., Nes, N., Héman, S.: Monetdb/x100 - a dbms in the cpu cache. IEEE Data Eng. Bull. 28(2), 17–22 (2005)

Implementation of Updateable Object Views in the ODRA OODBMS

Radosław Adamus, Tomasz Marek Kowalski, and Jacek Wiślicki

Computer Engineering Department, Technical University of Lodz, Poland
{radamus,tkowals,jacenty}@kis.p.lodz.pl

Abstract. The paper describes results of the research effort related to the implementation of the idea of Updateable Object Views for the ODRA project. ODRA (Object Database for Rapid Application development) is a prototype object-oriented database management system based on Stack Based Architecture (SBA) and Stack Based Query Language (SBQL). The updateable object views are one of the main features of the system that has great impact on its features. The paper focuses on the issues connected with views implementation related to external type system and optimization module.

Keywords: updateable views, OODBMS, compilers, optimizers.

1 Introduction

The goal of the paper is to describe the implementation of updateable object views that was made for the ODRA (Object Database for Rapid Application development) project [3]. ODRA is a prototype object-oriented database management system based on Stack Based Architecture (SBA) and Stack Based Query Language (SBQL) [1]. The main goal of the ODRA project is to develop new paradigms of database application development mainly through increasing the level of abstraction at which the programmer works. In ODRA this is achieved by introducing a new, universal, declarative programming language, together with distributed, database-oriented and object-oriented execution environment. One of the main features provided by the ODRA environment is the ability to define arbitrarily updateable object oriented views. Such a powerful tool allows for creating arbitrary complex transformation of data stored in the database to adapt the data interface to specific business needs or to integrate local data according to some external schemas in context of distribute processing. Lack of view update constraints cause that such views become also a background for many interesting researches [2][10][13][18] that results were (or going to be) introduced into the ODRA prototype implementation.

Views update problem has been widely studied in the context of relational and XML databases. The series of papers [4][15] propose a formal treatment of relational view update problem that is based on the concept of a view complement, yet it can be applied to very limited number of relatively simple views, e.g. views providing an aggregate value cannot have an appropriate complement defined. In [11] authors proposed a novel approach for separating the data instance into a logical and a physical level in order to solve a problem of translating view updates in a side-effect free manner. Nevertheless, the designed framework does not include views with aggregations and set operators. In the series of papers Elke Rundensteiner et al. [14][19] address the problem of how to maintain the consistency of materialized relational views under update operations and solution for problems addressing the XML view update specified over relational databases.

It should be emphasized that the paper is not devoted to a problem of updating data tables through relational view. Our researches concentrate on object oriented databases and object oriented views. Although some work in the field of relational views update may appear similar to the SBA Updateable Views. Especially the proposals called bidirectional languages and relational lenses [7]. In this approach every program denotes a pair of functions—one for extracting a view of some complex data structure, and another for putting back an updated view into the original structure. In other words it the update semantics can explicitly introduced by the view definer. Other works extends this to more complicated formations (e.g. aligments) [5]. Our approach is much more general and orthogonal (does not introduces any special semantics that can be used only in limited scope). But the work on relational lenses and our researches [2][10][13] show that the problem of a view update can be solved only through allowing the view designer introducing the definition of view update semantics.

The paper focuses on the updateable views implementation issues connected with type control and view invocation optimization. The rest of it is organized as follows: Section 2 shortly describe the basis of Stack Based Architecture. Section 3 introduces the idea of Updateable Object Views in the SBA. Section 4 describes implementation of Updateable Object Views in the context of ODRA type system. Section 5 presents view rewrite algorithm designed for the ODRA static optimization module. Section 6 focuses on the future of the ODRA system and the research ideas concerning views. Section 7 concludes.

2 Stack-Based Architecture

The Stack-Based Architecture (SBA)[1] is a formal methodology concerning the construction and semantics of database query languages, especially object-oriented. In SBA query languages' concepts are reconstructed from the point of view of programming languages (PLs). The main SBA idea is that there is no clear and final boundary between programming languages and query languages. Therefore, there should arise a theory consistently describing both aspects. SBA offers a complete conceptual and semantic foundation for querying and programming with queries, including programmes with abstractions (e.g., procedures, functions, classes, types, methods, views).

SBA assumes a semantic specification method that is referred to as abstract implementation. It is a kind of operational semantics where one has to determine precisely on an abstract level a query (and program) execution machine. It involves all the data structures that participate in query/program processing and then, specifies the semantics of all the languages' operators in terms of actions on these structures. SBA introduces three such structures that are well-known in the specification of PLs: an object store, an environment stack, and a query result stack (thus the stack-based architecture). Query operators, such as selection, projection, joins and quantifiers, can be precisely specified using the above three abstract structures.

SBA introduces a model query and programming language SBQL (Stack-Based Query Language). SBQL plays the same role as relational algebra for the relational model. The power of SBQL concerns a wide spectrum of data structures that it is able to serve and complete algorithmic power of querying and manipulation capabilities. At the same time, SBQL is fully precise with respect to the specification of semantics. SBQL has been carefully designed from the pragmatic (practical) point of view. The quality of SBQL is achieved by orthogonality of introduced data/object constructors, orthogonality of all the language constructs, object relativism, orthogonal persistence, typing safety, introducing all the classical and some new programming abstractions (procedures, functions, modules, types, classes, methods, views, etc.) and following commonly accepted programming languages' principles.

The functionality of SBQL includes all well-known query operators (selection, projection, navigation, path expressions, join, quantifiers, etc.), some less known operators (transitive closures, fixed-point equations, etc.), imperative (updating) statements integrated with queries, modules, procedures, functions and methods (with parameters being queries and recursive). SBQL deals with static strong type checking and with query optimization methods based on indices, rewriting rules and other techniques.

3 Updateable Object Views

A database view is an image of data stored in a database. Depending on the image representation views divide into materialized and virtual. Materialized view representing selected data in the materialized form (i.e. copy of the data). In contrary a virtual view does not store data directly. Typically a view definition is a procedure that can be invoked in queries. The most important property of views is transparency. The user formulating the query needs not to distinguish between stored and virtual data.

The idea of updateable object views [13][2] relies in augmenting the definition of a view with the information on users' intents with respect to updating operations. Only the view definer is able to express the semantics of view updating. To achieve it, a view definition is divided in two parts – seed of virtual object generator and virtual object operators.

The first part is the functional procedure, which maps stored objects into virtual ones (similarly to SQL). The result of this procedure is a bag of so called seeds. Each seed contains the data required to execute the required update operation on the virtual object or to generate, on demand, virtual sub-object. Thanks to this, only those parts of a virtual object will be materialized that are used in the query.

The second part of a view definition contains procedures that define operations that can be performed on virtual objects. Those procedures express the users' intents with

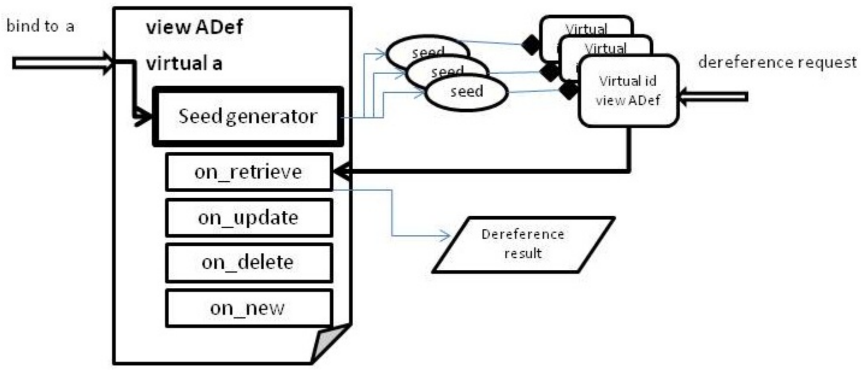


Fig. 1. Sample virtual object lifecycle

respect to update, delete, insert and retrieve (i.e. dereference) operations performed on virtual objects (the number of available operands can be extended according to a domain needs). The seed of a virtual object is an implicit argument passed to a view operator procedures. Figure 1 depicts sample lifecycle of the virtual object.

A view definition usually contains definitions of sub-views, which are defined on the same rule, according to the relativism principle. Because a view definition is a regular complex object, it may also contain other elements, such as procedures, functions, state objects, etc.

4 Updateable Object Views and the ODRA Type System

4.1 ODRA Type Control System

ODRA query/program evaluation involves compile-time type checking [1]. The type-checker is based on the original strong typing theory[16] distinguishing internal and external type systems. The internal type system reflects the behavior of the type checking mechanism, while the external type system is used by the programmer. A static strong type checking mechanism simulates run-time computations during compile time by reflecting the run-time semantics with the precision that is available at the compile time [1]. As the runtime elements are database, environment stack and query result stack, the compiler works on corresponding structures: meta-base, static environment stack and static result stack respectively. From the evaluation point of view runtime stacks process query results (including values, structures, database objects identifiers, binders etc.) and the compile-time stacks process type signatures – typing counterparts of corresponding runtime entities.

Externally the ODRA introduces schema definition language that allows to define database schema. The internal representation of the schema is stored in meta-base and use to type-check queries and programs. Schema language allows to define modules that are database units storing other database elements. Module stores data as well as meta-data for them. Elements (i.e. objects, according to object relativism principle) that can be stored inside modules includes: data objects, classes, procedures, views definitions, indexes, database links, etc. From the point of view of the programmer the process of module data definition involves the definition of object types, classes, procedures and class methods. Important part of this process is the definition of data objects. In ODRA this definition is equivalent to definition of variable (known from programming languages). The definition includes variable name, type and cardinality (specified in square brackets). The variable name defines the name for the objects, the type defines type of the objects, the cardinality of the variable defines the minimal and maximal number of objects that can appear in place where the declaration was introduced. Comparing it to relational databases, ODRA variable can be seen as counterpart of a table declaration. Below the syntax of sample module definition, including classes and variables, is presented:

```
module sample {

// classes definitions
class PersonClass {
   instance Person {
      name: string;
   }
}
class EmpClass extends PersonClass{
    instance Emp{
       deptName: string;
       salary: real;
       worksIn: ref Dept;
    }
    riseSalary(increaseValue:real) {
        self.salary := self.salary + increaseValue;
    }
}

 class DeptClass {
    instance Dept {
       name: string;
       location: string;
    }
}

// variables declarations
Emp: EmpType [0..*];
Dept: DeptType [0..*];
}
```

Above schema of module sample contains definition of three classes and declaration of two module level variables.

4.2 First Approach to Views Implementation

One of the most challenging elements of the Updateable Object Views implementation in the ODRA system was the type control extension towards view definition and view invocation type-checking. First implementation of updateable object views in ODRA introduces definitions that, from the syntactic point of view introduced the view definition following the syntax proposed in [13]. It introduced seed of virtual objects generator in form of the procedure called virtual objects and four operators for redefinition of operations performed on virtual objects: *on_retrieve* – for dereference of virtual identifier, *on_update* for updating virtual objects, *on_delete* – for definition of the operations that should be performed in response to virtual object delete request and *on_new* - for creating virtual object operation. The lack of a given operator in the view definition means that this operation is forbidden for a virtual objects defined by the view. Finally, the definition of sub-views can be added (in the same manner). As an addition the original declaration, the possibility of declaring types was introduced. Below code shows the syntax of the first version of a view definition in ODRA with underlined elements introduced according to type system requirement.

```
view [managerial view name] {
  virtual objects name: type[cardinality] proc_body
//view operators definitions
  [on_retrieve:type proc_body ]
  [on_update(name:type) proc_body ]
  [on_delete proc_body ]
  [on_new(name:type) proc_body ]
// sub-views definitions
  [view ...]
}
```

As can be seen each of the view procedures (seed generator and operators definition) had to have the type declared separately and there was no control over the consistency of e.g. return type for *on_retrieve* operator defining procedure and the *on_update* parameter type. The biggest problem, that often leads to misunderstandings lays in the type of virtual objects procedure that, in fact, does not define the type of defined virtual object. Instead, it defines type of the virtual object seed. From the point of view of virtual object user, the seed is a transparent part of the view implementation. It is defined by a view programmer to transfer the required data to view operators procedures (for which is an implicit argument) and sub-views seed generators. The type of a virtual object itself was implicitly defined by the type of view operators what made it illegible. Additionally, for each operator programmer was able to declare different type. For example the type of result of dereference operation could be set as string and the type of an update operator could be defines as integer. Instead of homogenous (according to its type) virtual object the implementation allows for defining virtual objects that were able to change its types depending on the operation that was performing on them. Moreover the types of virtual objects (i.e. attributes of the main virtual object) defined by sub-views could also have nothing in common with the type of the external virtual object. In cannot be excluded that, in some situations, it could be an interesting feature (expression flexibility connected with type control), but the usability and readability of the solution was, to put it mildly, slightly uncomfortable. Nevertheless, it is worth mention that this version of "a heterogeneously typed virtual objects" was successfully implemented in the ODRA type control system.

4.3 Modification of Views Implementation

An initial implementation has revealed some drawbacks of the approach to view syntax and view type description. Thus the goals of next approach was assumed as follows:

- Introduce explicitly the virtual object declaration and make it similar to ODRA variable declaration. It should strengthen the impression concerning similarity between variable and virtual object declaration.
- Separate the type of a view seed as internal view .
- Introduce the control over the types of view operator procedures.
- Introduce the control over the types of virtual objects defined by sub-views.

New version of view definition achieves above goals by introducing the following syntax of a view definition:

```
view [managerial view name] {
  virtual name: type[cardinality];
  seed : type[cardinality] proc_body
//view operators definitions
  [on_retrieve proc_body ]
  [on_update proc_body ]
  [on_delete proc_body ]
  [on_new proc_body ]
// sub-views definitions
  [view ...]
}
```

The first part of a view definition is a declaration of a virtual object name and type. The declaration is similar to ordinal ODRA variable declaration preceded with keyword virtual. The procedure that defines the seeds of virtual objects is now separated and renamed to seed. The return type of the seed functional procedure defines the type of seeds. The cardinality (by default equals 1) informs about the possible number of seeds (and thus virtual objects). Finally definition of view operators does not include type declaration. It is implicitly assumed that the type is equal to the type of a virtual object. For on_update operator it is the type of return value. For operators on_update and on_new it is the type of the procedure argument (by default the name that can be used inside those procedure to access the argument is value). Fragment of a definition of the simple view that is based on the sample module schema from section 4.1 is presented below.

```
view RichEmpDef {
    virtual RichEmp:record {name:string;}[0..*];
    seed: record{ e:ref Emp [0..*]} {
      return Emp where salary > 10000 as e;
    }
    on_retrieve {return deref(e);}
    on_update {
      if(value.salary > 10000) {
          e := value;
      } else {
          Exception exc;
          exc.message :=
           "decreasing salary below 10000 is forbidden";
          throw exc;
      }

    // sample nested sub-view definition
    view nameDef {
      virtual name:string;
      seed : record{ n:ref Emp.name} {
         return e.name as n;
    }
    on_retrieve { return deref(n);}
    }
}
```

Above simple view defines virtual objects named *RichEmp* with single attribute name representing those employees that are rich (i.e. earns more than 10000). The definition allows for performing dereference and update operation on virtual object *RichEmp* and only dereference for its virtual attribute name.

4.4 Virtual Pointer Objects

In SBA and thus in ODRA that implements the approach objects are divided into three categories according to the type of value it stores: simple, complex and pointer. Up to now virtual objects can be perceived by the user as simple objects (defined by the view without nested sub-views) or complex objects (defined by the view with nested sub-views describing attributes). For completeness of the transparency the ODRA implementation introduces virtual entities that can be perceived as pointer objects.

A pointer object allows to navigate in an object graph. The unique property of SBQL is that the environment of a pointer object is represented by the binder named with the name of pointed object; thus, navigation through the pointer object requires typing the name of the target object. This property allows us to separate the reference to pointer itself and the reference to pointed object. For example if we assume that friend is a pointer sub-object of Person object the query:

```
(Person where name = "Kim").friend
```

returns the reference (bag of references) of pointer object named friend. Such reference can be the subject of imperative operations (e.g. updated, deleted). To return the references to objects pointed by the friend objects one must write:

```
(Person where name = “Kim”).friend.Person
```

To define a virtual pointer with analogous semantics the implementation of views in ODRA introduced a new operator called *on_navigate*. As other view operators, the operator is defined as a functional procedure. The constraint is that it must return a reference (or a virtual reference) of a “virtually” pointed object. As for the other operators theirs return type implicitly corresponds to the declared type of a virtual object. These two assumptions enforce that only those virtual objects that return reference to the other objects can possess on_navigate operator and be perceived as (virtual) pointers.

5 Updateable Object Views and the ODRA Optimizer

One of the most important modules of the ODRA system is the static optimizer that implements optimization methods based on modifications of the query syntactic tree (AST). The methods include factoring out independent and weakly dependent subqueries [17][6], rules based on the distributivity property [17], removing dead subqueries [17], methods based on indices [12], methods based on query caching [8].

The views implementation enforce the implementation of another static optimization module - view rewriter. If the virtual object is processed by the query the view rewriter module can rewrite the query into the semantically equivalent form where view invocations are replaced by the code contained inside the view procedures. The view invocation is performed in two cases. First case is the moment in the query processing where exists explicit call to virtual object name. Second case is connected with operations that are performed on the virtual identifiers. Implemented algorithm allows for rewriting of all view operators (if appropriate conditions are met). The rewrite procedure takes as input the syntax tree representation that is already type-checked to assure type correctness and to supplement the AST nodes information with type signatures. Next section describes implemented view rewrite algorithm.

5.1 View Rewrite Procedure

The rewrite algorithm consists of the three general phases: query analyze, query rewrite and query configuration.

To exemplify the rewrite algorithm consider the following simple query based on the view sample from section 4.3.

```
deref(RichEmp where name = “Smith”);
```

The query returns information about rich employee named Smith. It contains calls to two virtual objects *RichEmp* and its virtual attribute name. It also contains two dereference operators call. First one, explicit, concerns *RichEmp* virtual object. Second one, implicit, concerns virtual attribute name and is enforced by the comparison operator. The rewrite algorithm implemented in ODRA is able to rewrite the query to the following form:

```
deref ((Emp where salary > 10000 as e) where ((e.name
as n).deref(n)) = “Smith”);
```

This form is then processed by the auxiliary names remover optimization module that is able to remove auxiliary names that are not required (‘e’ and ‘n’ in the above example):

```
deref ((Emp where salary > 10000) where (deref(name) =
“Smith”));
```

The final form can be a subject to another optimizations like factoring out independent sub-queries and/or removing dead sub-queries.

Query Analyze Phase

The goal of the query analyze phase is to analyze the AST of a subject query and to collect all the data required for view rewrite. The phase consists of the following steps:

1. For each AST name nodes that result signatures represent virtual identifiers and the name is not an auxiliary name:
   Save the mode together with the replacement AST taken from seed procedure contained inside the owner view definition (for potential replacement). If the procedure is not possible to rewrite[1] - mark the node as non-rewritable.
2. For each AST node representing operators (dereference, update, create, delete) if the operand sub-node result signature represents virtual identifier.
   Save the operator node together with AST taken from appropriate operator procedure. If the procedure is not possible to rewrite[2] - mark the node as non-rewritable.
3. If there was no node collected the rewrite procedure ended.
4. If at least one node was collected - For each collected name node:
   Determine the dependency: name node representing view invocation <–> name node representing sub-view invocation.
   In result the independent view invocation chains are created reflecting view aggregation in the context of a processed query.
5. For each collected operator node:
   Determine the dependency name node representing view invocation <–> operator node representing operation on the corresponding virtual object.
6. For each collected name node check if all dependent elements determined in step 4 and step 5 can be rewritten.
7. If yes start the rewrite procedure, else: end rewrite process.

[1] Current algorithm implementation constraint the rewrite only to the single expression/query procedure bodies (or the single return statement). If operator or seed implementation procedure has more than one expression/query, rewrite of the view invocation is not possible.

[2] Current algorithm implementation constraint the rewrite only to the single expression/query procedure bodies (or the single return statement). If operator or seed implementation procedure has more than one expression/query, rewrite of the view invocation is not possible.

Query Rewrite Phase

The goal of the query rewrite phase is to rewrite the AST by replacing view related nodes with view definition queries. Query rewrite execution consists of the following steps:

For each view invocation chains repeat steps 1-4:

1. Set first view in the view invocation chain as current view.
2. Rewrite name nodes that invoke the current view. Rewrite procedure causes the name node being replaced with the query from the current view seed procedure.
3. Rewrite operator nodes concerning current view. Rewrite procedure is similar for three operators: dereference, update and delete. Operator node representation is replaced with a dot (navigation) non-algebraic operator. Inserted operator left sub-node is set to the operand node of the operator node being replaced. And its right sub-node is set to the query taken from appropriate view operator procedure (one depending on the operator type (one of: *on_retrieve*, *on_update*, *on_delete*). Create operator requires different approach and has to be rewritten separately.
4. Set next view in the chain as current view. Repeat steps 2-4.
5. Rewrite each create operator. This operator procedure differs from described procedure in step 3 and has to be performed separately. This is due to the fact that semantically, *on_new* procedure is not called in the context of virtual identifier. Its role is to transform some query result into a virtual object. Thus the result of create operator is pass to a *on_new* operator procedure as an argument. Rewrite procedure simulates this behavior. The create operator node is replaced with the query from *on_new* procedure joined with create operator node sub-tree to calculate argument value.
6. Algorithm proceeds to phase: query configuration.

Query Configuration Phase

The query configuration phase passes the rewritten query through the type control module to regenerate type signatures. Finally, the rewrite procedure is restarted from the query analyze phase. The algorithm is restarted because it is possible that after the rewrite new view invocation appears. According to transparency of views there is constraint on using calls to virtual objects inside the view definition[3].

6 Future Development Ideas

Currently ODRA prototype is the subject to architecture redesign process for purpose of complete redefinition of its internal structure. The work is based on the existing implementation experience. Future system versions are going to be more stable and

[3] Problematic situation appears when a view implicitly make recursive calls on itself. This situation has to be detected and the rewrite has to be stopped. This case needs further research in the more general context of recursive views.

structuralized. It is also planned to equip it with many features that are inherent elements of commercial database system, and which were not introduced into the prototype due to less scientific significance.

However, as a scientific project, ODRA will be still used to develop new interesting features. In the area of updateable views it is planned to research the dependences between object classes and object views and for example allow the view to inherits from the class. There is also a need to do more research in the field of views defining virtual pointers. Current researches are also focused on using updateable views as tools for web application implementation. Such views could serve as RESTfull [9] web services interfaces to data stored in the database or be a key elements in framework that allow for scaffolding data with the web user interface.

7 Summary

The paper focuses on chosen issues and features of the implementation of Updateable Object Views in the ODRA system. Updateable views become not only a feature for ODRA users, but also determine the existence of many system features that could not be realized without their existence. This refers especially to the implementation of the generic wrapper to relational databases [18]. Relational data are available through (automatically generated) views. Internally the views refers to wrapper managed structures used to replace the SBQL query` fragments with wrapper execution of SQL queries.

ODRA views facility also supports the process of data integration into the virtual repository [12]. Thanks to an update capability they can map local data onto some global representation and, inversely, map global updates onto local operations.

Finally, the prototype implementation opens many new research fields in the context of views semantics and views usability.

Acknowledgments. This research work is funded from the Polish Ministry of Science and Higher Education finances in years 2010-2012 as a research project nr N N516 423438.

References

1. Adamus, R., Habela, P., Kaczmarski, K., Lentner, M., Stencel, K., Subieta, K.: Stack-Based Architecture and Stack-Based Query Language. In: Proceedings of the First International Conference On Object Databases ICOODB, Berlin, Tribun EU, pp. 77–96 (2008)
2. Adamus, R., Kaczmarski, K., Stencel, K., Subieta, K.: SBQL Object Views - Unlimited Mapping and Updatability. In: Proceedings of the First International Conference On Object Databases ICOODB, Berlin, Tribun EU, pp. 119–141 (2008)
3. Adamus, R., Kowalski, T.M., Subieta, K., Wiślicki, J., Stencel, K., Letner, M., Habela, P., Daczkowski, M., Pieciukiewicz, T., Trzaska, M., Wardziak, T.: Overview of the Project ODRA. In: Proceedings of the First International Conference On Object Databases ICOODB, Berlin, Tribun EU, pp. 179–197 (2008)

4. Bancilhon, F., Spyratos, N.: Update Semantics of Relational Views. ACM Trans. Database Syst. 6(4), 557–575 (1981)
5. Barbosa, D.M.J., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.: Matching Lenses: Alignment and View Update. In: ACM SIGPLAN International Conference on Functional Programming (ICFP), pp. 193–204. ACM, USA (2010)
6. Bleja, M., Kowalski, T.M., Adamus, R., Subieta, K.: Optimization of Object-Oriented Queries Involving Weakly Dependent Subqueries. In: Norrie, M.C., Grossniklaus, M. (eds.) ICOODB 2009. LNCS, vol. 5936, pp. 77–94. Springer, Heidelberg (2010)
7. Bohannon, A., Vaughan, J.A., Pierce, B.C.: Relational Lenses: A Language for Updateable Views. In: Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 338–347. ACM, USA (2006)
8. Cybula, P., Subieta, K.: Query Optimization through Cached Queries for Object-Oriented Query Language SBQL. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 308–320. Springer, Heidelberg (2010)
9. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis University of California at Irvine 2000 (2000), http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
10. Habela, P., Kaczmarski, K., Kozankiewicz, H., Subieta, K.: Modeling Data Integration with Updateable Object Views. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 188–198. Springer, Heidelberg (2005)
11. Kotidis, Y., Srivastava, D., Velegrakis, Y.: Updates Through Views: A New Hope. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006. IEEE Computer Society (2006)
12. Kowalski, T., Wiślicki, J., Kuliberda, K., Adamus, R., Subieta, K.: Optimization by Indices in ODRA. In: Proceedings of the First International Conference On Object Databases ICOODB, Berlin, Tribun EU, pp. 97–117 (2008)
13. Kozankiewicz, H.: Updateable Object Views. PhD Thesis, Institute of Computer Science, Polish Academy of Sciences (2005), http://www.sbql.pl/phds
14. Kuno, H.A., Rundensteiner, E.A.: Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views. In: Proc of the IEEE Int. Conf. on Data Engineering (ICDE-12), pp. 310–317. IEEE Computer Society (1996)
15. Lechtenbörger, J., Vossen, G.: On the Computation of Relational View Complements. In: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 142–149. ACM, USA (2002)
16. Lentner, M., Stencel, K., Subieta, K.: Semi-Strong Static Type Checking of Object-Oriented Query Languages. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 399–408. Springer, Heidelberg (2006)
17. Płodzień, J.: Optimization Methods in Object Query Languages. PhD Thesis, Institute of Computer Science, Polish Academy of Sciences (2001), http://www.sbql.pl/phds
18. Wiślicki, J.: An object-oriented wrapper to relational databases with query optimization, PhD Thesis, Computer Engineering Department, Technical University of Lodz (2008), http://www.sbql.pl/phds
19. Wang, L., Jiang, M., Rundensteiner, E.A., Mani, M.: An Optimized Two-Step Solution for Updating XML Views. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) DASFAA 2008. LNCS, vol. 4947, pp. 19–34. Springer, Heidelberg (2008)

Domain Expert Centered Ontology Reuse for Conceptual Models

Christian Kop

Alpen-Adria-Universitaet Klagenfurt
Unversitaetsstrasse 65 – 67, Klagenfurt
christian.kop@aau.at

Abstract. Generating a project specific domain model for the database of an information system is a critical task. The notions and structure must be well chosen since the application relies on that. With the Semantic Web, many notions and structures of certain domains already exist as a shared knowledge. Hence, it seems that the designer of an information system just has to take an ontology and reuse it for the information system. However, this strategy must be applied carefully since an ontology might have a different focus and purpose with respect to the intended conceptual database model. Nevertheless, it is worth to examine an ontology for elements which can be reused. Research results already exist for this question but these research approaches focus on ontology engineers or conceptual modelers who are able to analyze the ontology. Although these persons are necessary, they are not the only ones who usually must be involved in such a process! What about the domain experts, who do not have the skills to read and understand an ontology? They must be also involved in a good process. The approach presented here, focuses on the domain experts. It will be described how domain experts can be supported and involved in the process. With respect to the ontology language, this approach focuses on OWL.

1 Introduction

Generating an application specific domain model (e.g., the conceptual database model of an information system) is a critical task. The notions of the domain must be well chosen and related, such that the information system can fulfill the needs of the end users. There are several techniques to gather notions and functional requirements. Among these techniques, document analysis can be used, if the designer is aware of its risks (i.e., parts of the document might be out of date; it was generated for another purpose). If such documents are found, then parts of these documents can be used for the conceptual database model. With the Semantic Web, these documents are easily available. Many ontologies are already built and can be reused. Especially, OWL has become a well known language to specify ontologies. Therefore, OWL is used as the language for ontologies that are taken for reuse.

The literature focusing on the reuse of ontologies shows that the selection step is the critical and important step. It influences later steps of reuse. The selection step is the phase, where the needed concepts for the target model are chosen carefully from the original source ontology.

All these approaches also have in common, that a computer scientist, which has at least skills in conceptual modeling is necessary in the selection process. However, a domain model should not be based only on the skills of a computer scientist. What about the "domain experts" (e.g., the users of the information system)? How can they be involved in this process if they do not understand more formal languages of computer scientists? Of course, a computer scientist can try to translate and explain everything to the domain expert – but this would not be an efficient communication process.

Therefore, the approach, explained here, focuses on a more domain expert centered strategy for reusing concepts of an ontology. It is novel since it uses and combines different techniques of natural language processing and adopts them for ontology reuse. For instance, instead of reading the OWL specifications, the approach verbalizes important ontology elements (i.e., OWL classes, object properties, and some restriction patterns). Verbalization itself is a technique, which translates a formal ontology description to a controlled natural language. This is a language with a restricted grammar and/or a restricted lexicon to avoid ambiguities [47]. The controlled natural language is then the basis and input for the mapping process to a conceptual database model (see Figure 1). There are two advantages for doing this:

- A domain expert with no skills in ontology languages and conceptual modeling can understand the verbalized parts of the ontology.
- Once the candidates for reusable elements are known according to their natural language representation, the mapping to the conceptual database model is based on well known transformation guidelines from controlled natural language to a conceptual model.

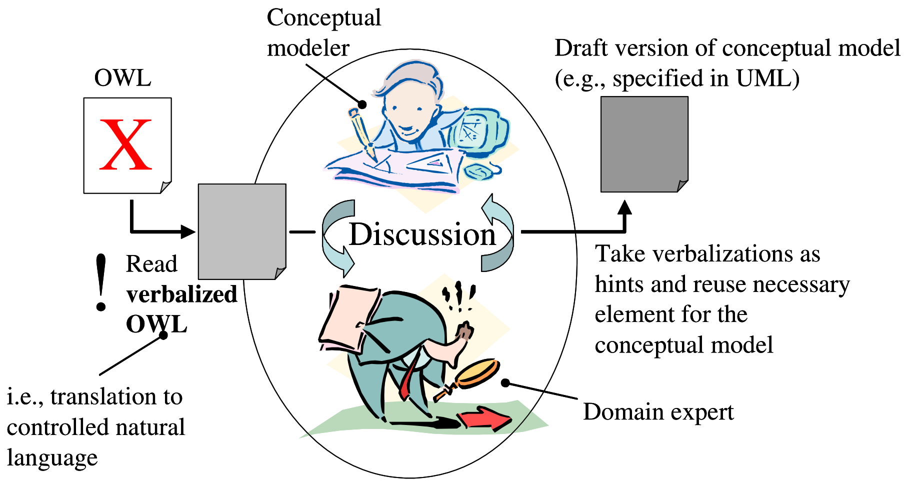

Conceptual modeler
OWL
X
Draft version of conceptual model (e.g., specified in UML)
Discussion
!
Read **verbalized OWL**
Take verbalizations as hints and reuse necessary element for the conceptual model
i.e., translation to controlled natural language
Domain expert

Fig. 1. Strategy of ontology reuse

Finally, the aim of the presented approach is also a computer supported selection mechanism of ontology elements, which considers a domain expert readable format too. Here, the term "computer supported" means, that a tool should provide proposals but the final decision making is done by the human stakeholders. Since a conceptual database model itself has the aim to be a basis for data retrieval and manipulation, natural language queries are chosen as the computer supported selection mechanism.

Thus, the paper is structured as follows: In the next section the related work is presented. Section 3 explains verbalization strategies in the context of ontology reuse and shows how the verbalized ontology elements can be transformed into a conceptual database model. Section 4 describes the selection mechanism. Section 5 describes the tests for verbalization and natural language query processing. Furthermore, an overview of the prototype is given. Section 6 summarizes the approach and gives an outlook to future work.

2 Related Work

2.1 OWL Ontologies versus Conceptual Models

OWL ontologies differ from conceptual database models since they have another purpose. OWL ontologies aim at providing knowledge that can be shared. From facts and data specified in the ontology, new facts can be derived using reasoning. Conceptual database models in the field of information systems on the other hand are mainly built to describe how data is structured and related for a certain application.

Due to their different aims, the content of an ontology can differ from the content of a conceptual database model. For instance, concepts that are modeled as classes in an OWL ontology might be modeled as the value of a category in a conceptual database model or they might be even ignored. It is also possible that individuals of an OWL ontology are modeled as classes (entity types) in a conceptual database model.

Beside the differences in the purpose and content, ontologies languages differ from a conceptual modeling language also in their structure. Baclawski et al. [32] and Hart et al. [37] describe structural differences between the conceptual modeling language UML and the ontology language OWL. These differences are:

- Different modeling elements in OWL,
- Independence of object properties,
- Different modeling of cardinalities.

OWL has many modeling elements, which cannot be found in conceptual modeling languages (e.g. intersectionOf, unionOf, complementOf, hasValue, allValuesFrom etc.). For these elements Brockmans et al. [39] and Gasevic et al. [38] use UML with a profile and stereotypes to visualize them. Since the aim of such a resulting UML diagram is on visualization of the ontology, this diagram cannot be used as a conceptual database model.

In OWL, object properties are independent modeling elements, which are not bounded to specific classes. If no domain and range are given, then "Thing" is implicitly the domain and/or the range. This is different to the modeling language concepts "association" in UML or "relationship type" in entity relationship diagrams. They are always bounded to UML classes or entity types respectively.

The cardinality specification in OWL restricts the minimum and/or maximum cardinality of the object property from the domain (where it is specified) to the range, but not vice versa. It is also possible, that a cardinality restriction for a class and an object property is specified, though the object property does not have concrete involved classes (independent object property). Furthermore, cardinality specifications are separated from the core object property definitions. They only reference the object property definitions. In conceptual modeling languages, cardinalities must be defined on both sides of an association and relationship type respectively. They are also important parts of the association or relationship type.

2.2 Usage of Ontologies for Information Systems

Despite the differences, many authors agree to use the shared knowledge of an ontology for application specific purposes. Guarino [36] highlighted the impact of ontologies for databases as well as for user interfaces and the application components of information systems. The research results presented by Sugumaran et al. [33], [34] showed, that a domain ontology can be used for conceptual model generation as well as for conceptual model validation. For the generation of a conceptual model, five steps are proposed, namely: identification of initial terms, mapping to domain terms, consistency checking, ER model generation and relational model generation. An ontology is involved in the second step. The terms derived from natural language requirements are checked with an ontology and if necessary the terms are expanded (i.e. new terms are derived).

In another approach rules for automatic transformations from an ontology to a conceptual model are proposed [35]. There, ontology elements are either transformed into conceptual model elements (e.g. classes, associations, attributes) or transformed into constraints.

Conesa et al. [40] proposed to use a large ontology for application specific purposes by firstly refining the ontology. Afterwards the needed elements of an ontology are selected. The ontology is pruned. Finally the resulting ontology or conceptual model must be refactored and refined. The selection of ontology elements is seen as an important step. This selection can be done with the support of specific requirements profiles for the target model [41]. Alani et al, [42] proposes to use RDQL queries.

A more general approach providing linguistically motivated guidelines to extract domain concepts and relationships from social tagged web sites is proposed in [43].

The above research shows, that there is a need to reuse ontologies. The process of selecting and reusing OWL ontology elements is currently managed mainly by ontology engineers or conceptual modelers. In fact, they should not be left alone. There must be a feedback and discussion process with the domain experts. Since domain experts normally do not understand OWL, the content must be translated to a human readable format. Also the very important selection process should be supported with a more human centered mechanism. Natural language queries, verbalization and the transformation of verbalized content to a conceptual model can help. In the next sub sections related work on these topics will be discussed.

2.3 Natural Language Queries

Ideas, techniques and problems of natural language querying are described in [1], [4], [12], [15], [16], [13], [14]. Some operate on a relational database model, others on a knowledge base or on a conceptual model. Some use additional information derived from linguistic lexicons or ontologies. In addition machine learning approaches are used [17], [19], [20], [21]. The main objective of all of these approaches is to support the retrieval of data or the generation of a more formal query language (e.g. SQL), which can then be executed on the data or knowledge base. This also holds for visual query tools described in [2], [10], [11] and form based query languages [5], [18]. Visual query languages produce SQL statements by navigating through the conceptual model.

In this paper, natural language queries are not used to retrieve data or to generate SQL statements. Instead, natural language queries are used as functional requirements for the database of an information system under development. In this role, they are applied on ontologies in order to find and select reusable OWL elements. Particularly, from the natural language queries, query notions are extracted that can be compared with ontology elements. According to the results, needed elements can be selected.

2.4 Verbalization

Strategies to verbalize ontologies are described in Fuchs et al. [8] and Hewlett et al.[9]. Information of individuals is verbalized in the research work of Bontcheva [3]. The verbalization of OWL keywords (subClassOf, intersectionOf, unionOf, hasValue, cardinality restrictions etc.) works very well. The problems of verbalization are the names (i.e., IRIs) of OWL classes and object properties. There is much freedom in the naming conventions. Therefore in [6] linguistically driven OWL class and object property naming guidelines are presented.

The verbalization approach presented here is a refinement of the work in [6]. It concentrates on a good verbalization of the class and object property names and represents them in a way which is understandable for the domain expert and supportive for the mapping to the conceptual model. Due to this special purpose, it does not try to exactly represent all the logical specifications in an OWL ontology like other verbalization approaches. Instead, it focuses on classes, object properties and some restriction patterns from which reusable information for conceptual database models can be derived.

2.5 Mapping from Controlled Natural Language to a Conceptual Model

The idea behind mapping of natural language to conceptual modeling elements was introduced in the 80's of the last century. In the beginnings, researchers gave guidelines how to manually

- Derive entity relationship diagram patterns from structured sentences [23]
- Derive program specification patterns from the appearance of certain word categories or combination of word categories in texts [28]
- Derive fact based conceptual model patterns from exemplary sentences [31]

For a better understanding of his Entity Relationship (ER) approach, Chen [23] proposed 11 heuristics to map sentences to an equivalent ER model. He used 11 structured English sentences and showed on the basis of their structure how each of the sentences and their elements can be mapped to elements of an ER model (i.e., entity types, relationship types or attributes). Abbott proposed heuristics for object oriented programming. Nijssen et al. [31] introduced a fact oriented modeling approach since this is much closer to natural language descriptions.

Other research results (e.g., [7], [22], [24], [25], [26], [27], [29], [30]) continued and refined these initial results. Moreno et al. showed similarities between conceptual models and controlled natural language sentences using formal instruments. Tjoa et al. [26] and Buchholz et al. [22] provided automatic transformation from controlled language sentences using parsers.

To summarize, once a controlled natural language sentence is available, it can be transformed to conceptual model elements.

2.6 Summary of Related Work

The related work shows, ontologies can be used as a basis for designing conceptual models if the differences of the ontologies and conceptual models in the scope and structure are considered. Therefore the approach described here, does not have the aim to automatically derive conceptual models. Instead, a feedback and discussion process must be established where the domain expert is strongly involved. The process of selecting ontology elements must be transparent to the domain expert. Particularly, the process consists of three domain expert centered steps:

- Selection step based on natural language queries, which are treated as explicit requirements statements for the database model under development,
- Verbalization step, which presents necessary parts of an ontology in a domain expert friendlier way,
- Transformation step, which helps to map the selected and verbalized ontology statements into conceptual modeling elements in a concise way.

With these steps, techniques and approaches are combined, which where used independently and for other reasons in previous work (i.e., natural language querying, verbalization, natural language analysis for conceptual modeling). These approaches and techniques are the basis for a domain expert centered reuse strategy as outlined in the Introduction Section (see Figure 1). Hence, reusing ontologies more becomes a communication task between domain experts and conceptual modelers.

In the succeeding Sections 3 and 4, the above mentioned steps will be described in more detail.

3 Verbalization and Transformation

For a better understanding of the output of the selection step this section describes this output and how it can be mapped to a conceptual model. Hence, the verbalization of ontology elements and their transformation to conceptual model elements are explained.

3.1 Verbalization

As mentioned in Section 2.4 the possible name variations of OWL classes and object properties can cause problems for verbalization. In order to produce better results and give ontology designers more freedom of defining the names, the following must be done:

(1) Separation of artificially connected terms
(2) More detailed verbalization of OWL object properties
(3) Adding extra information to complement verbalization

In the first step (1) different separation strategies are detected (e.g., Vegetarian_Pizza versus VegetarianPizza, has_Topping versus hasTopping etc.). The term is then transformed to a more natural and "normalized form" (e.g., vegetarian pizza). This step is also important for the second step. Instead of examining one name, different words can now be examined independently.

Whereas OWL classes can be already verbalized by the fist step, more has to be done for object properties. The example "hasTopping" gives the reason. In a natural language sentence, noun phrases are related with a verb. In an ontology specification you cannot always follow this natural way of expressing relationships. Most often, the name of an object property is a combination of a verb with a noun ("verb/noun"). In some cases a combination "noun/verb/noun" was found. If the nouns represent the domain and range of an object property, then they must be reduced since they are redundant. Sometimes, only a noun appears as a descriptor for an object property. If there are some additional patterns from which it can be concluded that the name of the object property is only a noun, then this can be transformed into a natural language sentence with a main verb. Often, in a verb/noun combination, the noun is neither the domain nor it is the range. However, in the ontologies, which were studied, it turned out, that the noun specified in the object property can be treated as a subset of the range or it is at least the role, which the range plays in the context of this object property. The second step gives some support in these cases.

The aim of the third step is to add extra information to make the resulting sentence of the verbalization more natural. An article is added to the involved nouns. In some cases, it turned out that an OWL range class is not a noun but an adjective. Of course, in such a case no article is added. In the cases, where the noun named in the object property is a subset of the range or the role of the range class within the object property, then the phrase "which is“ is used to combine the information given with the domain class, the object property and the range.

Since cardinality restriction and value restriction (e.g., hasValue, someValuesFrom, allValuesFrom) references to OWL object properties, the verbalization can be applied also on these restrictions. In one ontology, allValuesFrom were used to define a mapping between concepts and simple data types (e.g., OWL class “person” with an allValuesFrom-Restriction on the object property “first name” with the OWL class “string”). In this case the third step adds a “has” before the noun “first name”. The phrase “with data type” denotes that a data type is used. In more detail, let

N … be a descriptor for a noun or a compound noun or adjective + noun or adjective + compound noun
A … be an adjective only
V … be the verbal part (verb or verb + preposition)
c … be a numerical constant >= 0 (i.e., 0, 1, 2, … k)
D … be a simple data type (e.g., String)
N_1, N_2, N_3 … be a N at position 1, 2 or 3 within the resulting sentence pattern

then, depending on the underlying information stored in the OWL model, the following controlled natural language sentence patterns can be generated:

An OWL class can be transformed to the pattern N or A.

An OWL object property can be transformed to one of the following sentence patterns[1]:

- N_1 V <u>a</u> N_2
- <u>Anything</u> V <u>Anything</u>
- N_1 V N_2 <u>which is a</u> N_3
- <u>Anything</u> V N_2 <u>which is</u> <u>Anything</u>

An OWL subclassOf definition can be transformed to

- N_1 <u>is a</u> N_2

Cardinality restrictions can be transformed to one of the following sentence patterns

- N_1 V <u>exactly</u> c N_2
- N_1 V <u>at least</u> c N_2
- N_1 V <u>maximal</u> c N_2
- N_1 V <u>between</u> c_{min} <u>and</u> c_{max} N_2

A value restriction (hasValue, allValuesFrom, someValuesFrom) can be transformed to one of the following reduced sentence patterns.

- N_1 V N_2
- N_1 V N_2 <u>which is a</u> N_3
- N_1 V N_2 <u>which is</u> A
- N_1 V N_2 <u>with data type</u> D

[1] Underlined words are keywords. For instance, <u>a</u> represents the indefinite article “a”, “an” respectively. <u>Anything</u> stand fort he ontology root class „Thing“. The pattern <u>Anything</u> V <u>Anything</u> is derived if the object property is indipendent from a concrete OWL class.

3.2 Transformation and Reuse

Once the classes, object properties, value and cardinality restrictions are verbalized, the stakeholders have a set of natural language sentence patterns. Domain experts can be involved. They better get hints what the ontology describes. Another advantage is: Transformation heuristics, which are transparent to both, domain experts and designers can be applied to transform these sentence patterns to a conceptual model. Particularly, nouns, which were only implicitly mentioned within an object property (e.g., hasDrink) and which were not explicitly mentioned as OWL classes can now also be treated as conceptual model classes (e.g., UML classes).

The following table presents the transformations from sentence patterns to conceptual model patterns. Sentence patterns with "Anything" resulting from independent object properties do not appear in this table. These patterns must be refined. A computer supported search for more concrete restrictions based on these independent object properties can be executed. The conceptual modeling pattern of course must be integrated to an already existing work in progress conceptual database model. In cases, where one sentence pattern can lead to more than one conceptual modeling pattern, a decision from the stakeholders is necessary.

Table 1. Transformation from sentence patterns to conceptual model patterns

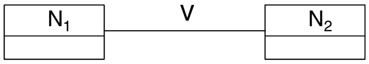
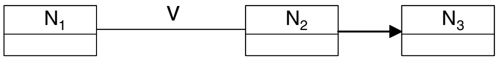
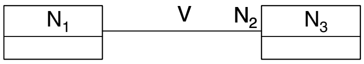
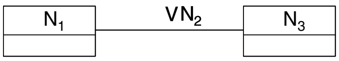
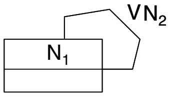
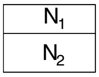
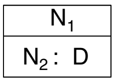
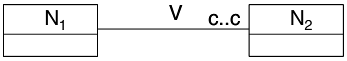
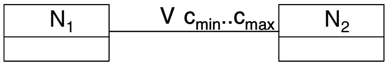
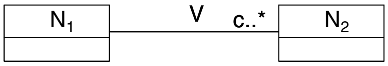
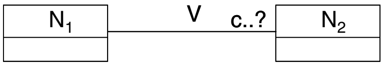
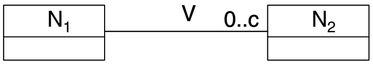
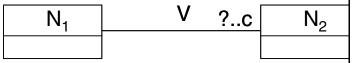
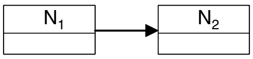
	Sentence pattern	Conceptual model pattern
1	N_1 V a N_2	
2	N_1 V N_2 which is a N_3 (N_1 <> N_3)	
		
		
3	N_1 V N_2 which is a N_3 (N_1 = N_3)	
4	N_1 V N_3 which is A	
5	N_1 V N_2 with data type D	

Table.1. (*Continued*)

6	N_1 V exactly c N_2	
7	N_1 V between c_{min} and c_{max} N_2	
8	N_1 V at least c N_2	
		
9	N_1 V maximal c N_2	
		
10	N_1 is a N_2	

4 Selection

Besides browsing the OWL class list in the ontology, classes can also be selected by requirements. Natural language queries processing is proposed here as a helpful instrument. The strategy has the advantage that also domain experts with no knowledge of query languages can understand such queries.

4.1 Linguistic Instruments for Natural Language Queries

In order to achieve this, natural language query processing is build on the following steps:

- Tagging
- Chunking

For the linguistic analysis the Stanford Tagger [44] is used. A tagger is a tool, which takes as input a text and returns a list of sentences with tagged (categorized) words (i.e., words categorized as noun, verb, adjective etc.). The chosen tagger categorizes the words according to the Penn-Treebank tagset [45]. In this tagset the word categories together with some important syntactical features of a word are encoded. For instance, if a noun is in plural then the category NNS is chosen. If a proper noun is detected then NNP is used. The tag JJ is one possibility to define Adjectives. Verbs are annotated with tags like VB, VBZ. Also here the specific tag (e.g., VB, VBZ) depends on the special usage of the verb in the sentence.

On top of the tagging module the chunking module clusters a set of words that belong together to chunks. The chunker module clusters nouns (e.g., customer number) as well as word categories strongly related to nouns (e.g., articles, adjectives) to a noun phrase (e.g., the customer number). It subsumes verbs and word categories, which are strongly related to a verb (e.g., adverb, verb particle) to a verb phrase. For natural language query sentences, an exception was introduced. If “many” or “much” follows the word “how” (e.g., “how many persons”) then “how” and “many” are combined to one chunk. The chunking output is forwarded to the interpreter.

4.2 Selection of Classes and Relationships

The query interpreter extracts all the noun phrases from the query. These extracted query notions are compared with verbalized OWL classes and individuals of the ontology. If there is a match, then the OWL class is treated as a selected concept. An individual is traced back to the class to which it belongs. This class is then treated as a selected concept.

For a concept (OWL class) the relationships (object properties, value restrictions, cardinality restrictions) are presented in the verbalized form described in the previous section. In order to give the stake holders the freedom to select those relationships that are really relevant they are provided with relationships in which the selected concepts are directly involved or inherited from their super classes. From these presented and verbalized relationships, the stake holders can freely choose the needed relationships.

If all the possible relationships of a selected concept “c” are needed, then the following is presented:

- The verbalized concept “c’.
- All the verbalized object properties, in which c is involved.
- All the verbalized cardinality restrictions, in which c is involved.
- All the verbalized value restrictions (hasValue, someValuesFrom, allValuesFrom) in which c is involved.
- The set of super classes SC of c.

Furthermore, for each s $\in$ SC the following is presented

- All the verbalized object properties, in which the concept s is involved.
- All the verbalized cardinality restrictions in which the concept s is involved.
- All the verbalized value restrictions (hasValue, someValuesFrom, allValuesFrom) in which the concept s is involved.

In the selection process, the resulting conceptual model elements that are proposed by the system are close to their origins if several mapping strategies exist. For instance, for the three mapping strategies of the pattern “N_1 V N_2 which is a N_3”, the one is chosen where V and N_2 together names the association.

5 Tests and Tool

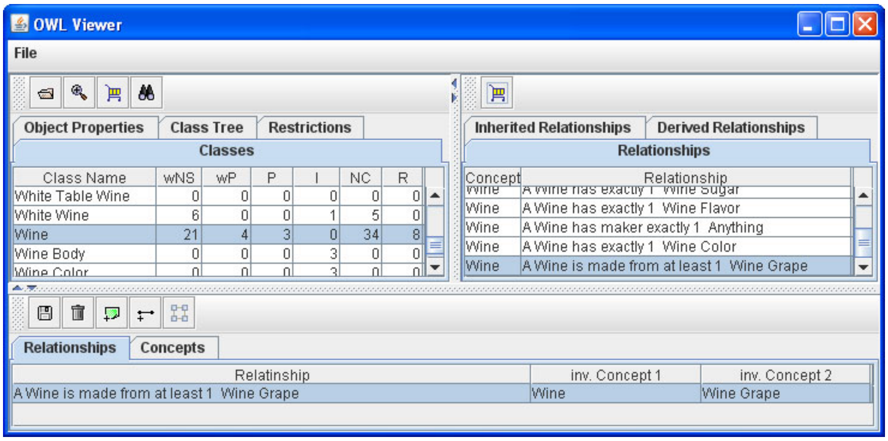
5.1 Tests of Tool Components

Some of the OWL documents from [48] and [49], specifying domain ontologies, were examined to get a good impression how ontologies are specified and which natural language pattern can be applied for verbalization of relationship candidates. There is a restriction for verbalization. This restriction appears if the object property name does not contain a verb but only a noun. If from the examination of the object property and its involved domain and range it cannot be concluded, that the noun mentioned in the object property is the role or the subset of the range then it is treated incorrectly as a verb.

For a better understanding which kind of database queries are generated, the natural language query processing component was tested with the Geo query corpus [46]. This corpus contains about 880 query statements. In addition, also queries found in literature were used as test cases. Examples for such queries are "which states have a major city named Austin" (from Geo Query corpus), "Tell me the order items that belong to Order 123". It is also possible to combine more than one sentence to one query.

5.2 Tool

The tool is implemented in Java. Figure 2 shows a screen shot of the tool.



Class Name	wNS	wP	P	I	NC	R
White Table Wine	0	0	0	0	0	0
White Wine	6	0	0	1	5	0
Wine	21	4	3	0	34	8
Wine Body	0	0	0	3	0	0
Wine Color	0	0	0	3	0	0

Concept	Relationship
Wine	A Wine has exactly 1 Wine Sugar
Wine	A Wine has exactly 1 Wine Flavor
Wine	A Wine has maker exactly 1 Anything
Wine	A Wine has exactly 1 Wine Color
Wine	A Wine is made from at least 1 Wine Grape

Relatinship	inv. Concept 1	inv. Concept 2
A Wine is made from at least 1 Wine Grape	Wine	Wine Grape

Fig. 2. Tool Screenshot

For reading and parsing OWL documents, the Jena API[2] is used.

The tool consists of three areas. In the left upper corner, the original ontology can be imported. The user can see the ontology with different views. One of these views is a

[2] Jena Semantic Web Framework Project: http://jena.sourceforge.net/

list of OWL classes together with some statistics (e.g., appearance in the subclass hierarchy, number of children, number of object properties, number of instances and number of restrictions). It is possible to sort the list according to one of these statistical columns. Doing this, the user gets a quick overview of the "structural importance" of a class. Hence, he can even make some first selections according to these sorted lists.

As an alternative, queries can be used as described in Section 4. Once, the needed classes are known, the possible relationship candidates of these classes can be presented in the upper right area after pressing the "magnifying glass" button. In the resulting controlled sentences for the relationships, involved concepts are firstly treated equally (i.e., no distinction between a class and attribute is made). There are three views in this area. The first view shows only the relationships, which can be derived from the object properties and restrictions in which the selected concept is directly involved. The second view shows the relationship candidates that can be inherited from super classes. In this view, instead of the super class the selected concept is now involved. As a reason for this step, suppose that in a conceptual model a super class might not be important. However, it is important to present the structure of the selected concept, even if it is inherited. The last view shows the "automatic proposal" of the tool. According to Section 4.2, the relationships are shown, in which the concept is directly involved. Afterwards, all the super classes of the concept are presented. Finally the super classes involved in their relationships (object properties and restrictions) are presented.

If the user finally decides to take the selected concepts and (some of) their relationships for his conceptual model, he can do it with the shopping cart button. The selected concept and relationships then are reused and also appear in the bottom area. The content of this "basket" can be stored into a XML file, which then can be imported into a conceptual modeling editor for further usage in a succeeding phase (i.e., refinement and integration of the chosen elements in order to generate and complete the conceptual database model).

6 Conclusion and Future Work

In order to reuse parts of ontologies for database models of information systems, the approach described in this paper focuses on a domain expert centered presentation and selection of OWL ontology elements. Verbalization is used as the presentation strategy. Natural language queries are used for initial selection of concepts. Candidates for relationships, which can be mapped to UML associations, were derived from verbalized object properties, value restrictions and cardinality restrictions.

In future, additional ontology elements will be examined for their usage in conceptual models. Up to now for example, a value restriction only makes sense for a conceptual model if the "range" of this restriction is a named OWL class. If it is a compound ontology concept (e.g., union of …), currently it is not considered for conceptual modeling. Furthermore, OWL datatype properties will be examined for the described ontology reuse approach. It is also planned to study if verbs found in queries can be used for matching with concepts in the ontology.

References

1. Berger, H., Dittenbach, M., Merkl, D.: Quering Tourism Information Systems in Natural Language. In: Godlevsky, M., Liddle, S., Mayr, H.C. (eds.) Informaton Systems Technology and its Applications – Proceedings of the 2nd Conference ISTA 2003. GI Lecture Notes in Informatics, vol. p-30, pp. 153–165. Koellen Verlag, Bonn (2003)
2. Bloesch, A.C., Halpin, T.A.: ConQuer: A Conceptual Query Language. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 121–133. Springer, Heidelberg (1996)
3. Bontcheva, K.: Generating Tailored Textual Summaries from Ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 531–545. Springer, Heidelberg (2005)
4. ter Hofstede, A.H.M., Proper, H.A., van der Weide, T.P.: Exploring Fact Verbalizations for Conceptual Query Formulation. In: van de Riet, R.P., Burg, J.F.M., van der Vos, A.J. (eds.) Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems, pp. 40–51. IOS Press, Amsterdam (1996)
5. Embley, D.W.: NFQL: The Natural Forms Query Language. ACM Transactions on Database Systems 14(2), 168–211 (1989)
6. Fliedl, G., Kop, C., Vöhringer, J.: From OWL Class and Property Labels to Human Un-Derstandable Natural Language. In: Kedad, Z., Lammari, N., Métais, E., Meziane, F., Rezgui, Y. (eds.) NLDB 2007. LNCS, vol. 4592, pp. 156–167. Springer, Heidelberg (2007)
7. Fliedl, G., Kop, C., Mayr, H.C., Salbrechter, A., Vöhringer, J., Weber, G., Winkler, C.: Deriving Static and Dynamic Concepts from Software Requirements using Sophisticated Tagging. Data and Knowledge Engineering 61(3), 433–448 (2007)
8. Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., Schneider, G.: Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In: Eisinger, N., Małuszyński, J. (eds.) Reasoning Web. LNCS, vol. 3564, pp. 213–250. Springer, Heidelberg (2005)
9. Hewlett, D., Kalyanpur, A., Kolovski, V., Halaschek-Wiener, C.: Effective Natural Language Paraphrasing of Ontologies on the Semantic Web. In: End User Semantic Web Interaction Workshop. CEUR-WS Proceedings, vol. 172 (2005), `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/`
10. Jaakkola, H., Thalheim, B.: Visual SQL – High-Quality ER-Based Query Treatment. In: Jeusfeld, M.A., Pastor, Ó. (eds.) ER Workshops 2003. LNCS, vol. 2814, pp. 129–139. Springer, Heidelberg (2003)
11. Järvelin, K., Niemi, T., Salminen, A.: The visual query language CQL for transitive and relational computation. In. Data & Knowledge Engineering 35, 39–51 (2000)
12. Kardkovács, Z.T.: On the Transformation of Sentences with Genitive Relations to SQL Queries. In: Montoyo, A., Muńoz, R., Métais, E. (eds.) NLDB 2005. LNCS, vol. 3513, pp. 10–20. Springer, Heidelberg (2005)
13. Owei, V., Rhee, H.-S., Navathe, S.: Natural Language Query Filtration in the Conceptual Query Language. In: Proceedings of the 30th Hawaii International Conference on System Science, vol. 3, pp. 539–550. IEEE Press, New York (1997)
14. Stratica, N., Kosseim, L., Desai, B.C.: Using semantic templates for a natural language interface to the CINDI virtual library. Data & Knowledge Engineering 55, 4–19 (2005)
15. Kapetainos, E., Baer, D., Groenewoud, P.: Simplifying syntactic and semantic parsing of NL-based queries in adavanced application domains. Data & Knowledge Engineering Journal 55, 38–58 (2005)

16. Kao, M., Cercone, N., Luk, W.-S.: Providing quality responses with natural language interfaces: thenull value problem. IEEE Transactions on Software Engineering 14(7), 959–984 (1988)
17. Tang, L.R., Mooney, R.J.: Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In: Proceedings of the 12th European Conference on Machine Learning (ECML 2001), pp. 466–477 (2001)
18. Terwillinger, J.F., Delcambre, L.M., Logan, J.: Querying through a user interface. Data Knowledge Engineering 63, 774–794 (2007)
19. Kate, R.J., Mooney, R.J.: Using String-Kernels for Learning Semantic Parsers. In: COLING/ACL Proceedings, Sydney, pp. 913–920 (2006)
20. Ge, R., Mooney, R.J.: A Statistical Semantic Parser that Integrates Syntax and Se-mantics. In: Proceedings of the Ninth Conference on Computational Natural Language Learning, Ann Arbor, MI, pp. 9–16 (2005)
21. Wong, Y.W., Mooney, R.J.: Learning for Semantic Parsing with Statistical Machine Translation. In: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2006), New York, pp. 439–446 (2006)
22. Buchholz, E., Cyriaks, H., Düsterhöft, A., Mehlan, H., Thalheim, B.: Applying a Natural Language Dialog Tool for Designing Databases. In: Proc. International Workshop on Applications of Natural Language to Databases (NLDB 1995), pp. 119–133 (1995)
23. Chen, P.P.: English Sentence Structure and Entity Relationship Diagrams. Int. Journal of Information Sciences 29, 127–149 (1983)
24. Moreno, A.M., Juristo, N., van de Riet, R.P.: Formal Justification in object-oriented modeling: A linguistic approach. Data & Knowledge Engineering 33, 25–47 (2000)
25. Rolland, C.: An Information System Methodology Supported by an Expert Design Tool. In: Pirow, P.C., Duffy, N.M., Ford, J.C. (eds.) Proceedings of the IFIP TC8 International Symposium on Information Systems, pp. 189–201. North Holland Publ. Company (1987)
26. Tjoa, A.M., Berger, L.: Transformation of Requirement Specification Expressed in Natural Language into an EER Model. In: Elmasri, R.A., Kouramajian, V., Thalheim, B. (eds.) ER 1993. LNCS, vol. 823, pp. 127–149. Springer, Heidelberg (1994)
27. Vadera, S., Meziane, V.: From English to Formal Specifications. The Computer Journal 37(9), 753–763 (1994)
28. Abbott, R.J.: Program Design by Informal English Descriptions. Communication of the ACM 26(11), 882–894 (1983)
29. Burg, J.F.M.: Linguistic Instruments in Requirements Engineering. IOS Press, Amsterdam (1997)
30. Dignum, F., Riet van de, R.P.: Knowledge base modeling based on linguistic and founded in logic. Data & Knowledge Engineering 7, 1–34 (1991)
31. Nijssen, G.M., Halpin, T.A.: Conceptual Schema and Relational Database Design – A fact oriented approach. Prentice Hall Publishing Company (1989)
32. Baclawski, K., Koka, M.K., Kogut, P.A., Hart, L., Smith, J., Holmes, W.S., Letkowski, J., Aronson, M.L., Emery, P.: Extending the Unified Modeling Language for Ontology Development. Journal of Software and System Modeling (sosym) 1(2), 142–156 (2002)
33. Sugumaran, V., Storey, V.: The Role of Domain Ontologies in Database Design: An Ontology Management and Conceptual Modeling Environment. ACM Transaction on Database Systems 31(3), 1064–1094 (2006)
34. Sugumaran, V., Storey, V.: Ontologies for conceptual modeling: their creation, use and management. Data & Knowledge Engineering 42, 251–271 (2002)

35. Vasilecas, O., Bugaite, D.: Ontology-Based Elicitation of Business Rules. In: Nilsson, A.G., Gustas, R., Wojtkowski, W.G., Wojtkowski, W., Wrycza, S., Zupancic, J. (eds.) Advances in Information Systems Development: Bridging the Gap between Academia & Industry, vol. 2, pp. 795–805. Springer, Heidelberg (2005)
36. Guarino, N.: Formal Ontology and Information Systems. In: Proceedings of FOIS 1998, pp. 3–15. IOS Pess, Amsterdam (1998)
37. Hart, L., Emery P., Colomb B., Raymond, K., Taraporewalla, S., Chang, D., Ye, Y., Kendall, E., Dutra, M.: OWL Full and UML 2.0 Compared, Version 2.4 (2004), `http://www.itee.uq.edu.au/~colomb/Papers/UML-OWLont04.03.01.pdf`
38. Gasevic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Converting UML to OWL Ontologies. In: ACM Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, pp. 488–489. ACM Press (2004)
39. Brockmans, S., Haase, P., Hitzler, P., Studer, R.: A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 303–316. Springer, Heidelberg (2006)
40. Conesa, J., Olivé, À.: Pruning Ontologies in the Development of Conceptual Schemas of Information Systems. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 122–135. Springer, Heidelberg (2004)
41. Bhatt, M., Rahayu, W., Prakash, S., Wouters, C.: OntoMove: A Knowledge Based Framework For Semantic Requirement Profiling and Resource Acquistion. In: Proceedings of the Australian Software Engineering Conference (ASWEC 2007), pp. 137–146 (2007)
42. Alani, H., Harris, S., O'Neill, B.: Ontology Winnowing: A Case Study on the AKT Reference Ontology. In: International CIMCA/IAWTIC Conference, pp. 710–715. IEEE Press (2005)
43. Sugumaran, V., Purao, S., Storey, V., Conesa, J.: On-Demand Extraction of Domain Concepts and Relationships from Social Tagging Websites. In: Hopfe, C.J., Rezgui, Y., Métais, E., Preece, A., Li, H. (eds.) NLDB 2010. LNCS, vol. 6177, pp. 224–232. Springer, Heidelberg (2010)
44. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: Feature rich part-of speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL, pp. 252–259 (2003)
45. Penn-Treebank TagSet, `http://www.cis.upenn.edu/~treebank` (last access: August 19, 2011)
46. Geo Query Project, `http://www.cs.utexas.edu/users/ml/geo.html` (last access: August 19, 2011)
47. Controlled Natural Language, `http://sites.google.com/site/controllednaturallanguage/` (last access: August 19, 2011)
48. `http://krono.act.uji.es/Links/ontologies` (last access: August 19, 2011)
49. Ontology Designpatterns.org (ODB), `http://www.ontologydesignpatterns.org/` (last access: August 19, 2011)

Semantic Invalidation of Annotations Due to Ontology Evolution

Julius Köpke and Johann Eder

Department of Informatics-Systems, University of Klagenfurt, Austria
firstname.lastname@aau.at
http://isys.uni-klu.ac.at

Abstract. Semantic annotations assign concepts of a reference ontology to artifacts like documents, web-pages, schemas, or web-services. When an ontology changes, these annotations have probably to be maintained as well. We present an approach for automatically checking whether an annotation is invalidated by a change in the reference ontology. The approach is based on annotation paths and on the explicit definition of change-dependencies between ontology artifacts. Ontology change-logs and change dependencies are then exploited to identify those annotation paths which require maintenance.

Keywords: Semantic annotation, ontology evolution, annotation maintenance, semantic invalidation, semantic dependencies.

1 Introduction

Semantic annotations were developed to assign semantics to various documents, e.g. XML-Schemas, XML documents, web-pages, or web-services by linking elements of these documents to elements of a reference ontology [20],[13]. When the reference ontology evolves, these annotations might have to be maintained as well. To ease the maintenance effort it is highly desirable to identify those annotations which have to be maintained in contrast to those which are invariant to the changes. So the goal of the research reported here[1] was to develop a technique to automatically identify those annotations which need attention as a consequence of a given set of ontology changes. The method should deliver all annotations where the annotations itself or instance data (might) need to be changed and it should return the smallest possible set of annotations (no false negatives and as little false positives as possible). Furthermore, the analysis of the necessity of maintenance should also deliver strategies for changing the annotation, if possible.

The proposal is based on annotation path expressions [14] as a method for semantic annotations. Annotations consists of paths of concepts and properties of the reference ontology. These annotation paths were developed to grasp the

[1] This work was partially supported by the Austrian Ministry of Science and Research; program GEN-AU, project GATIB.

semantics more precisely and to provide a pure declarative representation of the semantics opposed to the more procedural lifting and lowering scripts of traditional XML-Schema annotation [13] methods. These annotation paths can automatically be transformed to ontology concepts. The ontology concepts can then be used to create semantic matchings between annotated elements from different schemas or different versions of the same schema. An example for an annotation is */invoice/hasDeliveryAddress/Address/hasZipCode* which could be used to annotate a *zip-code* element in an XML-Schema for invoices. The example assumes that *invoice* and *address* are concepts of the reference ontology and *hasDeliveryAddress* is an object-property and *hasZipcode* is a datatype-property. Another motivation for the development of annotation paths was to provide a better basis for maintaining annotations when the reference ontology is changed. In [14] we have introduced different types of invalidations of the annotations when the ontology changes:

– **Structural invalidation:** An annotation path references concepts and properties of the reference ontology. All referenced concepts and properties must exist in the ontology and basic structural requirements must be met. For example */invoice/hasDeliveryAddress/Address/hasZipCode* gets invalid if the concept *Address* is removed from the reference ontology.
– **Semantic invalidation:** An annotation path is invalid if its semantic representation (an ontology concept) imposes contradictions to the reference ontology. For example */invoice/hasDeliveryAddress/Address/hasZipCode* gets semantically invalid if the domain of *hasDeliveryAddress* is changed to *order* and *invoice* is defined to be disjoint from *order*.

These types of invalidations can be tracked by structural checks and simple reasoning over the ontology representation of the annotation path. When an annotation got invalid it needs to be repaired. Thus, evolution-strategies [19] are required in order to maintain the annotations. For example, if a referenced concept is removed we may use the super-concept for the annotation instead. If it was renamed we need to rename the element in the annotation paths as well. In this paper we will focus on another kind of invalidation: Changes of the semantics of the annotations which lead to a misinterpretation of annotated data without invalidating the annotation structurally or semantically. We will describe this problem in the following section with an illustrative example. We will first present the notion of semantic changes and discuss if such changes to the semantics of an annotation can, still, be derived from the plain ontology in section 2 and describe and define the explicit dependency definitions in section 3 and 4. In section 5 we show how the explicit definitions can be used to track semantic changes. In section 6 we present a prototype-implementation of the approach. Section 7 gives an overview about the related work. The paper concludes in section 8.

2 Semantic Changes and Their Automatic Detection

Semantic changes are consequences of changes in the reference ontology which do not invalidate the annotation semantically or structurally, but might lead to misinterpretations. We illustrate such semantic changes with the following example:

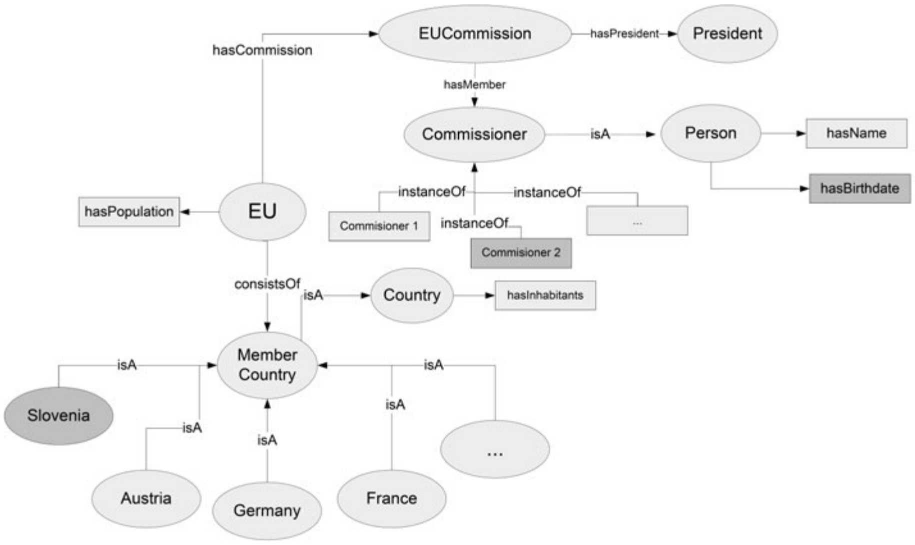


Fig. 1. Example Ontology

In figure 1 an example ontology[2] that represents parts of the European Union is depicted. We now assume that we have an annotation of an XML-Schema-element with the annotation path */EU/hasPopulation*. Some changes were made to the ontology: *Slovenia* was added as an additional *MemberCountry* and an additional *Commissioner*-instance was added and persons now have the property *hasBirthdate*. The differences between the old and the new ontology version are marked in dark-grey. Now obviously the semantics of an annotation */EU/hasPopulation* of some element in an XML Schema are changed because the parts of the European union were changed. This change does not influence the validity of the annotation itself - but the semantics of the annotation has changed. Documents that were annotated with the old ontology version will have a lower population number than documents of the current version. This imposes problems because it leads to the misinterpretation of the data. For example a human reader might come to the conclusion that the EU has a higher birthrate

[2] The member countries are modeled in form of concepts and specific commissioners as instances in order to show problems that may occur when concepts or instances evolve.

after the change. The goal of this paper is the automatic generation of warnings for such changes.

Since an ontology is used to express the semantics of a domain it should be possible to derive the changes of the semantics of annotations automatically. To avoid that each annotation has to be checked, if any ontology element has changed, we need to reduce the set of ontology elements which might invalidate a specific annotation. Ontology views [16] are methods to reduce the size of an ontology. Ontology module extraction [7] techniques can be applied for the same purpose. These methods generate a sub-ontology that only contains relevant elements for a given starting point (set of concepts). The starting point in our case is the semantic representation of an annotation path expression. When such sub-ontologies are created for the old and the new ontology version we can check, if there were changes between the sub-ontology versions. Since the sub-ontologies only contain elements that are relevant for the annotation in question we should be able to significantly reduce the number of false-positives.

Typical methods for the generation of sub-ontologies begin with a concept in question and then add more and more concepts that are related. The relation is expressed in form of sub/superclass relations or object-properties. We will illustrate the general idea of the generation of a sub-ontology with an example:

- The starting point is *EU/hasPopulation*
- This leads the inclusion of the concept *EU*
- The *consistsOf* property on *EU* requires the addition of *MemberCountry*
- *MemberCountry* requires the addition of its super-concept: *Country* with the property *hasInhabitants*
- *MemberCountry* requires the addition of *Slovenia*, *Austria*, *Germany*, ...

When we now compare the view of the old and the new ontology version we can figure out that *Slovenia* was added. We would throw a warning that the semantics of *EU/hasPopulation* was possibly changed. In this example we have assumed that we include all datatype-properties of a concept in the view and all concepts that are in the range of the object properties of the included concepts. In addition, all super- and sub-concepts as well as individuals of the included concepts are added. Unfortunately, such an algorithm would thus, include much more concepts:

- The *hasCommission* property of *EU* requires the addition of *EUCommission*
- The *hasPresident* property of *EUCommision* requires the addition of *President.*
- The *hasMember* property of *EUCommision* requires the addition of *Commissioner.*
- The *Commissioner* concept requires the addition of its instances.
- ....

At the end the entire ontology would be included in the view. Thus, all changes that happened between the old and the new ontology version are assumed to be relevant for *EU/hasPopulation.* This is certainly not true. In order to avoid this behavior the set of properties that are followed to build the view needs to be much smaller. Thus, strategies are required to choose the proper object-properties that should be followed. But where is the difference between *consistsOf* and *hasCommission*? From where can we know that if we want to build the view for *EU/hasPopulation* we need to follow the *consistsOf* property and that if we want to create the view for *EU/numberOfCommissioners* we need to follow the *hasCommision* property?

Thus, strategies are required in order to keep the view small and meaningful.

3 Requirements for Explicit Dependency-Definitions

As shown in the last section there is typically no knowledge about what may be invalidated semantically by changes since this is not an invalidation of the logical theory (which could be calculated) but a change of the semantics of the annotations. The reason for this is that the ontology does not fully specify the real-world domain. Therefore, a straight forward solution is the addition of the missing knowledge to the ontology. This means sentences like " *The population of the EU is changed when the MemberCountries change*" should be added to the ontology. Obviously this a very wide definition because we have not stated anything about the types of relevant changes. Is it changed when the name of a country changes or only if a specific attribute changes? In general which operations may invalidate our value? The examples of the population of the EU can be described as the aggregation of the population of the member countries. Thus, we need a way to describe such functions. These observations lead to the following requirements for change-dependency definitions:

1. The change-dependent concept or property must be described including the context. For example *hasPopulation* of *EU* and *hasPopulation* of *City* might depend on a different set of ontology elements.
2. The definition of the change-dependency should allow fine grained definitions of dependencies. For example it should be possible to define that *EU-/hasPopulation* is dependent on the *population* of the *MemberCountries*. It would not be sufficient to state that it is dependent on *population* in general.
3. It should be possible to define that one artifact is dependent on a set of other artifacts.
4. Multiple dependencies should be possible for one change-dependent concept or property.

According to the first requirement the dependent artifact needs to be specified precisely. This can be realized with the annotation path syntax. Therefore, the path *EU/hasPopulation* defines that the *hasPopulation* property on the concept *EU* is the subject of a dependency definition. The second requirement supposes that not only the subject should be described via path expressions but also the object of a change-definition should be described in terms of path expressions. Unfortunately, the plain annotation path syntax does not fulfill the third requirement. It, therefore, needs to be enhanced with expressions to address sets.

Dependencies on Sets and Aggregations: Some ontology artifact may be change-dependent on a set of other artifacts. In our running example the population of the *EU* depends on the set of *MemberCountries*. More precisely it is not dependent on the set of $MemberCountries$ in general but on the sum of the *hasInhabitants* property of each *MemberCountry*. In general, there are different kinds of sets in an ontology: subclasses, sub-properties and instances. Therefore, all those must be expressible. The sum function is only one aggregation function. Typical other aggregation functions are are min, max, count, and avg. In addition to aggregation functions another kind of function over sets is of interest: The value function. It can be used to state than one artifact is directly dependent on the values of a set of other artifacts. The subclasses and instances operator can be used in a path wherever a concept is allowed and the sub-properties operator can be used wherever a property-step is allowed. We will illustrate the ideas with examples:

1. *EU/hasPopulation* is dependent on *$/EU/consistsOf/sum(subclasses(MemberCountry))/hasInhabitants$.*
2. *EU/numberOfCommissioners* is dependent on *$/EU/hasCommission/$ $EUCommission/hasMember/count(instances(Commissioner))$.*
3. *MemberCountry/hasInhabitants* is dependent on *MemberCountry/subproperties(hasInhabitants).*
4. */city* is dependent on *value(subclasses(city))*

The first example calculates the sum of all *hasInhabitants* properties of all subclasses of $member - countries$, while the second one just counts the number of *commissioner* instances. The first example defines an abstract sum because the ontology cannot contain any information about the number of inhabitants on class level. It only defines that the value becomes invalid if the ontology structure changes in a way that the function would operate on a changed-set of ontology artifacts. In contrast the second example can return a defined number because it is a simple count operation. In addition, it defines the change-dependency over instances. In this case the ontology may contain instance data. In the third example it is assumed that the *hasInhabitants* property has sub-properties and that a change of the sub-properties will also invalidate *EU/hasPopulation*. Examples for sub-properties could be *hasMalePopulation* and *hasFemalePopulation*.

The last example shows the value function. It defines that elements that are annotated with */city* are change-dependent on all the subclasses of *city*. Therefore, a rename of a subclass of *city* requires a rename of the specific city-element in XML-documents as well.

4 Definition of Change-Dependencies

In order to introduce the proposed change-dependency definitions we will first define our ontology model. We use an abstract ontology model which can be compared to RDFS[2] shown graphically in figure 2. It contains the relevant aspects of typical ontology languages. Basically an ontology consists of concepts, properties and individuals. Concepts and properties are hierarchically structured. An individual is an instance of a set of concepts. A property has a domain that defines the set of classes that have this property. Properties are divided into object-properties and datatype-properties. Object-properties form relationships between classes and, therefore, have a range that defines the set of classes which are targets of the property. Datatype-properties have a definition of the data type. Properties are modeled on the class level while an instantiation of a property is done on instance-level using property assertions. Concepts may restrict the usage of properties. A restriction has a type and a value. The type can be a typical OWL [1] cardinality- or value- restriction. The type indicates the type of restriction (min, max, value) the value indicates the value that is attached to this restriction. Individuals can have *propertyAssertions* that indicate that an individual instantiates some property. In case of an object-property the target of a *propertyAssertion* is another individual. In case of datatype-properties it is some data-value. This abstract ontology model covers the important concepts of most ontology languages. By using this abstract model we can apply the work on different ontology formalisms that can be transformed to our representation. This does not require to transform the whole ontologies to our ontology model. It is sufficient to formulate the changes that occur to the ontology in terms of our ontology-model. Depending on the used ontology formalism reasoning may induce additional changes that we can simply also add to our change-log by comparing the materialized ontology version before and after the change.

Based on this ontology model we define annotation paths that are used to annotate artifacts of an XML-Schema with a reference ontology. Both annotation path and dependency-definitions are modeled in the meta-model in figure 3. An *AnnotationPath* consists of a sequence of *AnnotationPathSteps*. Each step has

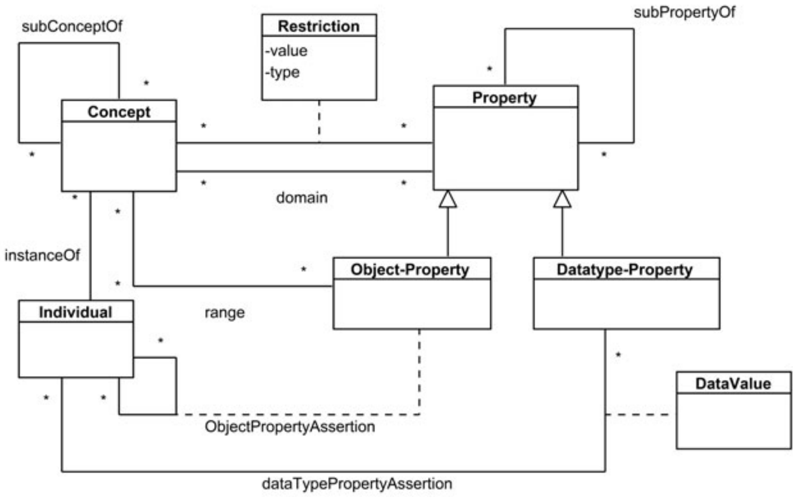


Fig. 2. Ontology Meta-Model

a position and a *uri* that points to some property or concept of the reference ontology. According to the referenced ontology artifact an *AnnotationPathStep* is either a *conceptStep* or a *propertyStep* with the subclasses *objectPropertyStep* and *dataTypepropertyStep*. Each step except the last step has a succeeding step. Each step except the first step has a previous step. An annotation path has a defined first and a defined last-step.

A *DependencyDefinition* has one *hasSubject* relation to an *AnnotationPath* and one or more *hasObject* relations to *dependencyDefinitionPath*. Each *dependencyDefinitionPath* consists of a number of *DependencyPathSteps*. A *DependencyPathStep* is a subclass of an *annotationPathStep* which is extended with an optional *setExpression*. The *setExpression* has a *type (subclasses, subproperties, instances)* and an optional *function* which has a type that can be *value*, *min*, *max*, *avg*, *count*. Each *DependencyDefinitionPath* has a *hasAnnotationPath* relation to one *AnnotationPath*. This specific *annotationPath* is created by casting all steps to standard *AnnotationPathSteps*. An *annotationPath* can be represented in form of an ontology concept. This concept can be obtained with the method *getConcept*().

In order to meet the requirements that were introduced in the last section some integrity constraints on *AnnotationPath* and *DependencyDefinitionpath* are required. We refer the interested reader for the complete definition of *annotation Path* to [14].

4.1 Integrity Constraints on *AnnotationPath*

1. The first step must be a *ConceptStep*.
2. An *AnnotationPath* must not contain *DependencyPathSteps*.
3. The last step must be a *ConceptStep* or a *dataTypePropertyStep*.
4. When a *conceptStep* has a previous step then the previous step must be an *ObjectPropertyStep*.
5. The next step of a *ConceptStep* must be an *ObjectPropertyStep* or a *DataTypePropertyStep*.
6. A *DataTypePropertyStep* can only exist as the last-step.
7. A *ConceptStep* must not reference to another *AnnotationPath*.

4.2 Integrity Constraints on *DependencyDefinitionPath*

1. All integrity constraints of standard steps except (2) and (7) also apply on *DependencyDefinitionPath*.
2. Only the last two steps may have a *setExpression* including a *function*.
3. The *setExpression* of type *subclasses* is only allowed for *conceptSteps*.
4. The *setExpression* of type *instances* is only allowed for *conceptSteps*.
5. The *setExpression* of type *subproperties* is only allowed for *propertySteps*.

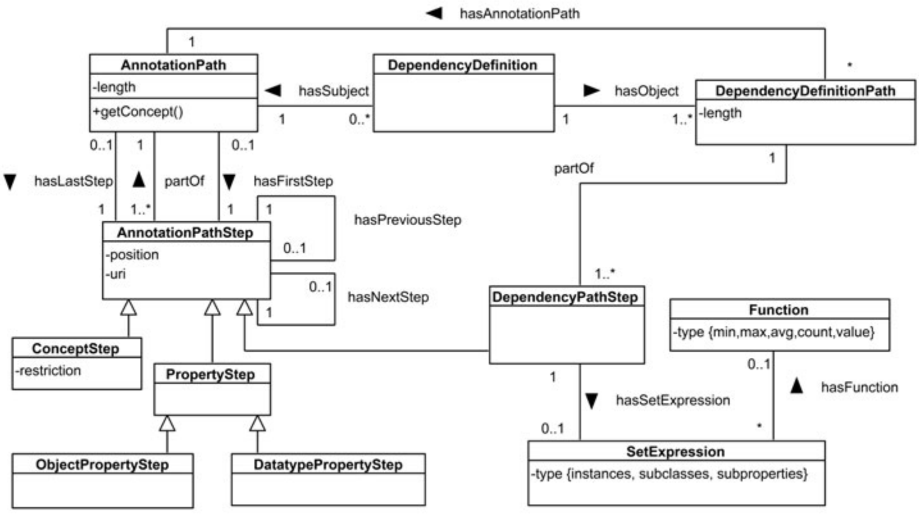


Fig. 3. Meta-Model of the Change-Dependency Definitions

5 Detection of Semantic Changes

In order to detect if the semantics of an annotation path got invalid by a change we first introduce the change-model for our ontology-model. The changes are stored in a change-log and the detection of relevant changes is realized by rules that operate on the statements of the change-log and the old and the new ontology versions.

5.1 Change-Log

The change-log consists of a set of changes C. Each change $c \in C$ is a tuple $c = (op, tid, p1, ...pn)$. Where op defines the kind of change, tid is a unique identifier of a change and also creates an order over the changes. A change with a lower tid was made before a change with a higher tid. The parameters $p1$ .. pn are parameters for the change. A multi-version ontology O has different versions $O_1...O_n$. Each version has a timestamp tid that uniquely identifies the version and also creates an order over the different versions. The set of changes that were made between two ontology versions O_n and O_{n+1} is denoted $Changes(O_n, O_{n+1}, C)$.

$$Changes(O_n, O_{n+1}, C) = \{\forall c \in C | c.tid \geq O_n.tid \wedge c.tid \leq O_{n+1}.tid\}$$

There are different change operations. We will discuss the different operations in the next section. For readability reasons we present the changes in form of predicates. Thus, $c.op$ is the name of the predicate. In addition each change has an inverse change c^{-1} that compensates the change-effect of the change c. We assume that the changes in $Changes(O_n, O_{n+1}, C)$ are free of redundancies between O_n and O_{n+1} such that no change is compensated by an inverse change in the change-log.

Global Atomic Changes: Atomic changes are basic changes that add or remove concepts, properties or instances. It must be ensured by the ontology management system that a removal of a concept, property or instance is only possible when the artifact is not, still, referenced. The table shows the basic add-operations and their inverse delete operations.

c	c^{-1}
addConcept(tid,uri)	delConcept(tid,uri)
addOProp(tid,uri)	delOProp(tid,uri)
addDProp(tid,uri)	delDProp(tid,uri)
addInstance(tid,uri)	delInstance(tid,uri)

Hierarchy Changes: In addition to theses basic operations the following operations that maintain the hierarchy of the artifacts are required. They are used to express changes in the concept- or property-hierarchy.

c	c^{-1}
addChildC(tid,childUri,parentUri)	remChildC(tid,childUri,parentUri)
addChildOProp(tid,childUri,parentUri)	remChildOProp(tid,childUri,parentUri)
addChildDProp(tid,childUri,parentUri)	remChildDProp(tid,childUri,parentUri)
addInstToC(tid,instanceUri,conceptUri)	remInstToC(tid,instanceUri,conceptUri)

Update Changes: Update changes are used to modify the domain and range of properties as well as to maintain restrictions over properties on concepts and to modify property assertions on individuals. The rename operations change the URI of a specific concept, property or individual. We assume that these operations are global in the sense that every usage of the URI is changed automatically. In addition we assume that these operations are added to the change-log as atomic operations. Therefore, a rename does only show up as a rename but not as a delete and subsequent insert in the change-log. The inverse operations of update-changes are update operations of the same type but with swapped parameters.

– updateRestriction(tid, conceptUri, propertyUri, oldValue, oldType, newValue, newType)
– updateDomain(tid, propertyUri, {oldConceptUri}, {newConceptUri})
– updateRange(tid, propertyUri, {oldconceptUri}, {newconceptUri})
– updateType(tid, propertyUri, oldDataType, newDataType)
– updatePropertyAssertion(tid, instanceUri, propertyUri, oldValue, newValue)
– renameConcept(tid, oldUri, newUri)
– renameProperty(tid, oldUri, newUri)
– renameIndividual(tid, oldUri, newUri)

Composite Changes: In addition to the basic operations a set of composite operations is of interest. A merge operation merges a a set of concepts, properties or instances to one single concept, property or instance. The inverse operation of a merge is a split.

c	c^{-1}
mergeC(tid,{conceptUri},conceptUri)	splitC(tid,conceptUri,{conceptUri})
mergeP(tid,{propertyUri},propertyUri)	splitP(tid,propertyUri,{propertyUri})
mergeI(tid,{instanceUri},instanceUri)	splitI(tid,instanceUri,{instanceUri})

A composite operation is reflected as a sequence of other change-operations. In order to specify that an explicit change-operation is part of a composite change the atomic operation is annotated with the *tid* of the corresponding composite change with statements of the form:
$ChangeAnnotation(tidOfCompositeChange, tidOfAtomicChange)$

5.2 Implicit Changes

In addition to explicit changes there are implicit changes. These changes can directly be caused by explicit changes or by classification according to the source ontology language. For direct explicit changes we except the following implicit changes to be automatically included in the change-log by the ontology management system.

- If a concept is added or removed as a child of an existing concept and the existing concept or one of its parents has a restriction on a property then a restriction change is made on the added or removed concept implicitly.
- If a property is added or removed as a child of another property then the domain and range of the added or removed property is changed implicitly.
- If an individual is added or removed to/from a concept then it is added/removed to all its super-concepts implicitly.

The implicit changes as shown above and additional changes that can be derived by comparing the materialized ontology versions of the ontology before and after each change are stored in the change-log. The comparison algorithm can benefit from the fact that the change-log contains information about renames, additions and removes. Therefore, the complexity is strongly reduced since each element from the source and target ontology can directly be mapped. In order to trace which change caused the addition of these implicit changes the implicit changes are annotated with predicates of the form $causedBy(impl\_tid, tid)$. This allows to keep the change-log clean of redundant implicit changes if the change that caused the implicit changes is compensated by an inverse operation.

5.3 Detection of Semantically Invalid Annotation Paths

Now we define semantic invalidation as a change affecting the semantics of an annotion path as follows.

The predicate $subClassOf(subc, superc, O_n)$ states that *subc* is a subclass of the superclass *superc* according to the ontology version O_n. As an equivalent class is logically defined as being sub- and superclass at the same time we assume that every class is a subclass of itself. The predicate $subPropertyOf(subp, superp, O_n)$ expresses the sub-property-relationship analogously.

Definition 1. ***Semantic Invalidation of an Annotation-Path***

Given an ontology version O_n, a succeeding ontology version O_{n+1}, a set of changes $Changes(O_n, O_{n+1})$ abbreviated by C, a set of explicit dependency-definitions DEP, and a set of XML-Schema-annotations A. An annotation path $a \in A$ is semantically invalid if:

$semInvalid(a, C, DEP, O_n, O_{n+1}) \leftarrow InvalidByDep \neq \{\}$

RelevantDependencies is the set of definitions where the corresponding subject is an equivalent- or superclass of the annotation path a:
$RelevantDependencies = \{\forall dep \in DEP | subClassOf(a, dep.subject, O_n)\}$
InvalidByDep is the set of change-dependency definitions where one of the objects got invalid because it contains a step that is invalid or if the semantics of the annotation path of the object itself got changed.

$InvalidByDep = \{\forall dep \in RelevantDependencies | (hasObject(dep, obj)$
$\wedge\, isInvalid(obj)) \vee (hasAnnotationPath(obj, annotationPathObject)$
$\wedge\, semInvalid(annotationPathObject, C, DEP, O_n, O_{n+1}))\}$

Thus, dependency-definitions are transitive. If a depends on b and b depends on c then a is invalid when c is invalid.

A $DependencyDefinitionPath$ is invalid if at least one of its steps is invalid:
$isInvalid(obj) \leftarrow \exists step \in obj.steps \wedge InvalidStep(step)$
When a step is invalid is described in the next subsections.

Rules for the Invalidation of Steps: For the sake of simplicity we will define the invalidation of steps in form of rules omitting quantifiers. In addition all rules operate on the change-set defined by $Changes(O_n, O_{n+1}, C)$. Rules for the detection of additions operate on O_{n+1} while rules for the detection of removals operate on O_{n-1}. The rules 1-3, 6, 8, 10 create possible invalidations, while the others create invalidations. Possible invalidations are invalidations where an invalidation may have taken place but additional review by the user is required.

1. A *PropertyStep* gets possibly invalid, if the domain of the property or of a super-property has changed.
   $PropertyStep(?step) \wedge subPropertyOf(?step.uri, ?superProperty,$
   $O_{n+1}) \wedge updateDomain(\_, ?superProperty, \_, \_)$
   $\Rightarrow InvalidStep(?step, 'DomainOfPropertyChanged')$

2. A property-step is possibly invalid, if the range of the property or a super-property has changed.
   $ObjectTypePropertyStep(?step) \wedge subPropertyOf(?step.uri, ?superProperty, O_{n+1}) \wedge updateRange(\_, ?superProperty, \_, \_)$
   $\Rightarrow InvalidStep(?step,' RangeOfPropertyChanged')$
   The same holds for the change of the data type of a datatype-property analogously.

3. A concept-step gets possibly invalid, if a restriction on the property of the next step has changed.
   $ConceptStep(?step) \wedge isSubConceptOf(?step.uri, ?superuri, O_{n+1}) \wedge hasNextStep(?step, ?next) \wedge subPropertyOf(?next.uri, ?supernexturi, O_{n+1}) \wedge updateRestriction(\_, ?superuri, ?supernexturi, \_, \_, \_, \_)$
   $\Rightarrow InvalidStep(?step,' RestrictionOnNexStepChanged')$

4. A set expression over subclasses without a function becomes invalid, if a subclass is added.
   $ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge !hasFunction(?exp, \_) \wedge equals(?exp.type,' subclasses') \wedge subConceptOf(?suburi, ?step.uri, O_{n+1}) \wedge addChildC(\_, ?newc, ?suburi)$
   $\Rightarrow Invalid(?step,' SubclassAdded')$

5. A set expression over subclasses without a function becomes invalid, if a subclass is removed.
   $ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge !hasFunction(?exp, \_) \wedge equals(?exp.type,' subclasses') \wedge subConceptOf(?suburi, ?step.uri, O_n) \wedge remChildC(\_, ?newc, ?suburi)$
   $\Rightarrow Invalid(?step,' SubclassRemoved')$

6. A set expression over subclasses without a function becomes possibly invalid, if a restriction on the property of the next step is changed in one of the sub-concepts.
   $ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge !hasFunction(?exp, \_) \wedge equals(?exp.type,' subclasses')$
   $\wedge subConceptOf(?suburi, ?step.uri, O_{n+1}) \wedge hasNextStep(?step, ?next)$
   $\wedge subPropertyOf(?nextpropuri, ?next.uri, O_{n+1}) \wedge updateRestriction(\_, ?suburi, ?nextpropuri, \_, \_, \_, \_)$
   $\Rightarrow Invalid(?step.'RestrictionOnSubclassChanged')$

7. A set expression over instances without a function becomes invalid, if instances are added or removed to/from the specified concept or one of its subconcepts. We will only depict the rule for the addition here.
   $ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge !hasFunction(?exp, \_) \wedge equals(?exp.type,' instances') \wedge subConceptOf(?conceptUri, ?step.uri, O_{n+1})$
   $\wedge addInstToC(\_, \_, ?conceptUri) \Rightarrow Invalid(?step,' InstancedAdded')$

8. A set expression over instances becomes possibly invalid, if the succeeding-step is a property-step and property assertions on instances for that property are modified.
$ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge$
$!hasFunction(?exp, \_) \wedge equals(?exp.type,' instances') \wedge$
$subConceptOf(?conceptUri, ?step.uri, O_{n+1}) \wedge hasNextStep(?step, ?next) \wedge$
$PropertyStep(?next) \wedge instanceOf(?insturi, ?conceptUri) \wedge$
$updatePropertyAssertion(\_, ?insturi, ?next.uri, \_, \_)$
$\Rightarrow Invalid(?step,' PropertyAssertionChanged')$

9. A set expression over sub-properties becomes invalid, if a sub-property is added or removed. We will only depict the rule for the addition here.
$PropertyStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge$
$!hasFunction(?exp, \_) \wedge equals(?exp.type,' subproperties') \wedge$
$subPropertyOf(?suburi, ?step.uri, O_{n+1})$
$\wedge(addChildOProp(\_, ?newc, ?suburi) \vee (addChildDProp(\_, ?newc, ?suburi))$
$\Rightarrow Invalid(?step,' SubpropertyAdded')$

10. A set expression over sub-properties becomes possibly invalid, if the domain or range of a sub-property is changed. *uDomainOrRange* is the superclass of *updateDomain* and *updateRange*
$PropertyStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge$
$!hasFunction(?exp, \_) \wedge equals(?exp.type,' subproperties')$
$\wedge subPropertyOf(?suburi, ?step.uri, O_{n+1}) \wedge$
$uDomainOrRange(\_, ?suburi, \_, \_)$
$\Rightarrow Invalid(?step,' DomainOrRangeOfSubpropertyChanged')$

Functions on Set Expressions: The rules in the last section excluded the existence of functions over $setExpressions$. Therefore, any change that has consequences for the $setExpression$ is considered to invalidate the step. When a function is given then the problematic change-operations depend on the used function and, therefore, additional rules are required. The sum-function is vulnerable to *add* and *delete* operations but is resistent to local *merge* or *split* operations. All other aggregation functions are vulnerable to *add*, *del*, *split*, and *merge*. The value function is vulnerable to renames of sub-concepts or sub-properties as well as to *delete*, *split* and *merge* operations. Therefore, specific rules for the different kinds of functions are required. Since *merge* and *split* are complex change-operations the rules need to operate on the annotation of the changes $(ChangeAnnotation(...))$. Due to space limitations we will only provide rules for sum-functions with added sub-concepts and value-functions with renames.

1. A concept-step with a sum-function over sub-concepts gets invalid, if sub-concepts are added or removed and the add and remove operations are not linked to local split or merge operations. A non-local split operation happens when the source concept was a sub-concept of the step and one of the new concepts is not, still, a sub-concept of the step according to the current

ontology version. The following rule represents the case of the addition of sub-concepts.

$$\begin{aligned}
&ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge \\
&hasFunction(?exp, ?fu) \wedge equals(?fu.type,' sum') \wedge \\
&subConceptOf(?suburi, ?step.uri, O_{n+1}) \wedge addChildC(?tid, ?newc, ?suburi) \\
&\wedge!(splitC(?stid, ?source, ?suburi) \wedge ChangeAnnotation(?stid, ?tid) \\
&\wedge subConceptOf(?source, ?step.uri, O_n) \wedge \\
&!(splitC(?stid, ?source, ?otherSplitUri) \wedge notequals(?otherSplitUri, ?newc) \\
&\wedge addChildC(?tid, ?otherSplitUri, ?otherAddUri) \\
&subClassOf(?otherAddUri, ?step.uri, O_{n+1}))) \\
&\Rightarrow Invalid(?step,' SubclassAdded')
\end{aligned}$$

2. A concept-step with a value-function gets invalid if one of the sub-concepts is renamed or deleted. We will show the rule for the renames here.

$$\begin{aligned}
&ConceptStep(?step) \wedge hasSetExpression(?step, ?exp) \wedge \\
&hasFunction(?exp, ?fu) \wedge equals(?fu.type,' value') \wedge \\
&subConceptOf(?oldUri, ?step.uri, O_{n+1}) \wedge \\
&renameConcept(\_, ?oldUri, ?newUri) \\
&\Rightarrow Invalid(?step,' Value - Changed')
\end{aligned}$$

6 Proof of Concept Implementation

We have implemented this approach for computing semantic invalidations of annotation paths using the Jena API[3] and the pellet[4] reasoner. The input for the algorithm consists of a set of annotation paths, a set of dependency-definition paths, and information about renames, splits and merges. The output is a subset of the input annotation path where the semantics has (possibly) changed. Additionally, explanations for the semantic invalidations are provided. The system transforms the materialized source and target ontology to instances of our ontology-meta-model as shown in figure 2. In a next step SPARQL[5] queries are used to generate the change-log. Each log-entry is an individual of a change-ontology. The change ontology is a representation of the change-hierarchy defined in section 5.1.

The rules as proposed in section 5.3 are implemented in form of SPARQL queries that operate on instances of the change-log and the instances of the meta-ontology. The required negation is implemented in form of SPARQL filters.

The special property $subClassOf(c1, c2, ?v)$ (and $subpropertyOf(p1, p2, ?v)$ analogously) is realized in form of two distinct properties $subClassOfOld(c1, c2)$ and $subClassofNew(c1, c2)$ that are added to the instances of the source and target ontology version of the ontology meta-model. Most invalidation rules can

[3] http://jena.sourceforge.net/
[4] http://clarkparsia.com/pellet/
[5] http://www.w3.org/TR/rdf-sparql-query/

directly be represented in SPARQL, while some more complex queries need additional post-processing. The prototype demonstrated the feasibility of our approach.

7 Related Work

In [10] the consistent evolution of OWL ontologies is addressed. The authors describe structural, logical and user-defined consistency requirements. While the structural and logical requirements are directly defined by the used ontology formalism the user-defined requirements describe additional requirements from the application domain that cannot be expressed with the underlying ontology language. The authors do not make any suggestion on how these requirements should be expressed. Therefore, our approach can be seen as one specific form of user-defined-consistency requirements. The main difference is that in our case the artifact that becomes inconsistent if a user-defined consistency-definition is violated is not the ontology itself but instance-data in XML-documents. In [4] functional dependencies over Aboxes (individuals) are addressed. The dependencies are formulated in the form antecedent, consequent and an optional deterministic function. The antecedent and consequent are formulated via path expressions which can be compared to our approach. The dependencies are directly transformed to SWRL-rules. Therefore, the functional dependencies directly operate on the individuals (Abox) and additional knowledge can be added to the Abox. In addition, data that does not comply with the rules can be marked as inconsistent. In contrast to our approach, the approach is limited to the instance layer which makes it unusable for our scenario where instance-data from XML-documents is never added to the Abox. Therefore, knowledge about changes needs to be evaluated in order to predict semantic-changes of the semantics of instance-data.

In [18] the validity of data-instances after ontology evolution is evaluated. An algorithm is proposed that takes a number of explicit changes as input and calculates the implicit changes that are induced by the explicit changes. These explicit and implicit changes can then be used to track the validity of data-instances. The general idea of the approach is that if an artifact gets more restricted existing instances are invalidated. Since the approach only takes into account implicit changes that can be computed based on explicit changes it does not support the explicit definition of change-dependencies.

Our work heavily depends on the existence of an expressive change-log between two versions of an ontology. The automatic detection of changes that happened between two versions of an ontology are covered with approaches like [15], [9] or [11]. If a change-log exists this can also be used as a basis to generate more expressive changes as required by our approach. Approaches that operate on a change-log in order to generate additional changes are [12], [17]. Methods to efficiently store and manage different ontology versions are presented in [8],[5].

While there is only limited work on dependencies in the field of ontologies it is traditionally broadly studied in the database community. Recent and related research in this field is for example [3] and [6]. In [3] a model and system is presented that keeps track of the provenance of data that is copied from different (possibly curated) databases to some curated database. Changes in the source databases may influence the data in the target databases. Therefore, provenance information is required to track those changes. In [6] the problem of provenance in databases is formalized with an approach that is inspired by dependency analysis techniques known from program analysis or slicing techniques. In contrast to our approach both provenance approaches cope with changes of instance data and do not address changes of schema/meta-data.

8 Conclusion

In this paper we have addressed the problem of semantic changes of annotations that occur due to the evolution of their reference ontologies. As ontologies have to keep up with changes in the modeled domain (the real world) such changes occur frequently. Unrecognized semantic changes (may) lead to incorrect results for document transformations, semantic queries, statistics etc. based on semantic annotations. So there is an urgent necessity to maintain semantic annotations when a reference ontology changes. As experience shows, high maintenance costs are a severe obstacle against wide adoption of techniques. We presented a technique to identify those annotations which have to be considered for maintenance due to changes in the reference ontology. This should ease the burden of maintenance considerably.

References

1. OWL web ontology language reference. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/owl-ref/
2. RDF vocabulary description language 1.0: RDF Schema. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/rdf-schema/
3. Buneman, P., Chapman, A.P., Cheney, J.: Provenance management in curated databases. In: Proc. of SIGMOD 2006, pp. 539–550. ACM (2006)
4. Calbimonte, J.-P., Porto, F., Maria Keet, C.: Functional dependencies in owl abox. In: Brayner, A. (ed.) Proc. of SBBD 2009, pp. 16–30. SBC (2009)
5. Chen, C., Matthews, M.M.: A new approach to managing the evolution of owl ontologies. In: Arabnia, H.R., Marsh, A. (eds.) Proc. of SWWS 2008, pp. 57–63. CSREA Press (2008)
6. Cheney, J., Ahmed, A., Acar, U.A.: Provenance as Dependency Analysis. In: Arenas, M. (ed.) DBPL 2007. LNCS, vol. 4797, pp. 138–152. Springer, Heidelberg (2007)
7. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M.: Ontology Modularization for Knowledge Selection: Experiments and Evaluations. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 874–883. Springer, Heidelberg (2007)
8. Eder, J., Koncilia, C.: Modelling Changes in Ontologies. In: Meersman, R., Tari, Z., Corsaro, A. (eds.) OTM-WS 2004. LNCS, vol. 3292, pp. 662–673. Springer, Heidelberg (2004)

9. Eder, J., Wiggisser, K.: Change Detection in Ontologies Using DAG Comparison. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 21–35. Springer, Heidelberg (2007)
10. Haase, P., Stojanovic, L.: Consistent Evolution of OWL Ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 182–197. Springer, Heidelberg (2005)
11. Hartung, M., Gross, A., Rahm, E.: Rule-based generation of diff evolution mappings between ontology versions. CoRR, abs/1010.0122 (2010)
12. Khattak, A.M., Latif, K., Han, M., Lee, S., Lee, Y.-K., Kim Il, H.: Change tracer: Tracking changes in web ontologies. In: Proc. of ICTAI 2009, pp. 449–456. IEEE Computer Society (2009)
13. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: Sawsdl: Semantic annotations for wsdl and xml schema. IEEE I.C. 6, 60–67 (2007)
14. Köpke, J., Eder, J.: Semantic Annotation of XML-Schema for Document Transformations. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6428, pp. 219–228. Springer, Heidelberg (2010)
15. Noy, N.F., Musen, M.A.: Promptdiff: a fixed-point algorithm for comparing ontology versions. In: Proc. of AAAI 2002, pp. 744–750. AAAI (2002)
16. Noy, N.F., Musen, M.A.: Specifying Ontology Views by Traversal. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 713–725. Springer, Heidelberg (2004)
17. Plessers, P., De Troyer, O.: Ontology Change Detection Using a Version Log. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 578–592. Springer, Heidelberg (2005)
18. Qin, L., Atluri, V.: Evaluating the validity of data instances against ontology evolution over the semantic web. Inf. Softw. Technol. 51, 83–97 (2009)
19. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-Driven Ontology Evolution Management. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 285–300. Springer, Heidelberg (2002)
20. Uren, V., Cimiano, P., Iria, J., et al.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. Web Semantics: Science, Services and Agents on the World Wide Web 4(1), 14–28 (2006)

The Role of Constraints in Linked Data

Marco Antonio Casanova[1], Karin Koogan Beitman[1], Antonio Luz Furtado[1], Vania M.P. Vidal[2], José A.F. Macedo[2], Raphael Valle A. Gomes[1], and Percy E. Rivera Salas[1]

[1] Department of Informatics – PUC-Rio – Rio de Janeiro, RJ – Brazil
{casanova,karin,furtado,rgomes,psalas}@inf.puc-rio.br
[2] Department of Computing, Federal University of Ceará – Fortaleza, CE – Brazil
{vvidal,jose.macedo}@lia.ufc.br

Abstract. The paper argues that a Linked Data source should publish an application ontology that includes a set of constraints that capture the semantics of the classes and properties used to model the data. Furthermore, if the Linked Data source publishes a mapping between its vocabulary and the vocabulary of a domain ontology, then it has to specify the application ontology constraints so that they are consistent with those of the domain ontology. The main contributions of the paper are methods for constructing the constraints of the application ontology of a Linked Data source, defined as fragments of domain ontologies. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, which explores the structure of sets of constraints.

Keywords: constraints, application ontology, Linked Data.

1 Introduction

The term *Linked Data* refers to a set of best practices for publishing and connecting structured data on the Web [4]. A Linked Data source may publish an *application ontology* that models the exported data and a mapping between the application ontology vocabulary and a *domain ontology* vocabulary (or several such vocabularies). The mapping may be expressed as a set of RDF triples that link classes and properties in one vocabulary to those in another, or it may be defined using a schema mapping language.

In this paper, we argue that a Linked Data source should include, in the definition of the application ontology, a set of constraints that capture the semantics of the classes and properties used to model the data. Furthermore, if the Linked Data source publishes a mapping between its vocabulary and the vocabulary of the domain ontology, then it has to specify the application ontology constraints so that they are consistent with those of the domain ontology.

More precisely, an *ontology* is a pair $O=(V_O,C_O)$, where V_O is a vocabulary and C_O is a set of constraints over V_O. A *domain ontology* $\mathcal{D}=(V_D,C_D)$ models the application domain. In fact, $\mathcal{D}$ may be a combination of ontologies covering distinct domains. An *application ontology* $\mathcal{A}=(V_A,C_A)$ models the data exported by a Linked Data source.

The problem we address can be formulated as follows: "Given that V_A is a subset of V_D, how to derive C_A from C_D". We offer two alternative answers to this question, depending on what we require from the data exported by the data source.

Suppose first that we require that the data exported by the Linked Data source must satisfy C_A, and that C_A must logically imply all constraints that can be derived from C_D and that use only symbols in V_A. In this case, we say that the application ontology is an *open fragment* of the domain ontology. Section 4 formulates these requirements in detail and describes a method to derive C_A.

Suppose now that we require that the data exported by the Linked Data source must satisfy C_D, when all classes and properties in V_D, but not in V_A, are taken as the empty set (when the source data is published). In this case, we say that the application ontology is a *closed fragment* of the domain ontology. Section 5 addresses this case.

Applications may benefit from these concepts as follows. Consider a Linked Data source S whose data is published according to an application ontology $\mathcal{A}$. Suppose first that $\mathcal{A}$ is designed as an open fragment of $\mathcal{D}$. Then, in general, any application that processes data modeled according to $\mathcal{D}$ and *that uses only the classes and properties in the vocabulary of* $\mathcal{A}$ will also be able to process data published by S (since the application expects data consistent with the constraints derived from C_D that apply to the classes and properties in the vocabulary of $\mathcal{A}$). Now, suppose that $\mathcal{A}$ is designed as a closed fragment of $\mathcal{D}$. Then, any application that processes data modeled according to $\mathcal{D}$ will also be able to process data published by S (since the application expects data consistent with the constraints in C_D).

In particular, consider a query optimizer that wishes to submit a query Q to the data source S. Again, suppose first that $\mathcal{A}$ is designed as an open fragment of $\mathcal{D}$. Then, if Q uses only the classes and properties in the vocabulary of $\mathcal{A}$, the optimizer needs to consider only the constraints derived from C_D that apply to the classes and properties in the vocabulary of $\mathcal{A}$. Indeed, any data from S will satisfy such constraints since $\mathcal{A}$ was designed as an open fragment of $\mathcal{D}$. Any other constraint derived from C_D will be irrelevant to the process. Suppose now that $\mathcal{A}$ is designed as a closed fragment of $\mathcal{D}$. Then, the optimizer needs to consider the constraints in C_D, but it may also assume that any class or property not in the vocabulary of $\mathcal{A}$ is empty (by definition of closed fragment). This opens new opportunities for relatively straightforward optimizations.

The main contributions of the paper are methods for constructing application ontology constraints when the application ontology is an open or a closed fragment of the domain ontology. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, which explores the structure of sets of constraints, captured as *constraint graphs* [7]. The methods are also useful in other application domains, such as data integration and data mashups, where a set of constraints have to be constructed from other sets of constraints.

The paper is organized as follows. Section 2 further discusses the motivation for the paper. Section 3 presents the formal framework adopted in the paper. Section 4 focuses on how to construct open fragments of the domain ontology. Section 5 analyses the case of closed fragments. Section 6 summarizes related work. Finally, Section 7 contains the conclusions.

2 An Informal Example

The ‘Linked Data Principles’ [2] provide a basic recipe for publishing and connecting data using the infrastructure of the Web. From an application development perspective, Linked Data has the following characteristics [5]:

1. Data is strictly separated from formatting and presentational aspects.
2. Data is self-describing. If an application consuming Linked Data encounters data described with an unfamiliar vocabulary, the application can dereference the URIs that identify vocabulary terms in order to find their definition.
3. The use of HTTP as a standardized data access mechanism and RDF as a standardized data model simplifies data access compared to Web APIs, which rely on heterogeneous data models and access interfaces.
4. The Web of Data is open, meaning that applications do not have to be implemented against a fixed set of data sources, but they can discover new data sources at run-time by following RDF links.

We are particularly interested in the second characteristic. The definition of vocabulary terms ultimately includes a set of constraints that capture the semantics of the terms. Therefore, when publishing Linked Data, we argue that the designer should go further and analyze the constraints of the ontology from which he is drawing the terms to construct his vocabulary. We further motivate this argument with the help of examples, adopting the *Music Ontology* (***MO***) [17] as the domain ontology.

The Music Ontology is used by several Linked Data sources, including MusicBrainz and BBC Music. The Music Ontology RDF schema uses terms from the ***FRBR*** [14], ***FOAF*** [6] and the XML Schema vocabularies. We adopt the prefixes “mo:”, “frbr:”, “foaf:” and “xsd:” to respectively refer to the ***MO***, ***FRBR***, ***FOAF*** and XML Schema vocabularies.

Fig. 1 shows the class hierarchies of ***MO*** rooted at classes event:Event and frbr:Expression. Fig. 2 shows the class hierarchies of ***MO*** rooted at classes foaf:Agent and foaf:Person.

Suppose that the designer wants to publish a dataset, using ***MO*** as the domain ontology. He then proceeds to define an application ontology, which we call *Signal ontology* (***SGL***). As the first step, he selects classes and properties from the ***MO*** vocabulary to create the ***SGL*** vocabulary. Assume that he selects classes mo:Signal, mo:DigitalSignal and mo:analogSignal, the datatype property mo:isrc and the object property mo:sampled_version to form the ***SGL*** vocabulary (all shown in Fig. 1).

Usually, strategies to publish Linked Data treat domain ontology vocabularies only up to this stage. We argue that the strategies should go further and include an analysis of the constraints of the domain ontology that apply to the data being published.

By observing the ***MO*** constraints (informally depicted in Fig. 1), the designer may directly derive the following constraints for the application ontology ***SGL***:

- mo:DigitalSignal and mo:analogSignal are subclasses of mo:Signal
- mo:DigitalSignal and mo:analogSignal are disjoint classes
- the domain and range of mo:sampled_version are mo:analogSignal and mo:DigitalSignal, respectively
- the domain and range of mo:isrc are mo:Signal and xsd:String, respectively

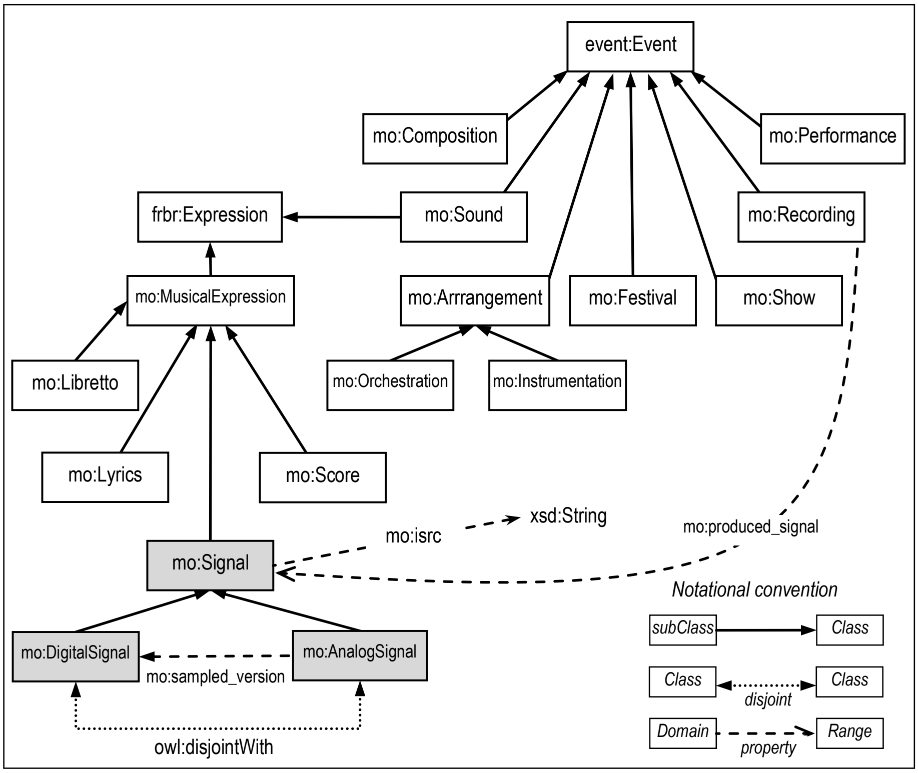


Fig. 1. The class hierarchies of ***MO*** rooted at classes event:Event and frbr:Expression

The constraints of the application ontology ***SGL*** were straightforward to obtain because they were exactly the constraints of ***MO*** that involve the classes and properties in the ***SGL*** vocabulary. Hence, if the designer publishes his data source so that the set of triples satisfies the ***SGL*** constraints, then any Web application that processes data modeled according to ***MO*** and *that uses only the classes and properties in the* ***SGL*** *vocabulary* will also be able to process the triples published by the data source. This follows because the application expects data consistent with the ***MO*** constraints that apply to the classes and properties it uses.

Suppose that the designer wants to publish a second dataset, again using ***MO*** as the domain ontology. He proceeds to define an application ontology, which we call *Artist Contract* (***AC***). Assume that he selects classes mo:SoloMusicArtist, mo:MusicGroup and mo:label, and the object property mo:member_of to create the ***AC*** vocabulary (all shown in Fig. 2).

The derivation of the ***AC*** constraints is not as straightforward as before since ***MO*** has no constraints involving just the terms in the ***AC*** vocabulary. However, observe from Fig. 2 that foaf:Person and foaf:Organization are disjoint classes. Therefore, the designer may infer that their subclasses, mo:SoloMusicArtist and mo:Label, respectively, are also disjoint. This constraint must therefore be in the set of ***AC*** constraints. Note that this constraint is inferred from, but not a member of the set of ***MO*** constraints.

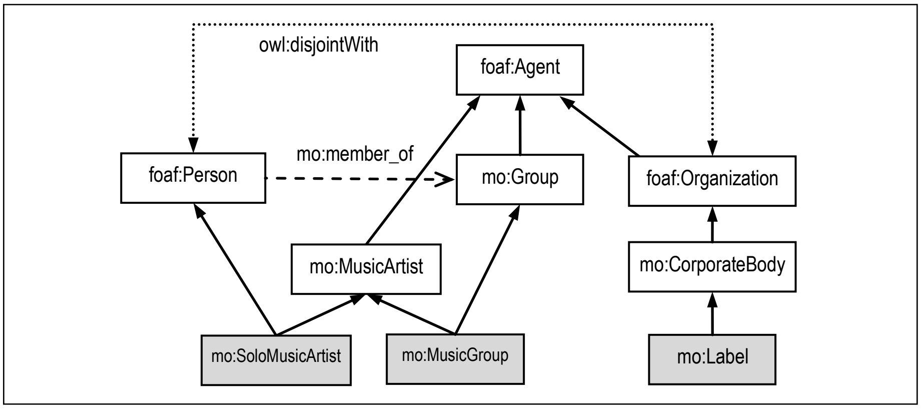


Fig. 2. The class hierarchies of ***MO*** rooted at classes foaf:Agent and foaf:Person

The question of including mo:member_of in the ***AC*** vocabulary, without including the original classes foaf:Person and mo:Group used to define its domain and range, raises a different question. One alternative is to ignore domain and range constraints when designing the ***AC*** constraints. A second alternative is to inform the designer that he should also include foaf:Person and mo:Group in the ***AC*** vocabulary. Sections 4 and 5 discuss these two alternatives in depth.

These informal examples illustrate that the domain ontology constraints play an essential role when designing the application ontology since they carry the semantics of the terms in the domain ontology vocabulary. They also raise the question that the design process cannot be reduced to merely copying the constraints from the domain ontology to the application ontology. We must take into account constraints derived from those of the domain ontology. This point is addressed in the rest of the paper.

3 A Formal Framework

3.1 A Brief Review of Attributive Languages

The reader may wish to skip this section on a first reading and go directly to Section 3.2 that introduces the notion of extralite ontologies and that has an informal description of the constraint semantics.

We adopt a family of *attributive languages* [1] defined as follows. A *language* $\mathcal{L}$ in the family is characterized by an *alphabet* $\boldsymbol{A}$, consisting of a set of *atomic concepts*, a set of *atomic roles*, the *universal concept* $\top$ and the *bottom concept* $\bot$. The set of *role descriptions* and the set of *concept descriptions* of $\mathcal{L}$ (or in $\boldsymbol{A}$) are defined as follows:

- An atomic concept, and the universal and bottom concepts are concept descriptions, and an atomic role is a role description

- If e and f are concept descriptions and p is a role description, then $\neg e$ (*negation*), $e \sqcup f$ (*union*), and $(\geq n\ p)$ (*at-least restriction*) are concept descriptions, and p^{-} (*inverse*) is a role description.

An *interpretation* $\mathbf{s}$ for $\mathbf{A}$ consists of a nonempty set Δ^s, the *domain* of $\mathbf{s}$, whose elements are called *individuals*, and an *interpretation function*, also denoted $\mathbf{s}$, where:

- $\mathbf{s}(\bot) = \emptyset$ and $\mathbf{s}(\top) = \Delta^s$
- $\mathbf{s}(A) \subseteq \Delta^s$, for each atomic concept A of $\mathbf{A}$
- $\mathbf{s}(P) \subseteq \Delta^s \times \Delta^s$, for each atomic role P of $\mathbf{A}$

The function $\mathbf{s}$ is extended to role and concept descriptions of $\mathcal{L}$ as follows:

- $\mathbf{s}(\neg e) = \Delta^s - \mathbf{s}(e)$ (the complement of $\mathbf{s}(e)$ w.r.t. Δ^s)
- $\mathbf{s}(e \sqcup f) = \mathbf{s}(e) \cup \mathbf{s}(f)$ (the union of $\mathbf{s}(e)$ and $\mathbf{s}(f)$)
- $\mathbf{s}(\geq n\ p) = \{I \in \Delta^s \,/\, |\{J \in \Delta^s \,/\, (I,J) \in \mathbf{s}(p)\}| \geq n\}$
  (the set of individuals that $\mathbf{s}(p)$ relates to at least n distinct individuals)
- $\mathbf{s}(p^{-}) = \mathbf{s}(p)^{-}$ (the inverse of $\mathbf{s}(p)$)

A *formula* of $\mathcal{L}$ (or in $\mathbf{A}$) is an expression of the form $u \sqsubseteq v$, called an *inclusion*, or of the form $u \equiv v$, called an *equivalence*, where u and v are both concept descriptions or they are both role descriptions of $\mathcal{L}$. A *definition* is an equivalence of the form $D \equiv e$, where D is an atomic concept and e is a concept description, or D is an atomic role and e is a role description.

Let $\mathbf{s}$ be an interpretation for $\mathbf{A}$, σ be a formula and Σ and Γ be sets of formulas of $\mathcal{L}$. We say that

- $\mathbf{s}$ *satisfies* $u \sqsubseteq v$ iff $\mathbf{s}(u) \subseteq \mathbf{s}(v)$, and $\mathbf{s}$ *satisfies* $u \equiv v$ iff $\mathbf{s}(u) = \mathbf{s}(v)$
- $\mathbf{s}$ is a *model* of Σ, denoted $\mathbf{s} \models \Sigma$, iff $\mathbf{s}$ satisfies all formulas in Σ
- Σ *logically implies* σ, denoted $\Sigma \models \sigma$, iff any model of Σ satisfies σ
- Σ *logically implies* Γ, denoted $\Sigma \models \Gamma$, iff any model of Σ is also a model of Γ

If $\mathbf{B}$ is a subset of $\mathbf{A}$, then $\Sigma/\mathbf{B}$ denotes the set of formulas σ that use only symbols in $\mathbf{B}$ and that are logically implied by Σ.

In the next sections we will also use the following abbreviations: "$e \sqcap f$" (*intersection*) for "$\neg(\neg e \sqcup \neg f)$", "$\exists p$" (*existential quantification*) for "$(\geq 1\ p)$", "$(\leq n\ p)$" (*at-most restriction*) for "$\neg(\geq n{+}1\ p)$" and "$u \mid v$" (*disjunction*) for "$u \sqsubseteq \neg v$".

3.2 Extralite Ontologies

We will work with *extralite ontologies* [7] that partially correspond to OWL Lite.

Definition 1

(a) A *strict extralite ontology* is a pair $O=(V_O, C_O)$ such that

(i) V_O is a finite alphabet, called the *vocabulary* of *O*, whose atomic concepts and atomic roles are called the *classes* and *properties* of *O*, respectively.
(ii) C_O is a set of formulas in V_O, called the *constraints* of *O*, which must be of one the forms shown in Fig. 3.
(iii) For each property *P* in V_O, there is at least one domain and one range constraint for *P* in C_O.

(b) A *non-strict extralite ontology* is a pair $O=(V_O, C_O)$ that satisfies only conditions (i) and (ii) above.

(c) An *extralite ontology* is either a strict or a non-strict extralite ontology. □

Fig. 3 introduces the constraint types allowed in extralite ontologies and informally defines their semantics, recalling that a class denotes a set of individuals and a property denotes a set of pairs of individuals. Fig. 3 also shows the *unabbreviated form of* a constraint. Note that a constraint and its unabbreviated form are equivalent. For example, the unabbreviated form of "$C \mid D$" is "$C \sqsubseteq \neg D$".

Finally, a *constraint expression* is an expression that may occur on the right- or left-hand sides of an unabbreviated constraint.

Constraint Type	**Formalization**	**Unabbreviated form**	**Informal semantics**
Domain Constraint	$\exists P \sqsubseteq D$	$(\geq 1\ P) \sqsubseteq D$	property *P* has class *D* as domain, that is, if *(a,b)* is a pair in *P*, then *a* is an individual in *D*
Range Constraint	$\exists P^{-} \sqsubseteq R$	$(\geq 1\ P^{-}) \sqsubseteq R$	property *P* has class *R* as range, that is, if *(a,b)* is a pair in *P*, then *b* is an individual in *R*
minCardinality Constraint	$C \sqsubseteq (\geq k\ P)$ or $C \sqsubseteq (\geq k\ P^{-})$		property *P* or its inverse P^{-} maps each individual in class *C* to at least *k* distinct individuals
maxCardinality Constraint	$C \sqsubseteq (\leq k\ P)$ or $C \sqsubseteq (\leq k\ P^{-})$	$C \sqsubseteq \neg(\geq k{+}1\ P)$ or $C \sqsubseteq \neg(\geq k{+}1\ P^{-})$	property *P* or its inverse P^{-} maps each individual in class *C* to at most *k* distinct individuals
Subset Constraint	$E \sqsubseteq F$		each individual in *E* is also in *F*, that is, class *E* denotes a subset of class *F*
Disjointness Constraint	$E \mid F$	$C \sqsubseteq \neg D$	no individual is in both *E* and *F*, that is, classes *E* and *F* are disjoint

Fig. 3. Extralite constraints

3.3 Constraint Graphs

The notion of concept graphs captures the structure of sets of constraints and is essential to the constraint construction methods of Sections 4 and 5. Again, the reader may wish to skip this section on a first reading and go directly to Section 3.4 that contains self-contained examples of constraint graphs.

We say that the *complement* of a non-negated expression *e* is $\neg e$, and vice-versa; the *complement* of $\bot$ is $\top$, and vice-versa. If *c* is an expression, then $\overline{c}$ denotes of complement of *c*. Let Σ be a set of unabbreviated constraints and Ω be a set of constraint expressions.

Definition 2. The labeled graph $g(\Sigma,\Omega)=(\gamma,\delta,\kappa)$ that *captures* Σ and Ω, where κ labels each node with an expression, is defined as follows:

(i) For each concept expression *e* that occurs on the right- or left-hand side of an inclusion in Σ, or that occurs in Ω, there is exactly one node in γ labeled with *e*. If necessary, the set of nodes is augmented with new nodes so that:
  (a) For each atomic concept *C*, there is one node in γ labeled with *C*.
  (b) For each atomic role *P*, there is one node in γ labeled with ($\geq 1\ P$) and one node labeled with ($\geq 1\ P^-$).

(ii) If there is a node in γ labeled with a concept expression *e*, then there must be exactly one node in γ labeled with $\bar{e}$.

(iii) For each inclusion $e \sqsubseteq f$ in Σ, there is an arc (M,N) in δ, where *M* and *N* are the nodes labeled with *e* and *f*, respectively.

(iv) If there are nodes *M* and *N* in γ labeled with ($\geq m\ p$) and ($\geq n\ p$), where *p* is either *P* or P^- and $m<n$, then there is an arc (N,M) in δ.

(v) If there is an arc (M,N) in δ, where *M* and *N* are the nodes labeled with *e* and *f* respectively, then there is an arc (K,L) in δ, where *K* and *L* are the nodes labeled with $\bar{f}$ and $\bar{e}$, respectively.

(vi) These are the only nodes and arcs of $g(\Sigma)$. □

Definition 3. The labeled graph $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ that *represents* Σ and Ω, where λ labels each node with a set of expressions, is defined from $g(\Sigma,\Omega)$ by collapsing each clique of $g(\Sigma,\Omega)$ into a single node labeled with the expressions that previously labeled the nodes in the clique. When Ω is the empty set, we simply write $G(\Sigma)$ and say that the graph *represents* Σ. □

If a node *K* of $G(\Sigma,\Omega)$ is labeled with an expression *e*, then $\overline{K}$ denotes the node labeled with $\bar{e}$ (which may be *K* itself). We use $K \rightarrow M$ to indicate that there is a path in $G(\Sigma,\Omega)$ from *K* to *M*.

Definition 4. Let $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ be the labeled graph that represents Σ and Ω. We say that a node *K* of $G(\Sigma,\Omega)$ is a *⊥-node with level n*, for a non-negative integer *n*, iff one of the following conditions holds:

(i) *K* is is a *⊥-node with level 0* iff
  a. *K* is labeled with ⊥, or
  b. there are nodes *M* and *N*, not necessarily distinct from *K*, and a non-negated concept expression *h* such that *M* and *N* are labeled with *h* and $\neg h$, and $K \rightarrow M$ and $K \rightarrow N$.

(ii) *K* is is a *⊥-node with level n+1* iff
  a. There is a ⊥-node *M* of level *n*, distinct from *K*, such that $K \rightarrow M$, and *M* is the ⊥-node with the smallest level such that $K \rightarrow M$, or
  b. *K* is labeled with a minCardinality constraint of the form ($\geq 1\ P$) (or of the form ($\geq 1\ P^-$)) and there is a ⊥-node *M* of level *n*, distinct from *K*, such that *M* is labeled with ($\geq 1\ P^-$) (or with ($\geq 1\ P$)), and *M* is the ⊥-node with the smallest level labeled with ($\geq 1\ P^-$) or ($\geq 1\ P$). □

Definition 5. Let $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ be the labeled graph that represents Σ and Ω. Let K be a node of $G(\Sigma,\Omega)$. We say that K is a $\bot$-*node* iff K is a $\bot$-node with level n, for some non-negative integer n. We also say that K is a $\top$-*node* iff $\overline{K}$ is a $\bot$-node. □

Finally, we introduce the **IMPLIES** procedure (in Fig. 4) to test logical implication for extralite ontologies, whose soundness and completeness is established in [7].

```
IMPLIES($\Sigma$, $e \sqsubseteq f$)
input:    a set $\Sigma$ of unabbreviated constraints and an unabbreviated constraint $e \sqsubseteq f$
output:   "YES - $\Sigma$ logically implies $e \sqsubseteq f$"
          "NO - $\Sigma$ does not logically imply $e \sqsubseteq f$"
begin  Construct $G(\Sigma, \{e,f\})$, the representation graph for $\Sigma$ and $\{e,f\}$;
       if  the node of $G(\Sigma, \{e,f\})$ labeled with $e$ is a $\bot$-node, or
           the node of $G(\Sigma, \{e,f\})$ labeled with $f$ is a $\top$-node, or
           there is a path in $G(\Sigma,\{e,f\})$ from the node labeled with $e$
                   to the node labeled with $f$,
         then return "YES - $\Sigma$ logically implies $e \sqsubseteq f$";
         else return "NO - $\Sigma$ does not logically imply $e \sqsubseteq f$";
  end
```

Fig. 4. The **IMPLIES** procedure

3.4 Examples of Extralite Ontologies

The following example illustrates the concepts introduced thus far, using those parts of the Music Ontology introduced in Section 2 (to save space, examples throughout the text use only some parts of the Music Ontology).

Example 1. Let ***AGL*** $= (V_{AGL}, C_{AGL})$ be the ontology that corresponds to the part of the Music Ontology shown in Fig. 2. Fig. 5 formalizes the set C_{AGL} of constraints and Fig. 6 depicts the graph $G(C_{AGL})$ that represents C_{AGL} (using unabbreviated constraints). Each constraint $e \sqsubseteq f$ in Fig. 5 corresponds to two arcs in Fig. 6: the arc from the node labeled with e to the node labeled with f, and the arc from the node labeled with $\bar{f}$ to the node labeled with $\bar{e}$ (where $\bar{c}$ denotes of complement of c). Note that, in the constraint graph of Fig. 6, there is a path from mo:Label to ¬mo:SoloMusicArtist, which indicate that C_{AGL} logically implies mo:Label ⊑ ¬mo:SoloMusicArtist. That is, C_{AGL} logically implies that these two classes are disjoint. □

Example 2. Let ***SGL*** $= (V_{SGL}, C_{SGL})$ be the ontology that corresponds to the mo:Signal class, its subclasses and properties, shown on the bottom-left of Fig. 1. Fig. 7 formalizes the set of constraints C_{SGL} and Fig. 8 contains the graph $G(C_{SGL})$ that represents C_{GGL} (using unabbreviated constraints). $G(C_{SGL})$ is constructed as the graph in Fig. 6. In particular, note that there is a path in $G(C_{SGL})$ from (≥2 mo:sampled_version) to

Constraint	Informal specification
(≥1 mo:member_of) ⊑ foaf:Person (≥1 mo:member_of⁻) ⊑ foaf:Group	The domain of mo:member_of is foaf:Person The range of mo:member_of is foaf:Group
mo:MusicArtist ⊑ foaf:Agent foaf:Group ⊑ foaf:Agent foaf:Organization ⊑ foaf:Agent mo:SoloMusicArtist ⊑ foaf:Person mo:SoloMusicArtist ⊑ mo:MusicArtist mo:MusicGroup ⊑ mo:MusicArtist mo:MusicGroup ⊑ foaf:Group mo:CorporateBody ⊑ foaf:Organization mo:Label ⊑ mo:CorporateBody	mo:MusicArtist is a subset of foaf:Agent foaf:Group is a subset of foaf:Agent foaf:Organization is a subset of foaf:Agent mo:SoloMusicArtist is a subset of foaf:Person mo:SoloMusicArtist is a subset of mo:MusicArtist mo:MusicGroup is a subset of mo:MusicArtist mo:MusicGroup is a subset of foaf:Group mo:CorporateBody is a subset of foaf:Organization mo:Label is a subset of mo:CorporateBody
foaf:Person ⊑ ¬foaf:Organization	foaf:Person and foaf:Organization are disjoint

Fig. 5. Constraints of ***AGL*** (unabbreviated form)

¬(≥2 mo:sampled_version). This means that C_{SGL} logically implies that (≥2 mo:sampled_version) ⊑ ¬(≥2 mo:sampled_version) or, equivalently, C_{SGL} logically implies that (≥2 mo:sampled_version) ⊑ ⊥. Intuitively, the set of individuals that mo:sampled_version maps to two or more individuals is empty, that is, mo:sampled_version is a functional property. Similar remarks apply to mo:isrc⁻, implying that mo:isrc is an inverse functional property (i.e., a key). □

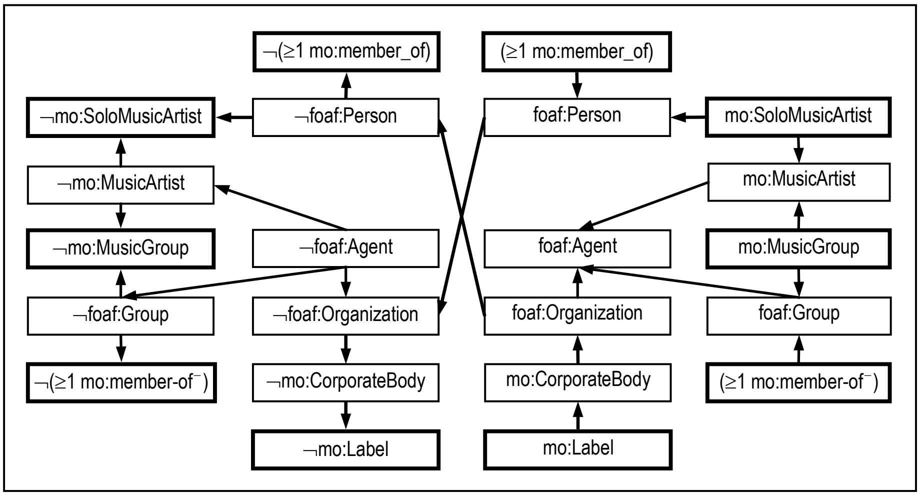


Fig. 6. The graph $G(C_{AGL})$ representing the constraints of ***AGL***

Constraint	Informal specification
(≥1 mo:sampled_version) ⊑ mo:AnalogSignal (≥1 mo:sampled_version⁻) ⊑ mo:DigitalSignal (≥1 mo:isrc) ⊑ mo:Signal (≥1 mo:isrc⁻) ⊑ xsd:String	The domain of mo:sampled_version is mo:AnalogSignal The range of mo:sampled_version is mo:DigitalSignal The domain of mo:isrc is mo:AnalogSignal The range of mo:isrc is mo:DigitalSignal
mo:AnalogSignal ⊑ ¬(≥2 mo:sampled_version) mo:String ⊑ ¬(≥2 mo:isrc⁻)	mo:sampled_version maps each individual in mo:AnalogSignal to at most one individual mo:isrc⁻ maps each individual in mo:String to at most one individual
mo:AnalogSignal ⊑ mo:Signal mo:DigitalSignal ⊑ mo:Signal	mo:AnalogSignal is a subset of mo:Signal mo:DigitalSignal is a subset of mo:Signal
mo:DigitalSignal ⊑ ¬ mo:AnalogSignal	mo:DigitalSignal and mo:AnalogSignal are disjoint

Fig. 7. Constraints of ***SGL*** (unabbreviated form)

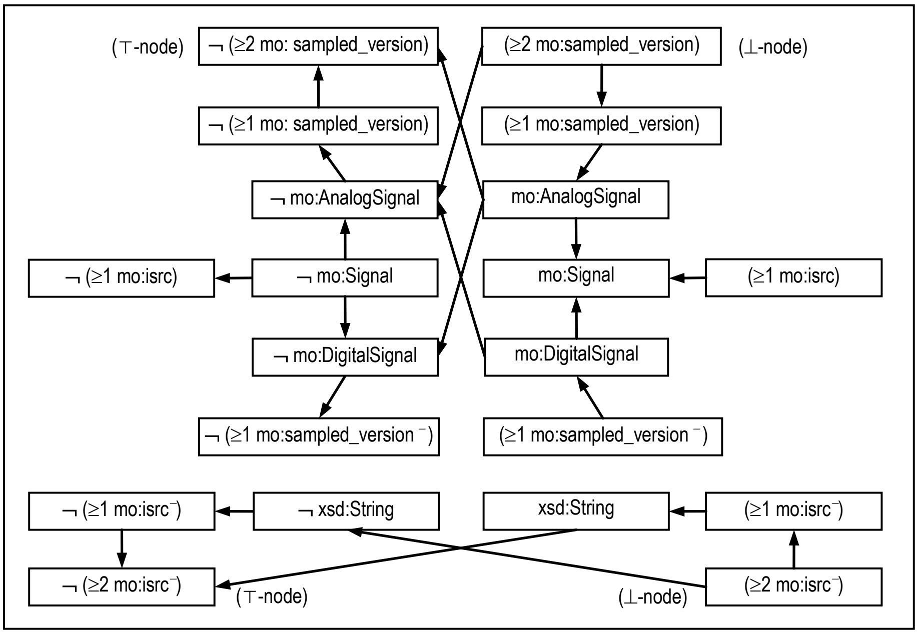


Fig. 8. The graph $G(C_{SGL})$ representing the constraints of ***SGL***

4 Open Fragments of Domain Ontologies

Let $\boldsymbol{D} = (V_D, C_D)$ denote the domain ontology, and $\boldsymbol{A} = (V_A, C_A)$ denote the application ontology. The design of the application ontology $\boldsymbol{A}$ depends on what requirements it must satisfy, discussed in detail in this and the next sections.

Recall that C_D / V_A denotes the set of formulas σ using only symbols in V_A such that C_D logically implies σ. Consider the following set of requirements:

R0. V_A is a subset of V_D
R1. C_A logically implies C_D / V_A
R2. Data exported by the data source satisfies C_A

An application ontology ***A*** that satisfies R0 and R1 is called an *open fragment* of ***D***. Requirement R0 guarantees that the data is exported using a subset of the vocabulary of the domain ontology. Requirements R1 and R2 indicate that the data published by the data source will be consistent with all constraints that can be derived from C_D and that use only symbols in V_A. Intuitively, Requirements R0, R1 and R2 imply that any Web application that processes data modeled according to ***D*** and *that uses only the classes and properties in* V_A will also be able to process the data published by the data source (since the application expects data consistent with C_D / V_A).

Assume that the designer has already created V_A by selecting symbols from V_D so that R0 is trivially satisfied. Procedure **OpenFragment** (in Figure 9) generates C_A so that R1 is satisfied, based on the representation graph of C_D. The procedure does not guarantee, however, that ***A*** $= (V_A, C_A)$ is a strict extralite ontology since it does not try to generate missing domain and range constraints.

```
OpenFragment($C_D$ , $V_A$ ; $C_A$)
input:    the set $C_D$ of normalized constraints of the domain ontology
          the vocabulary $V_A$ of the application ontology
output:   the set of constraints $C_A$ of the application ontology
begin  Initialize $C_A = \varnothing$;
       Construct $G(C_D)$, the representation graph for $C_D$;
       Mark all nodes of $G(C_D)$ labeled with expressions that use only
           atomic concepts and atomic roles in $V_A$;
       for each pair of nodes $M$ and $N$ of $G(C_D)$
           if   $M$ and $N$ are marked and there is a path from $M$ to $N$ in $G(C_D)$
           then add $e \sqsubseteq f$ to $C_A$ where
                   $e$ and $f$ are expressions that label nodes $M$ and $N$, respectively, and
                   $e$ and $f$ are expression of $V_A$, and
                   $e \sqsubseteq f$ is an allowed constraint (in the sense of Section 2), and
                   $\bar{f} \sqsubseteq \bar{e}$ is not already in $C_A$ /* to avoid redundant constraints */
       return $C_A$
end
```

Fig. 9. Procedure OpenFragment

OpenFragment is an almost direct variation of **IMPLIES**, introduced at the end of Section 3.3. It generates all constraints that involve only symbols in V_A and that are logical consequences of C_D. However, it avoids generating both $e \sqsubseteq f$ and $\bar{f} \sqsubseteq \bar{e}$, which are equivalent. We note that **OpenFragment** is non-deterministic since the set of constraints generated depends on the order that the for-loop selects pairs of nodes of $G(C_D)$, which is not unique.

The above argument can be generalized into a correctness proof of the **Open-Fragment** procedure, in the following sense:

Theorem 1. Let C_D be the set of unabbreviated constraints of the domain ontology and V_A be the vocabulary of the application ontology. Let C_A be the set of constraints which **OpenFragment** outputs for C_D and V_A. Then, C_A logically implies C_D/V_A. □

We close this section with an example that illustrates how the **OpenFragment** procedure operates.

Example 3. Assume that the domain ontology is ***AGL*** $= (V_{AGL},\ C_{AGL})$, introduced in Example 1, and that the designer wants to formally specify the *Artist Contract* application ontology ***AC*** $= (V_{AC},\ C_{AC})$, informally introduced in Section 2 as an open fragment of ***MO***. He starts by defining the vocabulary V_{AC} by selecting symbols from V_{AGL}:

(1) V_{AC} = { mo:SoloMusicArtist, mo:MusicGroup, mo:label, mo:member_of }

Then, **OpenFragment** generates the following set of constraints C_{AC} for ***AC***:

(2) mo:SoloMusicArtist $\sqsubseteq \neg$mo:Label

(3) mo:Label $\sqsubseteq \neg(\geq 1$ mo:member_of)

Recall that Fig. 6 shows $G(C_{AGL})$, the representation graph for C_{AGL}. To help follow this example, the thicker boxes in Fig. 6 indicate the marked nodes (that contain terms in V_{AC}) and the thicker lines indicate the paths between marked nodes.

Indeed, **OpenFragment** outputs:

- the constraint in (2) since there is a path from mo:SoloMusicArtist to $\neg$mo:Label, which implies that (2) is a logical consequence of C_{AGL}
- the constraint in (3) since there is a path from mo:Label to $\neg(\geq 1$ mo:member_of), which implies that (3) is a logical consequence of C_{AGL}. Note that this constraint was not anticipated in the informal analysis at the end of Section 2 since it is not an immediate, trivial consequence of the constraints in C_{AGL}

In fact, constraints (2) and (3) use only symbols in V_{AC} and they are logical consequences of C_{AGL} (albeit not necessarily in C_{AGL}). They also meet Requirement R1 by Theorem 1.

However, **OpenFragment** will not output, for example, the following formulas:

(4) $(\geq 1$ mo:member_of) $\sqsubseteq \neg$mo:Label

(5) mo:Label $\sqsubseteq \neg$mo:SoloMusicArtist

The procedure does not output (4) since this is not an allowed constraint, and it does not output (5) because (2) is already in C_{AGL}. However, since **OpenFragment** is non-deterministic, it could have returned (5) instead of (2). □

5 Closed Fragments of Domain Ontologies

Let ***D*** $= (V_D, C_D)$ be the domain ontology, and ***A*** $= (V_A, C_A)$ the application ontology. Let C_A^+ be the set of constraints C_A extended with new axioms of the form $C \sqsubseteq \perp$ (or $(\geq 1\ P) \sqsubseteq \perp$) that force each class C (or property P) in V_D, but not in V_A, to be the empty set. We now consider a different set of requirements:

R0. V_A is a subset of V_D

R1'. C_A^+ logically implies C_D

R2'. Data exported by the data source satisfies C_A^+

An application ontology ***A*** that satisfies R0 and R1' is called a *closed fragment* of ***D***. Requirement R0 again guarantees that the data is exported in a subset of the vocabulary of the domain ontology. Requirements R1' and R2' indicate that the data published by the data source satisfies C_D, when each class C (or property P) in V_D, but not in V_A, is taken to be the empty set.

```
ClosedFragment($C_D$ , $V_A$ ; $V_A$ , $C_A$, $S$)
input:    the set $C_D$ of normalized constraints of the domain ontology
          the vocabulary $V_A$ of the application ontology
output:   a new version of the vocabulary $V_A$ of the application ontology
          the set of constraints $C_A$ of the application ontology
          a set $S$ of suggested mapping definitions
begin  Initialize $C_A = \emptyset$ and $S = \emptyset$ ;
       Construct $G(C_D)$, the constraint graph for $C_D$;
       /* Stage 1: Analyze nodes of $G(C_D)$ that contain expressions in $V_A$ */
       Mark all nodes of $G(C_D)$ labeled with expressions that use only atomic concepts
           and atomic roles in $V_A$;
       Create a new graph $G_A$ by deleting any node $N$ from $G(C_D)$ such that
           $N$ is labeled with positive expressions and
               $N$ has no antecedent which is marked and labeled with a positive expression, or
           $N$ is labeled with negative expressions and
               $N$ has no descendent which is marked and labeled with a negative expression;
       /* Stage 2: Generate definitions for the classes and properties to be added to $V_A$ */
       for each node $N$ of $G_A$, in topological reverse order (i.e., from sinks to sources) do
           begin  if $N$ is labeled with a class $E$ not in $V_A$
                    then  add $E$ to $V_A$;
                          add "$E \equiv s_1 \sqcup \ldots \sqcup s_n$" to $S$, where $E$ labels a node $M$ and
                              $s_1, \ldots, s_n$ label nodes $M_1, \ldots, M_n$, and
                              $M_1, \ldots, M_n$ are all nodes such that $(M_k, M)$ is in $G_A$ ;
                  if $N$ is labeled with an expression involving a property $P$ not in $V_A$
                    then  add $P$ to $V_A$;
                          add "Skolemize[$P, G_A$]" to $S$ ;  /* (see explanation in the text) */
           end
       /* Stage 3: Generate the constraints of the application ontology */
       for each arc $(M,N)$ of $G_A$
           add  $e \sqsubseteq f$ to $C_A$ where
                $e$ and $f$ are expressions that label nodes $M$ and $N$, respectively, and
                $e$ and $f$ are expression of $V_A$, and
                $e \sqsubseteq f$ is an allowed constraint (in the sense of Section 2), and
                $\bar{f} \sqsubseteq \bar{e}$ is not already in $C_A$ ;          /* to avoid redundant constraints */
       return $V_A$, $C_A$, $S$
   end
```

Fig. 10. Procedure ClosedFragment

Intuitively, Requirements R0, R1' and R2' imply that any Web application that processes data modeled according to ***D*** will also be able to process data published by the data source, when each class C (or property P) in V_D, but not in V_A, is taken to be the empty set.

Assume that the designer has already created V_A by selecting symbols from V_D so that R0 is trivially satisfied. Procedure **ClosedFragment** (in Figure 10) extends V_A and creates C_A so that R1' is satisfied. Unlike **OpenFragment**, it guarantees that $\boldsymbol{A}=(V_A,C_A)$ is a strict extralite ontology since it generates domain and range constraints for all properties in V_A.

ClosedFragment has three stages. The first stage is preparatory for the next stages and analyzes which nodes of the constraint graph of C_D have expressions using only symbols in V_A.

The second stage includes a class E in V_A, if E is in V_D, but not in V_A, and there is an expression s_i using only symbols in V_A which the constraints in C_D force to be a non-empty subset of E. If this is the case, E cannot be forced to be always empty (by an axiom of the form $E \sqsubseteq \bot$). The solution is to include E in V_A and define E as the union of all expression s_i using only symbols in V_A such that there is a constraint of the form $s_i \sqsubseteq E$ which is a logical consequence of C_D. Note that, to be correct, this stage has to process nodes in topological reverse order.

The second stage also adds properties to V_A, if necessary. Let P be a property in V_D, but not in V_A. Assume that there is an expression s_i using only symbols in V_A which the constraints in C_D force to be a non-empty subset of an expression p involving property P. If this is the case, P cannot be forced to be always empty (by an axiom of the form $(\geq 1\ P) \sqsubseteq \bot$). The solution is to include P in V_A and define P as follows. Let $\boldsymbol{s}$ be the union of all expression s_i using only symbols in V_A such that there is a constraint of the form $s_i \sqsubseteq p$ which is a logical consequence of C_D. Then, P is defined as the set of all pairs of individuals (x,y) such that x is in the set denoted by $\boldsymbol{s}$ and y is one or more new individuals introduced until p is satisfied. We denote by *Skolemization[P,G]* such definition for P, and note that it is not expressible in the attributive languages of Section 3.1. The details of this construction and why it is always possible is outside the scope of this paper.

The third stage of **ClosedFragment** is a variation of **IMPLIES**. However, we note that this stage also automatically generates missing domain and range constraints. Indeed, assume that a property P is in V_A and that $(\geq 1\ P) \sqsubseteq E$ is the domain constraint for P in C_D. Then, the constraint graph for C_D will have an arc from the node labeled with $(\geq 1\ P)$ to the node labeled with E. Since P is in V_A, the first stage will then add E to V_A, if E is not already in V_A. Then, the third stage will add $(\geq 1\ P) \sqsubseteq E$ to C_A. An entirely similar argument applies to range constraints.

The above argument can be generalized into a correctness proof of the **Closed-Fragment** procedure, in the following sense:

Theorem 2. Let C_D be the set of unabbreviated constraints of the domain ontology and V_A be the original vocabulary of the application ontology. Let $\overline{V}_A$ and C_A be the vocabulary and the set of constraints that **ClosedFragment** outputs for C_D and V_A. Then, C_A^+ logically implies C_D, where C_A^+ is defined with respect to $\overline{V}_A$. □

We conclude this section with an example that illustrates how the **ClosedFragment** procedure operates.

Example 4. Let $\boldsymbol{ME} = (V_{ME}, C_{ME})$ be an ontology that formalizes the class hierarchies of the Music Ontology rooted at classes event:Event and frbr:Expression, informally introduced in Fig. 1 of Section 2. The reader may verify that $\boldsymbol{SGL} = (V_{SGL}, C_{SGL})$, the Signal ontology defined in Example 2, is an open fragment of ***ME***. The goal in this example is to redefine ***SGL*** so that it becomes a closed fragment of ***ME***, using the **ClosedFragment** procedure.

Recall from Example 2 that V_{SGL} is:

(1) V_{SGL} = { mo:DigitalSignal, mo:analogSignal, mo:Signal, xsd:String, mo:sampled_version, mo:isrc }

Let $\overline{V}_{SGL}$ be the new vocabulary, $\overline{C}_{SGL}$ be the set of constraints and S be the set of mapping definitions that **ClosedFragment** outputs, when given C_{ME} and V_{SGL} as input. To construct $\overline{V}_{SGL}$, $\overline{C}_{SGL}$ and S, **ClosedFragment** uses the constraint graph of C_{ME}, not shown here to save space (but the reader may observe Fig. 1 for an informal description of C_{ME}).

Then, $\overline{V}_{SGL}$ is the set:

(2) $\overline{V}_{SGL}$ = { mo:DigitalSignal, mo:analogSignal, mo:Signal, xsd:String, mo:sampled_version, mo:isrc, mo:MusicalExpression, mo:Expression }

$\overline{C}_{SGL}$ is shown in Fig. 11. Note that it includes two constraints not in C_{SGL}:

(3) mo:Signal $\sqsubseteq$ mo:MusicalExpression
(4) mo:MusicalExpression $\sqsubseteq$ mo:Expression

S contains the mapping definitions:

(5) mo:MusicalExpression $\equiv$ mo:Signal
(6) mo:Expression $\equiv$ mo:MusicalExpression

In particular, we observe that the mapping definitions in (5) and (6) force classes mo:MusicalExpression and mo:Expression to have the same set of individuals as mo:Signal. The definitions in (5) and (6) indeed indicate that the data exported will trivially satisfy the constraints in (3) and (4).

Finally, we note that, by definition of closed fragment, all classes and properties in V_{ME}, but not in $\overline{V}_{SGL}$, will be empty. □

($\geq$1 mo:sampled_version) $\sqsubseteq$ mo:AnalogSignal ($\geq$1 mo:sampled_version$^-$) $\sqsubseteq$ mo:DigitalSignal ($\geq$1 mo:isrc) $\sqsubseteq$ mo:Signal ($\geq$1 mo:isrc$^-$) $\sqsubseteq$ xsd:String	mo:String $\sqsubseteq \neg$ ($\geq$2 mo:isrc$^-$) mo:AnalogSignal $\sqsubseteq$ $\neg$ ($\geq$2 mo:sampled_version)	mo:AnalogSignal $\sqsubseteq$ mo:Signal mo:DigitalSignal $\sqsubseteq$ mo:Signal mo:DigitalSignal $\sqsubseteq \neg$ mo:AnalogSignal mo:Signal $\sqsubseteq$ mo:MusicalExpression mo:MusicalExpression $\sqsubseteq$ mo:Expression

Fig. 11. Constraints of ***SGL*** as a closed fragment of ***ME*** (unabbreviated form)

6 Related Work

The results reported in the paper cover a topic – the role that constraints play in the design of Linked Data – that is much neglected in the literature. The question of Linked Data semantics is not new, though. Recent investigation [11][12][15] in fact questions the correct use of `owl:sameAs` to inter-link datasets.

The results contribute to the discussion on the mapping process from relational databases (RDBs) to RDF [10]. Indeed, RDB-to-RDF tools (see [18] for a comprehensive survey) typically limit themselves to support vocabulary reuse, if at all. We argued that RDB-to-RDF tools should go further and include an analysis of the constraints of the domain ontology that apply to the data being published since such constraints capture the semantics of the reused terms. We introduced the notions of open and closed ontology fragments exactly to address this question. Such notions have no parallel in the published literature.

We note that the problem we address cannot be reduced to a question of ontology alignment in the context of Linked Data, addressed for example in [16][21]. Indeed, we stress that the problem we focus on refers to bootstrapping an application ontology (including constraints) as a fragment of a domain ontology.

The results in the paper also contribute to improving ontology browsing tools based on the idea of *focus+context* [20], where the notion of focus would be carried out by a vocabulary selection and the notion of context would be provided by the constraints. The methods to construct fragments of the domain ontology would act as a lens through which the user would browse the (large) domain ontology.

7 Conclusions

In this paper, we introduced automatic methods for constructing application ontology constraints, when the application ontology is an open or a closed fragment of the domain ontology. The final set of constraints will have useful properties, as detailed in Sections 4 and 5. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, based on constraint graphs.

The results in the paper are directly mapped to the RDF context and cover a topic – the role that constraints play in the design of Linked Data – that is much neglected in the literature.

As for current work, we are modifying an RDB-to-RDF tool [19] to generate application ontology constraints, as described in the paper. We are also extending the strategy to account for complex source-to-ontology mappings, using results from [13], to other types of constraints, using the development reported in [9].

Acknowledgements. This work was partly supported by CNPq, under grants 473110/2008-3 and 557128/2009-9, by FAPERJ under grant E-26/170028/2008, and by CAPES under grant NF 21/2009.

References

[1] Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuiness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation and Applications, pp. 43–95. Cambridge U. Press, Cambridge (2003)

[2] Berners-Lee, T.: Linked Data - Design Issues (2006), `http://www.w3.org/DesignIssues/LinkedData.html` (retrieved July 23, 2006)

[3] Berrueta, D., Phipps, J.: Best Practice Recipes for Publishing RDF Vocabularies - W3C Working Group Note, `http://www.w3.org/TR/swbp-vocab-pub/` (accessed June 14, 2009)

[4] Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web, `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/` (accessed June 14, 2009)

[5] Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)

[6] Brickley, D., Miller, L. FOAF Vocabulary Specification 0.98. Namespace Document August 9, 2010 - Marco Polo edn., latest version `http://xmlns.com/foaf/spec/(rdf, wiki)`

[7] Casanova, M.A., Lauschner, T., Leme, L.A.P.P., Breitman, K.K., Furtado, A.L., Vidal, V.M.P.: Revising the Constraints of Lightweight Mediated Schemas. Data & Knowledge Engineering 69, 1274–1301 (2010)

[8] Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P., Macêdo, J.A.F. The Role of Constraints in Linked Data. MCC21/11, Dept. Informatics, PUC-Rio (April 2011)

[9] Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P., Macêdo, J.A.F.: An Efficient Proof Procedure for a Family of Lightweight Database Schemas. In: Hinchey, M.G. (ed.) Conquering Complexity (to appear)

[10] Das, S., Sundara, S., Cyganiak, R.: R2rml: RDB to RDF mapping language. W3C RDB2RDF working group, `http://www.w3.org/TR/r2rml/` (accessed December 15, 2010)

[11] Halpin, H., Hayes, P.J.: When owl:sameAs isn't the same: An analysis of identity links on the semantic web. In: Proc. Int'l. Workshop on Linked Data on the Web (2010)

[12] Jaffri, A., Glaser, H., Millard, I.: URI disambiguation in the context of linked data. In: Proc. 1st Int'l. Workshop on Linked Data on the Web (2008)

[13] Lauschner, T., Casanova, M.A., Vidal, V.M.P., Macedo, J.A.F.: Efficient Decision Procedures for Query Containment and Related Problems. In: Proc. XXIV Brazilian Symposium on Databases (2009)

[14] Madison, O.: (Chair) Functional Requirements for Bibliographic Records - Final Report. IFLA Study Group on the Functional Requirements for Bibliographic Records (February 2009), `http://www.ifla.org/VII/s13/frbr/`

[15] McCusker, J., McGuinness, D.L.: owl: sameAs considered harmful to provenance. In: Proc. ISCB Conference on Semantics in Healthcare and Life Sciences (2010)

[16] Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z.: Ontology Alignment for Linked Open Data. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 402–417. Springer, Heidelberg (2010)

[17] Raimond, Y., Giasson, F.: Music Ontology Specification. Specification Document (November 28, 2010), latest version
`http://purl.org/ontology/mo/(RDF/XML, Turtle)`

[18] Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr., T., Auer, S., Sequeda J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. W3C RDB2RDF Incubator Group Report (2009)

[19] Salas, P.E., Breitman, K.K., Viterbo, J., Casanova, M.A.: Interoperability by Design Using the Std-Trip Tool: an a priori approach. In: Proc. 6th Int'l. Conf. on Semantic Systems (I-SEMANTICS 2010), Graz (2010)

[20] Villegas, A., Olivé, A.: A Method for Filtering Large Conceptual Schemas. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 247–260. Springer, Heidelberg (2010)

[21] Wang, Z., Zhang, X., Hou, L., Li, J.: RiMOM2: A Flexible Ontology Matching Framework. In: Proc. ACM WebSci 2011, Koblenz, Germany, pp. 1–2 (2011)

A Generic Approach for Combining Linguistic and Context Profile Metrics in Ontology Matching*

DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta

LIRMM, Univ. Montpellier 2
34392 Montpellier, France
firstname.name@lirmm.fr

Abstract. Ontology matching is needed in many application domains. In this paper, we present a machine learning approach for combining metrics, which exploits various linguistic and context profiles features in order to discover mappings between entities of different ontologies. Our approach has been implemented and the experimental results over Benchmark and Conference test cases on OAEI 2010 campaign[1] demonstrate its effectiveness and efficiency in terms of quality of matching and flexibility.

Keywords: Ontology matching, matcher combination, context profile, linguistic metrics.

1 Introduction

Numerous similarity metrics have been proposed so far for ontology mapping. According to [3], element metrics can be categorized in three groups: terminological, structural and semantic matching-based techniques. Metrics in the first group exploit text features such as name, labels and comments to calculate the similarity score between entities; whereas metrics of the last two groups exploit the hierarchy and semantic relations features.

Despite the fact that metrics in the first group are less semantic than that of the second and third, they are widely used in most of the matchers. During the matching process, mappings discovered by these metrics can be used as input mappings to other metrics of the second and third groups. Obviously, the more precise results terminological metrics are the more accurate results structural and semantic metrics have. Therefore, the aim of designing a high performance quality matcher exploiting terminological features becomes an importance task.

Due to the various types of heterogeneity of data sources, there is no single best metric overall matching scenarios. It is beneficial and necessary to combine several methods for improving matching quality. However, it is very difficult and time consuming even for experts to find a good combination. Therefore, the use

* Supported by ANR DataRing ANR-08-VERSO-007-04.
[1] http://oaei.ontologymatching.org

of supervised and machine learning approache is a promising way in order to reduce the required manual effort.

According to these necessity, we aim to build a high quality ontology matcher which utilizes machine learning approach to combine terminological similarity metrics. This matcher will be a premise for us to work in next step with structural and semantic matching methods.

The main contributions of this paper are the following. (i) We propose metrics dealing with terminological and context profile features of entities in ontology. (ii) We propose to use decision tree model to combine similarity metrics and strategies to select metrics and training data for the learning process. (iii) Experimental results performed on the benchmark and conference tests of OAEI 2010 campaign show that our system achieved stable and good results in comparison with other participants.

This paper is organized into 5 sections. Section 2 presents similarity metrics working with terminological and context profile features. Next, Section 3 contains our approach for combining similarity metrics. In Section 4 we describe the setting of experiments and show experimental results. Finally, we conclude and plan future work in Section 5.

2 Feature Extraction and Similarity Metrics

2.1 Similarity Metrics for Terminological Features

Terminological features of an entity consist of text information encoded in itself in ontology such as entities' URI(name space and local name), labels and comments. Terminological metrics can be categorized in two main groups: string-based and linguistic-based. String-based metrics take advantage of similar characters from two strings, whereas, linguistic-based metrics compare the meaning of strings.

Most of string-based metrics (e.g. Levenstein, SmithWaterman, JaroWikler, Qgrams, MongeElkan, etc.) are taken from open-source libraries SeconString[2] and SimMetric[3]. Additionally, we also implemented other string-based metrics such as Equality, Prefix, Suffix, Longest Common SubString [3] and Stoilois [8]. To deal with linguistic features, we implemented Lin, JiangConrath and Wu-Palmer [6] metrics working on WordNet[4] dictionary.

Because ontologies are designed by different people, consequently, names or labels indicating even to the same object or concept may be heterogeneous. For example, *"MscThesis"* and *"Ms.dissertation"* are different but they both indicate a master's thesis. Due to the heterogeneity of naming convention, primitive string-based or linguistic-based metrics mentioned above may be not sufficient. In order to deal with the heterogeneity problem, we perform analyses on terminological features of entities and propose solutions for each case.

[2] http://secondstring.sourceforge.net
[3] http://sourceforge.net/projects/simmetrics
[4] http://wordnet.princeton.edu

Firstly, labels and names of entities usually are compound of tokens. The whole strings may be not matched but their tokens may be highly similar. Therefore, a pre-processing procedure is needed to split a string into proper tokens. Afterward, tokens can be compared by primitive string and linguistic metrics.

Secondly, tokens may exist in various types of morphological forms of a word. To deal with this issue, we need a thesaurus or dictionary. In our system, we propose a generic algorithm to combine string and linguistic metrics at token level as follows:

In the Algorithm 1, function MorphologicalForms takes a token as input and finds all possible senses and morphological forms of token existing in Wordnet dictionary. For example, MorphologicalForms(*"published"*) returns { verb: *"publish"* , adjective: *"published"*}; MorphologicalForms(*"publishing"*) returns { noun: *"publishing"*, verb: *"publish"*}. Because two obtained sets of senses have a common {verb: *"publish"*}, therefore token *"published"* and token *"publishing"* are similar.

Algorithm 1. Compute similarity between two tokens

```
  Input: $token_1, token_2$ two tokens,
  $dictMetric$ a linguistic metric,
  $stringMetric$ a string metric
  Output: $score$ a numerical value
1 $MF_1 \leftarrow MorphologicalForms(token_1)$
2 $MF_2 \leftarrow MorphologicalForms(token_2)$
3 if $(MF_1 \neq \emptyset) \wedge (MF_2 \neq \emptyset)$ then
4 $score \leftarrow \max_{(pos_i, st_i) \in MF_1, (pos_j, st_j) \in MF_2}(dictMetric(st_i, st_j) \mid pos_i = pos_j)$
5 else $score \leftarrow stringMetric(token_1, token_2)$
```

Next, similarity score between two set of tokens is computed by two modifications of MongeEklan algortihms. One is proposed in [5] and the second is our proposal:

$$sim_m(a, b) = \frac{1}{|a|} \sum_{i=1}^{|a|} \left(sigmoid(max\{sim(a_i, b_j)\}_{j=1}^{|b|}) \right) \quad (1)$$

Here, $sigmoid(x) = \frac{1}{1+e^{-10\times(x-0.5)}}$ is a promoted function which makes the higher similar tokens is more informative than the lower ones. The idea and effectiveness of using promoted function can be seen in [5] for more detail.

Thirdly, name of entity may be an abbreviation (e.g. *"Misc."* instead *"Miscellaneous"*), an acronym (e.g. *"SW"* instead *"Semantic Web"*) or even a sequence of symbols which is not understandable. To deal with these cases, we expect that entities provide some human-readable labels in annotation information. In our approach, a local name is treated as a label of entity. The similarity measure between two entities based on their labels can be formulated as follows:

$$sim(e_i, e_j) = max_{(l_p \in labels(e_i), l_q \in labels(e_j))}(sim(l_p, l_q)) \quad (2)$$

For example, class **Chapter** in ontology #101 and class **dzqndbzq** in ontology #201[5] have the same label *"BookPart"*. Therefore two classes are matched.

An entity may also have several comments. They usually consist of a long descriptive text. Therefore, calculating similarity of two comments by comparing word by word is not a good choice. In our approach, we use comments in building text profile for each entity. Calculating similarity based on entities' profiles are explained in detail in the next section.

2.2 Similarity Metrics for Context Profile Features

In order to take advantage of relations information in ontology, we build variety types of text profile for each entity from its context. We divide context profiles of entities into three groups: IndividualProfile, SemanticProfile and ExternalProfile. Let us demonstrate how to build these profiles following a fragment of ontology in Fig.1

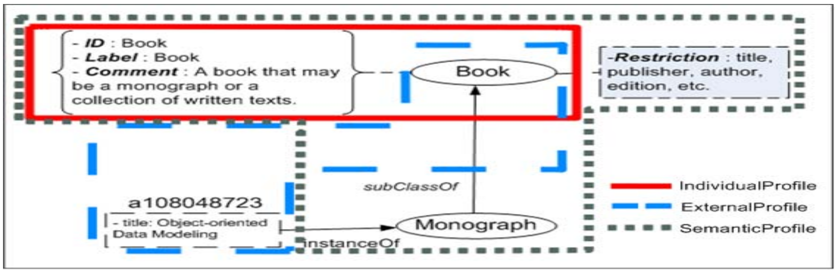

- ID : Book
- Label : Book
- Comment : A book that may be a monograph or a collection of written texts.
Book
-Restriction : title, publisher, author, edition, etc.
subClassOf
a108048723
- title: Object-oriented Data Modeling
Monograph
instanceOf
IndividualProfile
ExternalProfile
SemanticProfile

Fig. 1. Three types of context profile of class **Book**

The **IndividualProfile** of an entity is simply a string concatenation of its local name, labels and comments. *For example*: Individual profile of class **Book** is "Book Book A book that may be a monograph or a collection of written texts".

The **SemanticProfile** of an entity is an union of individual profiles of itself with individual profiles of its neighbors. Neighbors of a class consist of its sub-classes and all restricted properties. Neighbors of a property consist of its sub-properties, classes included in domain and range. *For example*: Semantic profile of class **Book** is created from individual profiles of {**Monograph, title, publisher, author, edition**}.

The **ExternalProfile** of an entity is created from texts taken from ontology instances. An external profile of a class is a string concatenation of texts of all instances belonging to either this class or its descendants. An external profile of a property is a string concatenation of value data corresponding to this property in all instances. *For example*: External profile of class **Book** is created from text value of instance **a108048723**. Therefore, external profile of class **Book** is "Object-oriented Data Modeling".

[5] http://oaei.ontologymatching.org

Having context profile for every entity, a similar technique described in [7] is used to compute similarity scores between entities. Let $sim_{IProfile}(e_i, e_j)$, $sim_{SProfile}(e_i, e_j)$ and $sim_{EProfile}(e_i, e_j)$ are similarity scores between entities (e_i, e_j) calculated by IndividualProfile, SemanticProfile and ExternalProfile respectively. To combine all of types of context profiles of entities, we propose the following generic formula:

$$sim(e_i, e_j) = \boldsymbol{f}(sim_{IProfile}(e_i, e_j), sim_{SProfile}(e_i, e_j), sim_{EProfile}(e_i, e_j)) \quad (3)$$

Where $\boldsymbol{f}$ may be *weighted average, max, etc.* If the combination function returns only similarity score achieved by SematicProfile, then our context profile metric is similar to metrics used in [1,7]. If the combination function returns only similarity score achieved by ExternalProfile, then our context profile is similar to the instance-based metric. This property makes our context profile metric more flexible.

3 Combining Similarity Metrics with Decision Tree Model

We have implemented a system named YAM++ - an extension of [2], which is based on decision tree model to combine our proposed similarity metrics above. In our approach, a decision tree is a tree whose non-leaf nodes are the similarity metrics, leaf nodes values are either 1.0 ore 0.0 indicating if there is a match or not. At a non-leaf node, a similarity value of to-be-matched entities is computed by the similarity metric in ongoing node. The returned value is compared with condition values on outgoing edges from current node in order to decide which child node will be reached. This process will start at root node and iterate until a leaf node is reached. The value of destination leaf node indicates whether the two entities should match or not. See [2] for more detail of the advantages of using decision tree model.

4 Experiments and Evaluations

4.1 Selection of Metrics and Training Data

In our system, similarity metrics are divided in three main groups: (i) **name metrics** exploit name feature; (ii) **label metrics** exploit label feature; (iii) **context metrics** exploit different types of context profiles. The selection of the most representative metrics for each group is based on the hypothesis "A good feature subset is one that contains features highly correlated with the class" [4]. The correlation value is calculated by Pearson's formula[6] between similarity scores obtained by a metric and values provided by experts for each test in Benchmark datasets. Finally, the similarity metrics having the highest average values in each of three groups above are selected.

[6] http://en.wikipedia.org/wiki/Correlation_and_dependence

Next, training data for learning process are selected from Benchmark datasets. It is based on our heuristic: a training data is representative with the respect to a feature if this feature is highly correlated to the class. For each test in Benchmark datasets, our system computes the average correlation coefficient for all selected metrics above. Then, tests having the highest average of correlation values are selected to build training data.

4.2 Experimental Evaluations

Two experiments were designed as follows: (i) The first experiment shows the effectiveness of our proposed metrics in different scenarios over Benchmark datasets. (ii) The second experiment presents the performance quality of our approach on Conference datasets and shows the comparison results with other participants on OAEI 2010 campaign.

Result on Benchmark datasets. According to terminological features described in Benchmark datasets, we select three representative groups of tests as follows: **(i) TestGroup1** contains various types of naming convention using in designing real ontologies. The typical selected tests are: **#104** (identical string), **#204** (different naming conventions) and **#205** (synonym words). **(ii) TestGroup2**: Names and labels of entities in test ontologies of this group are substituted by random meaningless strings. To discover mappings, we should take advantage from other features. We select tests **#201**, **#201-2**, **#201-4**, **#201-6** and **#201-8** for this group. Names and labels of entities in these tests are replaced by random strings with proportion **100%**, **20%**, **40%**, **60%** and **80%** respectively. However, they support annotation information and data instances for entities. **(iii) TestGroup3**: Test ontologies in this group are similar to ontologies in the second group except that they do not support annotation information for entities. We select tests **#202**, **#202-2**, **#202-4**, **#202-6** and **#202-8** for this group. Tests in the first group are suitable for name and label metrics. Tests in the second and the third groups are suitable for context metrics. In order to see how our metrics are appropriate to these scenarios above, we perform experiments on the following selected sets of similarity metrics: **(i) SimSet1**: Only name metrics; **(ii) SimSet2**: adding label metrics to **SimSet1**; **(iii) SimSet3**: adding Semantic context profile metric to **SimSet2**; **(iv) SimSet4**:

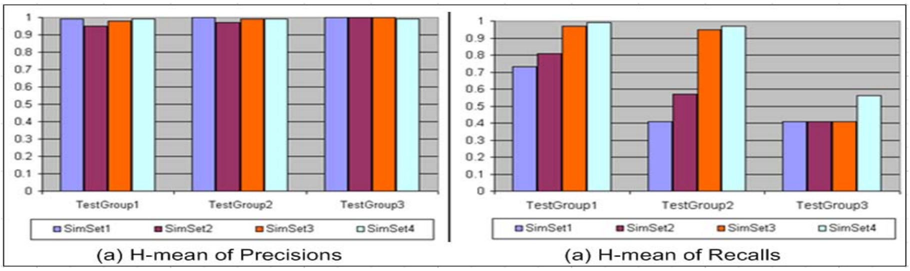


Fig. 2. H-mean of Precision and Recall on different scenarios

adding full context profile metric (Semantic and External context profiles) to **SimSet2**; After running 10 times with different selected training datasets, the harmonic mean values of Precision and Recall for each scenario are shown in Figure.2.

Generally, our system achieves very high precisions (≈**1.0**) in all scenarios (Fig.2a). This trend is exactly what we expected because our similarity metrics focus to high accuracy in calculating similarity scores between entities. Besides, the recall increases step by step thanks to adding metrics exploiting new features (Fig.2b). Now, we look at particular scenarios in detail.

In the **TestGroup1**, our string, linguistic and their combination metrics work well. Based on these metrics, our system discovered most of commonly real patterns used to name entities (e.g., synonym, different naming conventions). However, there are other real patterns requiring more knowledge of domain, for example (*"Academic"*, *"StudentReport"*), (*"LectureNotes"*, *"CourseMaterial"*). That is why using only string and linguistic metrics (in SimSet1 and SimSet2) our system obtained good recall (≈**0.8**) but not ideal (**1.0**). Thanks to context metrics (in SimSet3 and SimSet4), more mappings have been discovered.

In the **TestGroup2**, with name metrics only, our system achieves **0.41** for recall. Although this value is low but it is the expected number. Let see the description on test ontologies in this group. The average number of altered names of entities is (20% + 40% + 60% + 80% + 100%)/5 = 60%. It mean that the maximum number of mappings found by name metrics is 100% − 60% = 40%. This number is in line with the recall value (**41%**) obtained by our system. Similarly to TestGroup1, by adding new metrics (labels, contexts), our system discovered more mappings.

In the **TestGroup3**, recalls achieved by running with SimSet1, SimSet2 and SimSet3 are the same (**0.41**). That is because the test ontologies in this group do not support any annotation information. Only when running with SimSet4, thanks to ExternalContext profile, our system discovered more mappings.

Results on Conference datasets. In order to evaluate the performance of our approach with matching scenarios in another domain which is independent with training data, we select Conference datasets for testing. After running 10

Matcher	Confidence threshold	Precision	Recall	F-measure
AgrMaker	0.66	0.53	0.62	0.58
AROMA	0.49	0.36	0.49	0.42
ASMOV	0.22	0.57	**0.63**	0.60
CODI	*	**0.86**	0.48	**0.62**
Ef2Match	0.84	0.61	0.58	0.60
Falcon	0.87	0.74	0.49	0.59
GeRMeSMB	0.87	0.37	0.51	0.43
SOBOM	0.35	0.56	0.56	0.56
YAM++	*	**0.75**	**0.52**	**0.61**

Fig. 3. Optimal results of participants in Conference track

times with different selected training datasets, we obtain precision (**0.75**), recall (**0.52**) and f-measure (**0.61**) in harmonic mean. Fig.3 shows the performance quality of our system among participants in OAEI 2010 campaign.

Most of participants need to set a confidence threshold for finding mappings. Threshold values in the Fig.3 are found for the optimal f-measure value for matchers. Like CODI system, we do not need to set threshold to our system. We obtain the second position (under CODI) in term of harmonic mean F-measure among all participants in campaign OAEI 2010.

5 Conclusion and Future Work

In this paper, we have proposed new similarity metrics exploiting both terminological and context profile features. We also proposed a machine learning approach to combine these similarity metrics. Experiments over OAEI datasets show that our proposed metrics work effectively. Our system achieved high position among participants of OAEI 2010 campaign in Conference track. Additionally, our combining approach is automatic, flexible and extensible. In the future work, we plan to integrate structural and semantic methods to our system in order to improve its performance.

References

1. Cruz, I.F., Antonelli, F.P., Stroe, C.: Agreementmaker: Efficient matching for large real-world schemas and ontologies. Proceedings of The Vldb Endowment 2, 1586–1589 (2009)
2. Duchateau, F., Bellahsene, Z., Coletta, R.: A flexible approach for planning schema matching algorithms. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 249–264. Springer, Heidelberg (2008)
3. Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Heidelberg (2007)
4. Hall, M.A.: Correlation-based feature selection for discrete and numeric class machine learning. In: ICML, pp. 359–366 (2000)
5. Jimenez, S., Becerra, C., Gelbukh, A., Gonzalez, F.: Generalized Mongue-Elkan Method for Approximate Text String Comparison. In: Gelbukh, A. (ed.) CICLing 2009. LNCS, vol. 5449, pp. 559–570. Springer, Heidelberg (2009)
6. Lin, F., Sandkuhl, K.: A survey of exploiting wordnet in ontology matching. In: IFIP AI, pp. 341–350 (2008)
7. Qu, Y., Hu, W., Cheng, G.: Constructing virtual documents for ontology matching. World Wide Web Conference Series, pp. 23–31 (2006)
8. Stoilos, G., Stamou, G., Kollias, S.D.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 624–637. Springer, Heidelberg (2005)

ADERIS: An Adaptive Query Processor for Joining Federated SPARQL Endpoints

Steven Lynden, Isao Kojima, Akiyoshi Matono, and Yusuke Tanimura

Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST) Tsukuba, Japan
{steven.lynden,a.matono,yusuke.tanimura}@aist.go.jp, kojima@ni.aist.go.jp

Abstract. Integrating distributed RDF data is facilitated by Linked Data and shared ontologies, however joins over distributed SPARQL services can be costly, time consuming operations. This paper describes the design and implementation of ADERIS, a query processing system for efficiently joining data from multiple distributed SPARQL endpoints. ADERIS decomposes federated SPARQL queries into multiple source queries and integrates the results utilising two techniques: adaptive join reordering, for which a cost model is defined, and the optimisation of subsequent queries to data sources to retrieve further data. The benefit of the approach in terms of minimising response time is illustrated by sample queries containing common SPARQL join patterns.

1 Introduction

As individuals and organisations begin to publish RDF data using autonomous SPARQL services, more applications need to integrate RDF data from multiple SPARQL service endpoints using distributed query processing. Distributed query processing is facilitated by the adoption of common ontologies, for example as promoted by the Linking Open Data Project [1]. The problem of querying Semantic Web data is inherently a distributed one, as much of the data that needs to be accessed to answer such queries is held at different locations and is constantly updated. Issues related to scale, in terms of degree of distribution and data volume, require federated queries to access data sources directly rather than data warehouse-based solutions. This problem is well recognised, with federated extensions to the SPARQL query language in the pipeline for SPARQL 1.1 [2] and various distributed RDF query processing approaches.

In this paper, we demonstrate how adaptive query processing [3] can be used to address the problem of querying distributed RDF data utilising a mediator-based architecture, where it is assumed that a number of RDF data sources supporting SPARQL exist and the user requires a federated SPARQL query to be executed over these resources without specifying the details of how the query is executed, i.e. all optimisation is to be carried out automatically. Our approach presented in [4], which investigated the initial results of the application of adaptive query processing to joining data from RDF data sources is extended by a non-materialised intermediate storage that allows bound values to be adaptively passed to some data sources, reducing the amount of data retrieved when compared to materialising the full result set of a source query. Furthermore, the architecture the ADERIS query processor is described and an adaptive cost model is presented for optimisation query plans with the aim of adapting to service response times and minimising data transfer to improve performance and scalability.

2 Related Work

The Federated SPARQL extensions being developed for SPARQL 1.1 [2] include features such as the SERVICE construct, to match graph patterns with specific endpoints as part of a query, therefore providing support for distributed query processing, and a BINDINGS construct, which allows variable bindings to be specified in queries to control result sizes. A key difference in our approach is that instead of requiring explicit specification in the query which services are accessed using SERVICE/BINDINGS, our approach is completely declarative, the user does not have to be aware of which data can be obtained from which endpoint nor specify the details of how it should be retrieved. ADERIS adopts the concept of variable bindings from SPARQL 1.1 when accessing data sources, however to be compliant with existing SPARQL services they are translated into FILTER expressions. One feature supported by ADERIS is the optimisation of the number of bindings attached to source queries to reduce response times. This kind of optimisation is closely related to the optimisation of other aspects of inter-service communication such as data transfer rates and message sizes. Extremum control, among other approaches, has been investigated as a means of optimising block transfer for Web-services in [5].

ARQ [6] (the SPARQL query processor for the Jena [7] semantic web framework) is an example of an implementation providing support for the SERVICE construct, however, optimisation is generally left to the user or applications extending Jena, which must specify which services are queried with which triple patterns and the order in which they are joined. SPARQL-DQP [8], as an implementation of the SPARQL 1.1. Federation extensions, also utilises the SERVICE construct along with a set of re-write rules for optimisation.

ARQ is used to implement DARQ [9] (Distributed ARQ), an RDF distributed query processor supporting fully declarative SPARQL with automatic optimisation. DARQ optimises queries using statistics provided by service descriptions, a metadata format introduced in order to describe an RDF data source. The statistics used include information such as cardinalities and predicate selectivity values, and are utilised to generate a query plan, after which physical optimisation is implemented using iterative dynamic programming.

SemWIQ [10] is also implemented using ARQ, offering a similar approach to DARQ, supporting RDF distributed queries optimised using statistics about endpoints, in this case obtained by a monitoring component that sends SPARQL queries to data sources in order to generate statistics. SemWIQ uses various query optimisation strategies such as push-down of group patterns and join and union ordering, however, as is the case with DARQ, optimisation is carried out statically and requires comprehensive statistics about data sources.

A key difference when compared to much of the work described above is that ADERIS aims to be functional when only minimal information is known about the properties of the endpoints over which queries are executed. Although efforts are being made to encourage people to publish information about datasets that would be of aid to a distributed query processor, for example using the Vocabulary of Interlinked Datasets (VoID) [11] or SPARQL 1.1. Service Descriptions, it is not clear whether SPARQL service providers will use them to expose the kind of detailed histograms etc. required by a static query planner. In addition to the fact that services may behave unpredictably and service descriptions may be inaccurate or out-of-date, we believe adaptive approaches, such as the one described in this paper, can provide useful solutions.

3 The ADERIS Query Processor

The ADERIS system consists of a query processor that accepts a SPARQL query executed over a set of distributed RDF data sources by decomposing the initial query into a set of source queries that are sent to the individual RDF data sources. The results of the source queries are then integrated to provide an answer to the initial SPARQL query. An appropriate intermediate storage scheme is required to hold the results of the source queries, to which relational operators will be applied to produce a query result. The choice of storage scheme is influenced by the properties of SPARQL queries, for which, generally, the following observations made in [12] apply: the number of distinct predicate values is much less than the number of distinct subjects or objects, and in queries, predicates are usually specified as query constraints, whereas subjects and objects are more likely to be variables. A natural way to retrieve and process data from the individual RDF data sources is therefore to partition based on the predicate value of each triple, corresponding to the vertical partitioning scheme for RDF that has been investigated in [13], where each table is referred to as a predicate table.

When integrating data from SPARQL endpoints, optimising a static query plan is problematic, with join ordering being particularly difficult without information about join predicate selectivity values, table cardinalities etc. Work on adaptive query processing in relational database systems is directly applicable here, including recent work that has shown how join ordering can be changed in pipelined query execution plans during execution without having to duplicate the work performed by the plan before reordering took place. [14] presents a technique for reordering pipelined index nested loop (INL) join-based query plans during moments in which a sequence of joins is in a state whereby the join order can be changed without throwing away results that have already been produced. This approach allows the integration of the vertically partitioned RDF data to be done adaptively, as the tables are created. [4] presents in more detail the method used to achieve this in addition to some example query plans to further illustrate the approach.

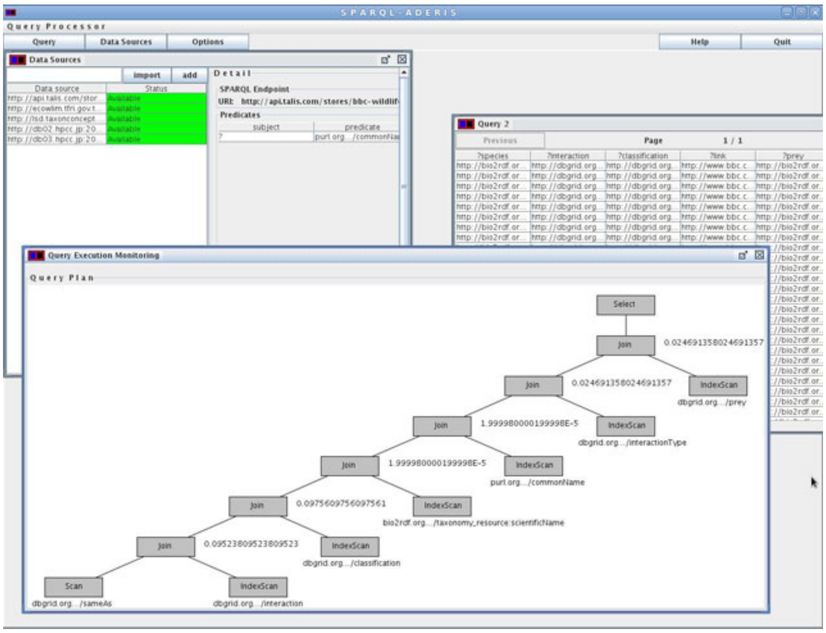


Fig. 1. ADERIS Graphical User Interface

ADERIS (Adaptive Distributed Endpoint RDF Integration System for SPARQL) is open-source software, available for download at:
`http://code.google.com/p/sparql-aderis`.

ADERIS provides a command-line, API and graphical user interface, which is illustrated in Figure 1. The software is currently being using in a scientific dataspace application [15] where multiple RDF data sources storing metadata about experiments must be integrated to provide scientists with up-to-date information about experiments being performed at multiple locations.

4 Retrieving Data from SPARQL Endpoints

ADERIS is primarily an adaptive approach that functions without statistics about the data in federated data sources, however a minimal set of information is required for basic static optimisations. This section describes how metadata is obtained from data sources to construct the source queries; the structure of the intermediate data structure (predicate tables) used to process source query results, and the decomposition of the federated query into source queries.

Given a federated SPARQL query over a set of data sources, it is necessary to generate a set of source queries to retrieve the required RDF data needed to answer the query. This is performed by mapping each triple pattern to a data source, combining triple patterns into the same query where possible, and pushing down predicates where possible. To do this efficiently, some information is required about the RDF triples contained within each data source, which should be constructed during an initialisation/configuration step taking place before any queries over the data sources can be executed. It is necessary to do this within the capabilities of the SPARQL query language rather than rely on out-of-band methods, e.g. as used in [9], therefore, the information assumed to be available is limited to presence/absence of predicate values within each specific data source. The set of distinct predicate values is usually straightforward to obtain via a single SPARQL SELECT query (using the DISTINCT function to select all unique predicate values), or alternatively a sequence of SPARQL ASK queries concerning specific predicate values (e.g. obtained from relevant ontologies or a specific query). Once the set of distinct predicate values is known, individual queries can use the COUNT function to retrieve the number of instances of each distinct predicate value. Failures to execute some queries do not prohibit the use of the corresponding endpoint during query processing, however less efficient optimisations are probable.

A predicate table is a two-column table used to store RDF triples with the same predicate value. The predicate value determines the table's name and the subject and object values of each triple determine the values of the attributes. Depending on the query execution strategy being used, an index may need to be constructed on predicate tables as they are formed using the results of source queries. Created predicate tables may support sequential and index-based access via the following operations:

1. `getNext(N)` - consume N tuples from the predicate table.
2. `hasNext()` - determine whether any more tuples can be consumed.
3. `subjectLookup(<values>)` - uses an index on the subject column to return all matching values for a set of input values.
4. `objectLookup(<values>)` - uses an index on the object column to return all matching values for a set of input values.

Operations 1 and 2 are used to support query processing based on a pipelined iterator model [16] (when N=1), which allows each operator to read tuples from its input one by one using `getNext` until `hasNext` returns false and the input is exhausted. Determining which indexes to build depends upon the possible joins that can be used to answer the query. Tables built without indexes do not support operations 3 and 4. For the INL join-based query execution plans used in this paper, at most one table can be constructed without an index, meaning that this table must be the left input of the first INL join. Having an index on all predicate tables is advantageous in some cases as it allows for the full space of possible query plans to be explored. On the other hand, constructing one table without an index also allows that predicate table to be incorporated into the query plan before it has been fully materialised.

Triple patterns present in some source queries may return too many results to be effectively processed in the manner described above. Furthermore, some

SPARQL endpoints may refuse to answer queries for which the estimated execution time is above a specific threshold. Where this is the case, instead of materialising the predicate tables that would be produced by such triple patterns, they are represented as non-materialised tables. A non-materialised table provides the `subjectLookup` and `objectLookup` operations but does not support `getNext` and `hasNext`, therefore it can only be used as the right input to an INL join. Non-materialised predicate tables implement `getNext` and `hasNext` by sending a query to the data source with bindings for the subject/object values. This is in essence a similar approach to the 'bind join' [17] approach as applied to distributed RDF query processing in [9], and is a nested loop join where value are passed in a sub-query to the other relation with join attribute values bound.

Source queries are compiled using metadata about data sources to assign triple patterns in the federated query to data sources from which they may be retrieved, as follows:

`1:` A set of source queries, S, are initialised.
`2:` All individual triple patterns are extracted from the federated query
`3:` The data sources in which each triple pattern can exist are determined (using the data source metadata). Triple patterns are assigned to the sources queries in S, using the SPARQL UNION operator to add the triple pattern to the corresponding source query.
`4:` FILTER predicates from the federated query are pushed down to the appropriate queries in S where possible.
`5:` Joins are pushed down to the source queries in S where possible.
`6:` The set of queries in S is examined and their cost estimated. Triple patterns with a high estimated cost are replaced by non-materialised predicate tables.

The pushing down of predicates and joins are standard techniques employed by many query processors, however the elimination of expensive triple patterns that takes place in the last step is necessary because of the complications involved in accessing SPARQL endpoints which may return too much data. The optimiser uses non-materialised predicate tables if the estimated result size from the source query is not known or estimated to be relatively large, likely resulting in inefficient query execution or failure of the endpoint to answer the query. If the result size cannot be estimated due to a lack of available information about a given data source, the optimiser will always attempt to use a non-materialised predicate table assuming no FILTER predicates can be pushed down to limit the expected result size. Naturally at least one source query must be issued (i.e. not all predicate tables can be non-materialised) so the optimiser ensures that this condition is not violated. Any source queries that fail (e.g. timeouts, refusal of the service to execute the query) are replaced by non-materialised tables. If no source queries are executable (i.e. all source queries are estimated to be too expensive in step 6) or the binding strategy results in queries that are not answered by any services, the federated query fails.

5 Optimisation

This section deals with the question of how to order joins, within the constraints of the approach previously presented, and how to optimise certain aspects of query execution, in particular when dealing with non-materialised predicate tables. Each time the optimiser is invoked, join ordering is performed based on a greedy algorithm that iteratively builds a left-deep query plan with the aim of minimising the estimated result cardinality when choosing each join, as described in [18]. In this work, all selectivity values are based on monitoring operators as the query is executed, and cardinalities are known (for materialised predicate tables) or roughly estimated for non-materialised tables. The join ordering algorithm estimates cardinality at each stage using the following function:

$$cardEst(t) = in\_card * card(t) * selectivity(t) \tag{1}$$

where in_card is the estimated input cardinality at each iteration; t denotes the table to be joined, with $card(t)$ being the cardinality of that table and $selectivity(t)$ being the monitored selectivity of the predicates applied when joining table t. Join ordering is achieved using the following cost function:

$$cost(t) = in\_card * lookupTime(t) + \sum_{i \in R} lookupTime(i) * cardEst(t) \tag{2}$$

where *lookupTime(t)* returns the average time taken to probe a given table and R denotes the remaining set of tables that need to be joined to the current plan. This cost function takes into account both cardinality, selectivity and the time taken to retrieve tuples using the indexes. Each input tuple needs to be used as a key to probe table t, therefore the cardinality of t is multiplied by the average lookup time per tuple for that table. Following this, the second part of the function estimates the consequence of the change in cardinality. The cost of the subsequent processing of the join result is roughly estimated by a summation of the join's result size multiplied by the lookup time of each of the remaining tables R that need to be joined with the current plan. The use of this rough estimate (independent of iterating possible subsequent join orders) corresponds to a greedy heuristic, providing an optimisation approach potentially scalable to a large number of joins.

Joins involving non-materialised predicate tables can be performed by passing multiple bound values within the same query, thereby reducing the overhead per tuple incurred when querying a SPARQL service. In order to achieve this, joins must consume multiple tuples before performing a lookup using the non-materialised table, therefore becoming blocking operators rather than fully pipelined. As a result of this, the query plan will enter states during which it can adapt less frequently. A potential trade-off therefore exists and the number of tuples consumed per iteration of each join with a non-materialised predicate table is something that can be potentially optimised to improve performance.

The optimal number of bindings to send in each query is dependent on various data source properties such as the number of triples in the data source and the SPARQL query processor implementation. To allow the query processor to adapt to the properties of the data source, a form of *extremum control* [19] is employed. In order to find the optimal number of bindings to send with each source query, it is assumed that there exists an unknown function $rt = f(n)$ where, rt is the source query response time per tuple sent to the data source and N is the number of bindings sent in the source query. Assuming that source queries are repeatedly issued in steps $x_k, x_{k+1}, ...$, an extremum controller attempts to find optimal values for N using a control function that determines the value of N at each step as follows:

$$N_k = N_{k-1} - g * sgn(\Delta rt_{k-1} * \Delta n_{k-1}) \tag{3}$$

where g is the gain, a configurable parameter used to control the rate at which N is explored ($g = 0.5$ and N is initialised to 1 in the results presented in this paper) and sgn returns 1 if the argument is positive and 0 otherwise. As join reordering requires a pipelined plan execution plan, when the query processor stops the plan to analyse whether join reordering is beneficial or not, N is reduced to 1 in all operators to force a state in which all operators have consumed their inputs, produced their outputs, and join reordering can take place. After the query processor analyses the plan in this state, and possibly reorders the joins, values of N can be resumed to their previous values.

6 Performance

Figure 2 illustrates some of the performance benefits that can be achieved by ADERIS when executing SPARQL queries involving joins over multiple endpoints. The figure shows an average of 10 warm runs for 4 different queries utilising two different strategies:

- no-adapt: No adaptivity allowed, the query planner makes static decisions on the join order and the number of bindings, N, is initialised to 1 and not changed during query execution.
- adapt: Adaptivity allowed, i.e. the full technique desribed in this paper is utilised. N is initialised to 1 but allowed to changed via extremum control.

SPARQL endpoints are implemented using the Jena framework's Joseki module (queries submitted via HTTP requests and results returned using XML) running on 3GHz Intel Xeon machines with 1MB available memory for each Java Virtual Machine providing the endpoint. 5 local data sources are deployed and connected to the query processor via a 100Mbs Ethernet LAN; additionally the SPARQL endpoint deployed by DBPedia is utilised in some of the queries. Figure 2 shows that a considerable reduction in response time can be achieved by the proposed technique.

```
dbo: <http://dbpedia.org/ontology/>
dbp: <http://dbpedia.org/property/>
owl: <http://www.w3.org/2002/07/owl#>
rdf:  <http://www.w3.org/2000/01/rdf-schema#>
skos: <http://www.w3.org/2004/02/skos/core#>
foaf: <http://xmlns.com/foaf/0.1/>

Query 1 (Result size = 150):
select * where {
 ?x dbp:reference ?ref .     777,679
 ?x rdf:comment ?comment .   10,000
 ?x skos:subject ?subj .     9971
 ?x foaf:page ?page .        10,000
 ?x rdf:type ?type .         800,000
 FILTER ( regex(str(?subj),"building") )
}
Query 2 (Result size = 8):
select * where {
 ?x dbp:reference ?ref .     777,679
 ?x rdf:comment ?comment .  10,000
 ?x skos:subject ?subj .    9971
 ?x foaf:page ?page .       10,000
 ?x rdf:type dbo:book       3105
}
Query 3 (Result size = 8):
select * where {
 ?x dbp:reference ?ref .     777,679
 ?x rdf:comment ?comment .  10,000
 ?x skos:subject ?subj .    9971
 ?x foaf:page ?page .       10,000
 ?x rdf:type dbo:book       3105
 ?x dbo:releaseDate ?date   (DBP) 126,737
}

Query 4 (Result size = 13):
select * {
 ?book rdf:type dbo:Book .  3105
 ?book foaf:page ?p .       10,000
 ?book owl:sameAs ?link     (DBP) 10,121,699
}
```

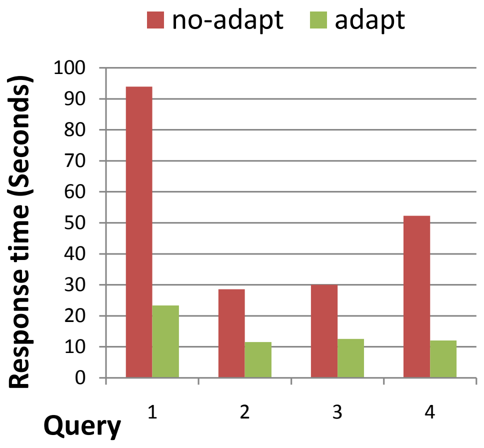


Fig. 2. This figure shows example performance benefits when executing queries over Joseki endpoints with subsets of the DBPedia data. The number of triples matching each triple pattern is annotated in bold-italics. In all cases, triples are fetched from endpoints we deployed except for where denoted by (DBP), corresponding to the actual DBPedia endpoint.

7 Conclusions and Future Work

An adaptive distributed query processor for joins over federated SPARQL queries has been presented. The challenging aspect of this work is that accessing remote SPARQL services complicates distributed query processing - detailed statistics and often unavailable, query response times may be unpredictable and many SPARQL services will not support queries for which the predicted execution time or result size is above a certain threshold. The work presented in this paper addresses these challenges by adaptively querying individual services and adaptively processing the results using a relational approach supporting join reordering. Advantages were shown by this approach over non-adaptive strategies both in terms of join reordering and the optimisation of the source queries issued to data sources.

Acknowledgment. This work is partly supported by the RENKEI project. (http://www.e-sciren.org/index-e.html)

References

1. Linked Data - Connect Distributed Data across the Web, http://linkeddata.org/
2. SPARQL 1.1 Federation Extensions, http://www.w3.org/2009/sparql/docs/fed/gen.html
3. Deshpande, A., Ives, Z.G., Raman, V.: Adaptive query processing. Foundations and Trends in Databases 1(1), 1–140 (2007)
4. Lynden, S., Kojima, I., Matono, A., Tanimura, Y.: Adaptive Integration of Distributed Semantic Web Data. In: Kikuchi, S., Sachdeva, S., Bhalla, S. (eds.) DNIS 2010. LNCS, vol. 5999, pp. 174–193. Springer, Heidelberg (2010)
5. Gounaris, A., Yfoulis, C., Sakellariou, R., Dikaiakos, M.D.: A control theoretical approach to self-optimizing block transfer in web service grids. TAAS 3(2) (2008)
6. ARQ SPARQL query processing framework, http://jena.sourceforge.net/ARQ/
7. Carroll, J.J., Dickinson, I., Dollin, C., Seaborne, A., Wilkinson, K., Reynolds, D., Reynolds, D.: Jena: Implementing the semantic web recommendations. Technical Report HPL-2003-146, Hewlett Packard Laboratories (2004)
8. Buil-Aranda, C., Arenas, M., Corcho, O.: Semantics and Optimization of the SPARQL 1.1 Federation Extension. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011. LNCS, vol. 6644, pp. 1–15. Springer, Heidelberg (2011)
9. Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
10. Langegger, A., Wöß, W., Blöchl, M.: A Semantic Web Middleware for Virtual Data Integration on the Web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 493–507. Springer, Heidelberg (2008)
11. Describing Linked Datasets with the VoID Vocabulary (W3C Interest Group Note March 03, 2011), http://www.w3.org/TR/void/
12. Tanimura, Y., Matono, A., Kojima, I., Sekiguchi, S.: Storage Scheme for Parallel RDF Database Processing Using Distributed File System and MapReduce. In: International Conference on High Performance Computing in the Asia Pacific Region (2009)
13. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. VLDB J. 18(2), 385–406 (2009)
14. Li, Q., Sha, M., Markl, V., Beyer, K., Colby, L., Lohman, G.: Adaptively Reordering Joins during Query Execution. In: Proc. ICDE, pp. 26–35. IEEE Computer Society, Los Alamitos (2007)
15. Elsayed, I., Brezany, P.: Towards Large-Scale Scientific Dataspaces for e-Science Applications. In: Yoshikawa, M., Meng, X., Yumoto, T., Ma, Q., Sun, L., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 6193, pp. 69–80. Springer, Heidelberg (2010)
16. Graefe, G.: Query evaluation techniques for large databases. ACM Comput. Surv. 25(2), 73–170 (1993)
17. Haas, L.M., Kossmann, D., Wimmers, E.L., Yang, J.: Optimizing queries across diverse data sources. In: 23rd Int. Conference on Very Large Data Bases, VLDB (1997)
18. Garcia-Molina, H., Widom, J., Ullman, J.D.: Database System Implementation. Prentice-Hall, Inc., Upper Saddle River (1999)
19. Astrom, K.J., Wittenmark, B.: Adaptive Control. Addison-Wesley (1995)

Asynchronous Replication for Evolutionary Database Development: A Design for the Experimental Assessment of a Novel Approach

Helves Humberto Domingues, Fabio Kon, and João Eduardo Ferreira

Department of Computer Science
University of São Paulo, Brazil
{helves,fabio.kon,jef}@ime.usp.br
http://www.ime.usp.br

Abstract. Environments with frequent changes in application requirements demand an evolutionary approach for database modeling. The challenge is greater when the database must support multiple applications simultaneously. An existing solution for database evolution is refactoring with a transition period. During this period, both the old and the new database schemas coexist and data is replicated in a synchronous process. This solution brings several difficulties, such as interference with the operation of applications. To minimize these difficulties, in this paper we present an asynchronous approach to keep these schemas updated. This paper presents the design for an experimental assessment of this novel approach for evolutionary database development.

Keywords: database evolution, asynchronous data replication, agile methods, performance evaluation

1 Introduction

In the conventional approach for database modeling [9], the complete conceptual, logical, and physical models must be developed before starting to write the application code. This method is often not effective to handle the complexity of many application domains and the speed of changing business requirements. An alternative is to use evolutionary modeling [1]: an iterative and incremental process to create the business data model. However, this modeling process generates the need to work with evolutionary databases [2,10,13] and each data modeling iteration has to deal with important production data, which must be preserved. The evolutionary database must take into account this legacy and allow controlled and organized changes with the data model.

There are studies that deal with schema evolution [4], not worrying about existing data in the database. There are others concerned with the evolution of applications that use this database [11]. When the database is object-oriented, we can find an extensive literature [15,14] on this subject. However, studies about evolutionary relational databases are rare. An exception is Ambler's book, *Refactoring Databases: Evolutionary Database Design* [3], which addresses evolutionary relational databases using refactorings - small changes to improve the internal structure without adding new functionality. Database refactorings have a transition period in which the old and new schemas coexist, allowing the updated applications and those still on maintenance, to work on the same database simultaneuously. To keep the data of the two schemas (new and old) it is necessary to develop a support code. This code, as proposed by Ambler, is a synchronous data replication process using triggers [9]. However, this synchronous process of updating schemas presents several difficulties. First, the programmer must write a specific trigger code for each refactoring; second, avoid ciclic sequences of triggers; third, deal with the increase in the transactions response time due to additional data replication code; and, finally, perform error handling.

Because of difficulties in writing the support code in the form of synchronous triggers, an alternative is to organize and structure the support code in three steps: collection, mapping, and execution. The first step is performed by triggers, which are used only to collect all transaction information. The data replication between the schema is performed by the execution step, following the mapping defined in the second step and using the data collected in the first. These three stages - collection, mapping, and execution - do all the work expected by the support code. We call synchronous replication the case in which these three steps are in the triggers, as proposed by Ambler. When only the data collection is performed with the application transaction and the other steps are performed later, we call it asynchronous replication. This new organization in three steps allows the development of refactoring tools and thus decreases the difficulties in evolving databases. This approach was presented by the authors, in Portuguese, in the Brazilian Symposium on Databases [8].

The main objective of the current paper is to propose the design of experiments to compare the current solution, synchronous replication, with the solution we proposed, asynchronous replication. We present the system description, the experiment hypothesis, the lab environment, and experimental scenarios.

Organization. First, Section 2 describes related works concerning refactoring and asynchronous replication. Second, Section 3 describes the transition period with the support code and, briefly, asynchronous replication on evolutionary databases. We then present the organization of the experimental performance evaluation (Section 4). Finally, we present our future works (Section 5).

2 Related Work

In this section, we discuss related work on refactoring and asynchronous replication that are necessary to understand our asynchronous replication approach.

Wiesmann et al. conceptually define five stages for the data replication protocol [18]:

1. **Request:** the client submits an operation to one or more replicas.
2. **Coordination of servers:** the servers, responsible for the replicas, communicate themselves to synchronize the transaction execution (ordering concurrent operations).

3. **Implementation:** the client-initiated operation is performed on the replica.
4. **Response:** the operation result is transmitted back to the client.
5. **Run result:** the servers communicate with each others to know the operation result.

According to Wiesmann et al., the protocols for data replication are different in the organization of these five phases. The asynchronous replication is similar to the delayed replication protocol, defined by Wiesmann et al. The advantage of this replication model is the low response time to customers, but the disadvantage is an eventual reconciliation process that must be executed to resolve conflicts. Comparing the asynchronous replication with this approach, we have three different points. First, the nomenclature: we use the term asynchronous model, not the *deferred* term. Second, we add the mapping phase to make the necessary changes in the transaction before updating the replica. Finally, we consider that the implementation phase is when the replicas are updated and the reconciliation task, if necessary, is executed.

In recent works, we can find frameworks to assist the schema evolution in temporal databases [6]. Curino et al. propose a framework named *Panta Rhei* designed to (i) provide database administration tools to facilitate the schema development, (ii) enable automatic old query rewrite to work in newer schemas, (iii) allow efficient historical data and metadata archiving, and (iv) allow complex temporal queries on such historical information. This framework is based on the work on the PRISM system [12] that defines the schema change operators. The asynchronous replication database has the same motivation reasons that Curino et al. have. Information systems with many and frequent changes in the database model need development tools to manage these changes. However, while the proposal by Curino et al. is based on automatic query rewrite to allow access to old schemas, our proposal uses asynchronous data replication during the transition period.

3 The Asynchronous Replication Approach

Due to limitations of the support code implementation in the form of synchronous triggers, an alternative is to reorganize the support code in three steps: collection, mapping, and execution. In this section, we briefly describe this approach that was presented by the authors in SBBD [8].

The **transition period**, when the old and new schemas coexist in the database, requires a support code to keep the two schemas synchronized. In the simplest case – when data changes occur only in the old schema, and the new one is used only for queries – we have a support code that must replicate all the changes from the old model to the new one. Thus, the code support allows applications not to be affected by the existence of two schemas – they always access updated data. At the end of the transition period, when all the applications that access the database have adapted to the new schema, the old schema and support code are destroyed. Because it is a code that will be used for a short period, this code is expected to be easy, fast, and simple to write.

The **collection step** is performed by a simple and generic trigger, called collector trigger. This trigger is used for all types of existing refactorings, with the goal of capturing all the transaction information. Captured information are the operation type (insert, delete, or update) and the previous and new values in the table. We do not use the database log to collect the transaction information because we want to use any database management system that has triggers. Database log files have different formats for each database management system.

The purpose of the **mapping step** is to define a mapping from the source to the target table. Each map represents one database refactoring that needs data replication.

In the **execution step**, the replication process is executed and the old and the new schemas are updated. The goal is to consume the information collected in the first step, following the mappings defined in the second one.

Figure 1 shows all asynchronous replication actions. The collector trigger writes the transaction information from the source table (1). The replication process reads the map (2), consumes the transaction information (3), and updates the target table (4).

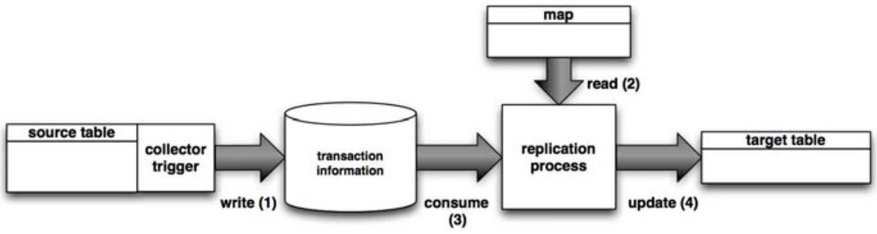


Fig. 1. Asynchronous replication approach

4 Proposed Experimental Assessment

To analyze the performance of the asynchronous alternative, we organized an experiment to compare it with Ambler's solution: the synchronous data replication for refactoring [3]. The comparison must be carried out by checking, at various levels of database concurrency, the performance of both solutions. Because asynchronous data replication is delayed, the time required to process a pending update must be measured to complete the assessment of the proposed approach.

According to Wohlin et al. [19], the technique that is appropriate for comparing the synchronous and asynchronous methods is the *engineering experiment*. This kind of experiment is defined as the method that observes the solutions, suggests the most appropriate solutions, develops, measures, analyzes, and repeats this process until no further improvement is possible.

4.1 System Description and Experiment Hypotheses

We designed the simplest and most representative environment to simulate one database refactoring for performing the comparison between the two approaches. For that, we used only one database table (`meetings`), which was the most complex one found in a specific system for the healthcare domain [7] developed by the authors. This table is the center of care modeling provided in public health centers and contains 19 attributes to store the necessary information.

The aim of the experiment is to validate the following hypotheses:

– **Hypothesis 1:** the synchronous method generates many locks and the amount is significantly greater than the one in the asynchronous method.
– **Hypothesis 2:** a system that updates the database with asynchronous replication has a better performance, measured in number of operations per second, than when using the synchronous method.
– **Hypothesis 3:** The time to process a pending operation in the asynchronous method is small and is on the order of tens of milliseconds.

Hypotheses 1 and 2 focuses on the comparison between synchronous and asynchronous methods in terms of performance. The first one aims at analyzing the amount of locks while the second considers the number of operations per second. Hypothesis 3 deals with the period of inconsistency in the database table. There will be no more inconsistency only after the execution of the replication process. The average processing time of a pending operation will provide an estimate of this inconsistency period.

4.2 The Laboratory Environment

In this section, we describe the virtual machine and the real machine created to be the testbed environment that is as close as possible to a real environment. The main elements of this laboratory environment are shown in Figure 2. The use of a virtual machine in the lab environment is very useful since we can easily manage all the resources of this machine. Below, we describe all parameters of our lab environment.

Virtual machine - The virtual machine was created with the software *VMware Fusion* [17], version 3.1.2, and has the following elements:

1. **Resources:** contains 1GB memory, 500GB hard disk with one of two processor cores of Core 2 Duo 2.8Ghz.
2. **Operating system:** uses GNU/Linux distribution, Ubuntu 9.10 Server 64-bit standard installation.
3. **Database manager:** PostgreSQL version 8.4, with a standard installation. The only changed parameter was the maximum amount of connections. We set the value to 200, due to the amount of virtual machine memory.

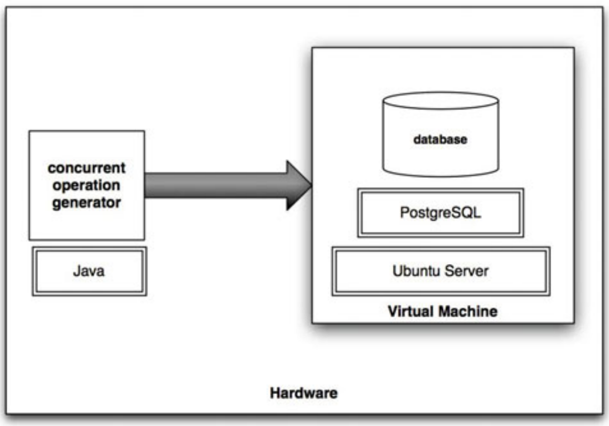


Fig. 2. The lab environment

Real machine - The hardware is a MacBook Pro with the following elements:

1. **Hardware:** contains 7GB of memory, 20GB hard disk with Core 2 Duo 2.8Ghz.
2. **Operating system:** Mac OS X version 10.6.4 64 bit.
3. **Concurrent operation generator:** the *Bristlecone Performance Test* [5] tool was used to generate a large volume of concurrent operations in the database.
4. **Java Virtual Machine:** the JVM used is the *SE Runtime Environment 1.6.0.22* [16]. Basically, the JVM was used to run the *Bristlecone Performance Test* tool. All client processes that generate load on the environment are actually Java threads.

Throughout the experiment, both the real machine and the virtual machine will be monitored by operating system tools (*Activity Monitor* for Mac OS X and *top* for GNU/Linux) to check for a lack of resources that could compromise the experiment.

4.3 Scenarios

In this section, we present three experiment scenarios. All scenarios have the same concurrent operation generator process. What differentiates each scenario is the existence of triggers and their type: synchronous or asynchronous.

We set up two tables (`meetings_0` and `meetings_1`) with the same structure but with different names and will perform data replication between them. The `meetings_0` table will be the source and `meetings_1` table, the target.

Below, we have the Figure 3 and the description of each scenario.

- **Scenario 1 - no trigger**: the two meetings tables do not have data replication, ie, there is no trigger associated with the two tables. This scenario aims to be the baseline.

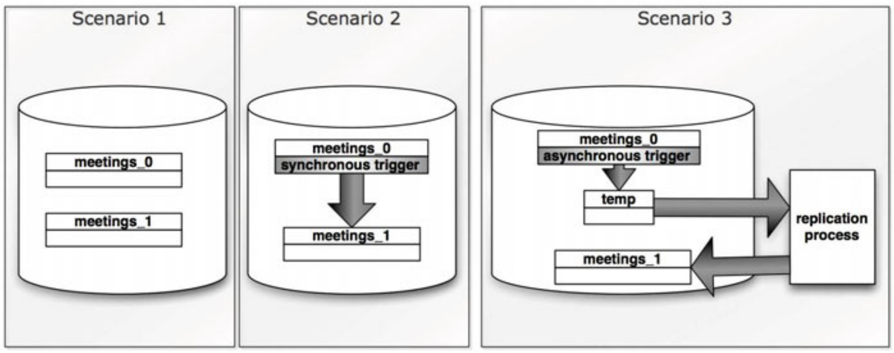


Fig. 3. Scenarios

- **Scenario 2 - synchronous trigger**: The purpose of this scenario is to evaluate the performance of the current solution for database refactorings. This trigger is as simple and efficient as possible: it only performs the update to `meetings_1` table from the `meetings_0` table.
- **Scenario 3 - asynchronous trigger**: The purpose of this scenario is to evaluate the asynchronous replication method. The asynchronous trigger records data operations into a temporary table (`temp`) and the replication process runs simultaneously with the concurrent operation generator process.

5 Future Works

Right after the first entry into database production, the user starts storing and updating the data of its business. Then, for each change in the database, it is necessary to address the existing data. The challenge to evolve one database is even greater when the database is used by multiple applications simultaneously. Each application will have different problems and will spend different times to make the necessary adjustments. A transition period with the old and new coexisting schemas is very useful, but requires a support code to maintain the schemas updated.

The main objective of this paper was to organize an experimental performance comparison between the current solution, synchronous replication, and the solution we proposed, asynchronous replication. We described the experiment objectives, the lab environment, and the scenarios that define the structure of the experiment. In a future paper we will describe the experiment execution and present its results, the data analysis, and the interpretation of experimental results.

References

1. Ambler, S.W.: Agile modeling: effective practices for eXtreme programming and the unified process. John Wiley and Sons (2002)
2. Ambler, S.W.: Agile Database Techniques. John Wiley & Sons Inc. (2003)

3. Ambler, S.W., Sadalage, P.J.: Refactoring Databases: Evolutionary Database Design (Addison Wesley Signature Series). Addison-Wesley Professional (2006)
4. Balsters, H., de Brock, B., Conrad, S. (eds.): Database Schema Evolution and Meta-Modeling. Springer, Heidelberg (2001)
5. Continuent: Bristlecone performance test, http://www.continuent.com/community/lab-projects/bristlecone (last access on February 18, 2011)
6. Curino, C., Moon, H.J., Zaniolo, C.: Managing the history of metadata in support for db archiving and schema evolution. In: Fifth International Workshop on Evolution and Change in Data Management, pp. 78–88. Springer, Heidelberg (2008)
7. Domingues, H., Correia, R., Kon, F., Kon, R., Ferreira, J.E.: Análise e modelagem conceitual de um sistema de prontuário eletrônico para centros de saúde. In: SBC - Workshop de Informática Médica, pp. 31–40. Belém, Brasil (2008)
8. Domingues, H., Kon, F., Ferreira, J.E.: Replicação assíncrona em modelagem evolutiva de banco de dados. In: SBBD - XXIV Simpósio Brasileiro de Banco de Dados (Brazilian Symposium on Databases), pp. 121–135. Ceará, Brazil (2009)
9. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, quinta edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
10. Fowler, M.: Evolutionary database design (2003), http://www.martinfowler.com/articles/evodb.html (last access on February 18, 2011)
11. Hick, J.-M., Hainaut, J.-L.: Database application evolution: A transformational approach. Data & Knowledge Engineering 59(3), 534–558 (2006)
12. Moon, H.J., Curino, C.A., Deutsch, A., Hou, C.-Y., Zaniolo, C.: Managing and querying transaction-time databases under schema evolution. Proceedings of the VLDB Endowment 1(1), 882–895 (2008)
13. Oertly, F., Schiller, G.: Evolutionary database design. In: Fifth International Conference on Data Engineering, pp. 618–624 (1989)
14. Rahm, E., Bernstein, P.A.: An online bibliography on schema evolution. SIGMOD Rec. 35, 30–31 (2006), http://doi.acm.org/10.1145/1228268.1228273
15. Rashid, A., Sawyer, P.: A database evolution taxonomy for object-oriented databases. Journal of Software Maintenance and Evolution: Research and Practice 17(2), 93–141 (2005)
16. Support, A.: Java for mac os x 10.5 update 8. http://support.apple.com/kb/DL971 (last access on February 18, 2011)
17. VMware: Desktop products, vmware fusion 3, http://www.vmware.com/products/fusion (last access on February 18, 2011)
18. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding replication in databases and distributed systems. In: Proceedings of ICDCS 2000, pp. 264–274. IEEE Computer Society (2000)
19. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: An Introduction. Kluwer (2000)

Improving the Accuracy of Ontology Alignment through Ensemble Fuzzy Clustering

Nafisa Afrin Chowdhury and Dejing Dou

Department of Computer and Information Science,
University of Oregon, Eugene, OR 97403, USA
{nafisa,dou}@cs.uoregon.edu

Abstract. Automatic ontology alignment tools perform matching between the concepts of two ontologies and provide a similarity measure for each pair of aligned concepts. However, none of the existing tools are perfect and multiple alignment tools produce varying similarity measures for a certain alignment. Also, the similarity measures provided by an alignment may not be helpful enough for indicating the degree of reliability. While using a random alignment tool we noticed that some quality alignments are given medium or even low similarity measures, and that causes the user ignoring those alignments. In this study we have proposed an ensemble model of ontology alignment that aggregates multiple alignment tools with the help of Fuzzy C Means clustering and Type 2 Fuzzy Membership Functions. We have shown that our approach helps the user to choose the best alignment results which has not been obtained by any other alignment tools we experimented with.

Keywords: ontology alignment, uncertainty, Fuzzy C Means clustering (FCM), Interval Type 2 Fuzzy Membership Function, Ensemble Fuzzy Clustering.

1 Introduction

Automatic ontology alignment tools reduce the tedious manual job of establishing the correspondences between two ontologies [1]. Most of the available automatic tools provide a similarity metric (generally 0.0 - 1.0) associated with the alignment based on the algorithm being used. The problem arises when a user is looking for quality alignments in order to use in his application. It is expected that an alignment with low similarity measure is not as helpful as an alignment with higher value. However, there is no absolute threshold value that can distinguish quality and poor alignments. We observed that the automatic alignment tool Falcon AO [2] that uses an integrated alignment algorithm of linguistic and graphical matcher, suffers this problem. When we applied two ontologies of camera domain we found only nine alignments with similarity measure more than 0.7, as there was a default threshold value 0.7. However, when we changed the threshold value to 0.1 in the source code, we found several other quality alignments even though they had lower similarity measure. Having the discrete threshold of 0.7 obscured some of the alignments such as “lens”-“lenGroup”, “PurchaseableItem”- “PurchasebleItem” etc. In our ensemble ontology alignment model we considered threshold value zero in order to obtain all possible alignments from an alignment tool.

Another problem is, the alignment tools using different similarity algorithms almost never agree to each other and this causes ambiguity of selecting the “appropriate” tool. We observed running a pair of bibliographic ontologies in three ontology alignment tools: Falcon AO [2], Anchor Flood [4] and OLA [5]. We found most of the alignments received three different similarity measure values from them. For example, an alignment “hasInstitution”-“institution” received 0.97, 1.0 and 0.33 from those tools respectively. Apparently this alignment is a strong one according to first two of the tools but weak based on the judgment of third the tool. Now, the question arises which similarity value is correct?

In order to address the issue of deciding the quality of an alignment our Ensemble Fuzzy Clustering approach incorporated fuzzy membership functions. Instead of specifying a discrete boundary of “quality” and “poor” alignments, fuzzy membership functions specifies the possibility of an alignment to be a member of certain class [3]. In most cases fuzzy membership functions are established by human experts. However, as it is expensive to have human experts we built membership functions automatically with the help of Fuzzy C Means (FCM) clustering [7]. The reason of choosing FCM is that it provides us with overlapping class boundaries, which is more likely a fuzzy membership function. When we obtained membership functions automatically from multiple alignment tools, we aggregated the results using Interval Type 2 Fuzzy Membership Functions (IT2MF) [3]. IT2MF is described as a blurred type 1 fuzzy membership function [3], where the degree of membership function will be represented as a range of available membership grades.

The rest of this paper is organized as follows: we will study several other research publications that expected to illuminate in related areas(section 2). In section 3 we illustrated our own ensemble model of choosing the best alignments in details. The section 4 demonstrates our experiments in two cases(bibliographic text and family). In the last section we discussed our contributions in this paper and future works.

2 Related Works

In order to address the problem of alignment tool selection Eckert et al [6] developed a framework using machine learning technique. This work was similar to ours in the sense that it also tried to aggregate the results of the alignment tools instead of using a single. The major difference with our approach is that, in this work the authors used discrete values to evaluate an alignment and we used floating point similarity measure values. Also, unlike our clustering the authors of this paper assigned human specified class labels along with the alignments. However, this causes the same problem of discrete threshold values as we described in the introduction.

There are not many works had been accomplished that addressed uncertainty in ontology alignment using fuzzy logic. Among the very few Niwattanakul et al proposed an approach to classify an ontology similarity based on fuzzy membership function [8]. Here the authors divided the alignments into some classes between most similar to least similar. In our approach we avoided using classification technique as our alignments did not have class labels.

Another work with slightly different objective proposed by Tordai et al studied longer chains of mappings when more than two ontologies are aligned [10]. This paper also tried to divide the alignments into several groups based on their similarity. The authors used sampling methods for evaluating the alignments and assign them into matching groups by human experts. Our approach outperforms in this regard as there is no human expert intervention required. We used a Fuzzy based clustering method that generates the grouping in membership function automatically.

3 Methodology

In our Ensemble Fuzzy Clustering Approach, we hypothesized a fuzzy membership function (MF) that will address the degree of uncertainty for all resultant alignments. Here an alignment will have a membership value for each of the classes and all the membership values of a particular alignment will be added up to one, which indicates the valid existence of the alignment. Figure 1 portrays our model of Ensemble Fuzzy Clustering.

In order to build MFs automatically clustering was more promising than classification methods, as there was no class labels associated with the alignments. Also, MFs require fuzzy or overlapping class boundaries which is unavailable in generic clustering methods (like K-means). We found Fuzzy C Means (FCM) clustering [7] [11] provides overlapping class boundaries where a data instance can belong to more than one clusters. We applied FCM on the similarity measure values provided by the alignment tools and, obtained the membership values for each of the alignments. We also obtained the center values of those clusters. Figure 1 shows how the membership functions are being generated from n alignment tools. If there are three clusters, we can easily distinguish the cluster with highest center value as the cluster of most quality alignments.

In figure 1 there are p membership functions generated from each of the tools. For a particular cluster we can combine the n MFs generated from n tools by applying the Interval Type 2 Fuzzy Logic theory. In our approach, for a particular cluster we constructed an IT2MF, where the membership degree interval of a single alignment was the interval of its observed maximum and minimum membership values. We will see examples of constructing IT2MF in the case studies section.

Now the question is how we can infer the best alignments from the IT2MF we developed in Fig 1. An alignment having longer interval denotes that the alignment tools did not agree upon its strength. So the alignment that has longer interval is more uncertain than the alignment with shorter interval. This intuition

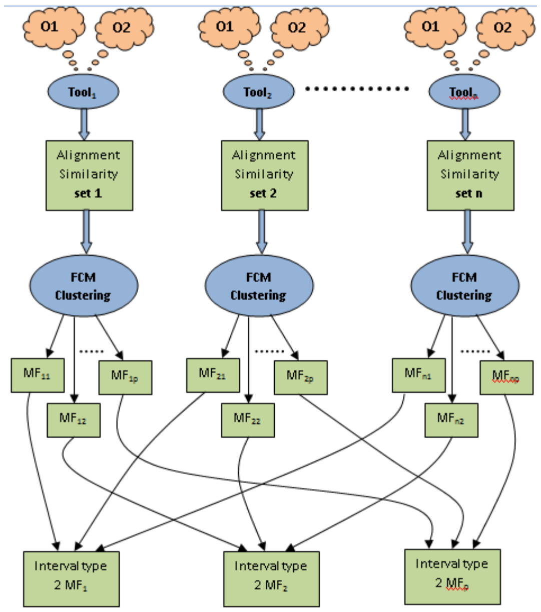


Fig. 1. Generating membership functions automatically by applying FCM on the results of the alignment tools. Here n is the number of alignment tools, p is the number of clusters or the membership functions. O_1 and O_2 are two ontologies being aligned.

gives us a way of comparing the alignments based on their uncertainties. Again, the alignment that has higher interval mean is stronger as it denotes that most of the alignment tools suggested this one higher similarity measure. So this alignment is most likely to be a strong one. At this point we needed to develop an equation that balances the two concepts of uncertainty(interval width) and strength(interval mean). We introduced "potential" as a quality metric of an alignment. The equation 1 is a weighted harmonic mean formula that provides more emphasis on the interval mean than the interval (width). The reason of using weighted harmonic mean instead of generic one is that, intuitively it is more significant to have higher interval mean rather than having lower uncertainty. If an alignment shows higher interval mean, it is more likely to be in the strong cluster.

$$P_i = (1+\beta^2)\frac{mean_i * interval_i}{mean_i + \beta^2 interval_i} \tag{1}$$

Where, P_i is the potential of the alignment with index i. β is a constant. $Interval_i$ is the interval of the ith alignment. $Mean_i$ is the arithmetic mean of $Interval_i$. The potential value reflects both the uncertainty and the strength of an alignment. Therefore, the best alignment has the highest mean and least interval(width).

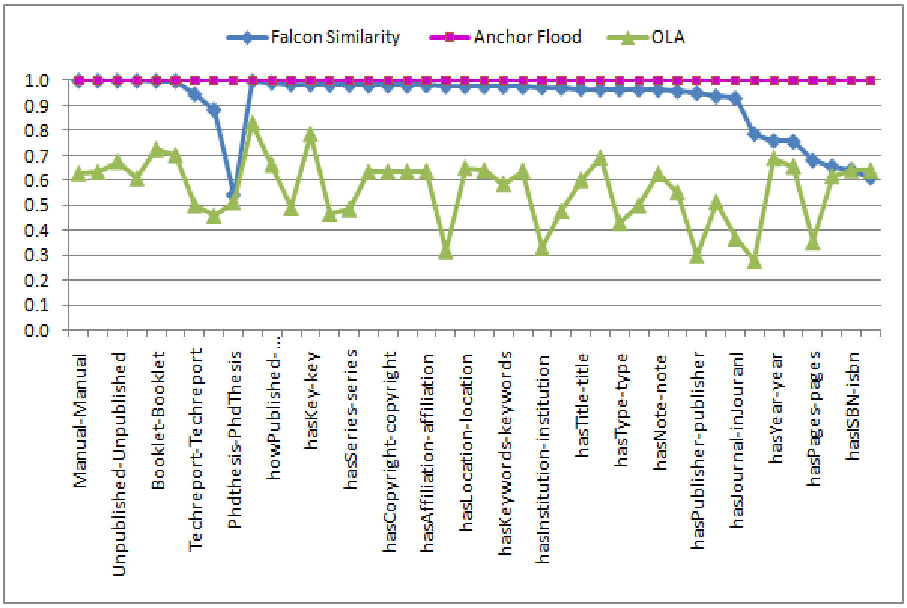


Fig. 2. Alignments with their similarity measure values found after running three ontology alignment tools for a pair of bibliographic text ontologies

4 Case Studies

4.1 Bibliographic Text Ontologies

We have experimented with a pair of ontologies in bibliographic text and three automatic alignment tools Falcon AO [2], Anchor Flood [4] and OLA [5]. Figure 2 shows the alignment results found after feeding the two OWL files of bibliographic text to all three of the alignment tools. For simplicity we restricted ourselves considering only the 42 alignments found common in all three of the tools. According to our methodologies we applied FCM. Fig. 3 shows the degree of membership values found after applying FCM [11] on the similarity measure values of Falcon AO [2]. In this figure, if we draw a line connecting all the membership values of strong cluster, we will obtain the membership function for this cluster. Similar membership functions can be drawn for medium and weak clusters.

We performed similar operations on the alignment similarity measure values obtained from other tools. As the tools did not agree upon the similarity measure values of the alignments, for a particular cluster we obtained very different MFs from them. In order to aggregate multiple alignment tools we applied IT2MF on these MFs. We found alignment number 11 ("howPublished"-"howPublished") and alignment number 13 ("hasKey"-"key") have membership degree interval from 0.2 to 1.0 and 0.93 to 1.0 respectively. According to our methodology, number 13 is stronger and less uncertain than number 11, as it has higher mean and shorter interval width. For strong cluster figure 4 clearly shows how the

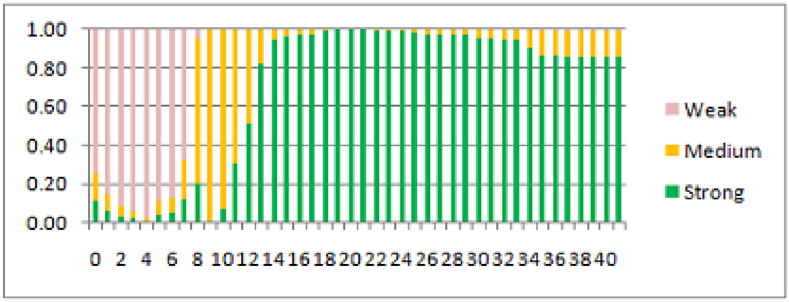


Fig. 3. Membership functions for three clusters: weak, medium and strong generated after applying FCM on Falcon similarities

uncertainty range looks like for the 42 alignments we considered. We excluded Anchor Flood [4], as it provided same membership degree for all the alignments and this will moderate the actual potential value of an alignment.

In order to sort our alignments based on their strength and uncertainty we calculated their potential values according to the equation 1. Here we have chosen $\beta = 10.0$, as it distributes the potential value in a way that the interval mean gets much higher emphasis than the interval (width). We found that alignments with ambiguous similarity measure (very high in one tool and very low in other) achieved a moderate potential value that respects all the algorithmic views of those tools we considered.

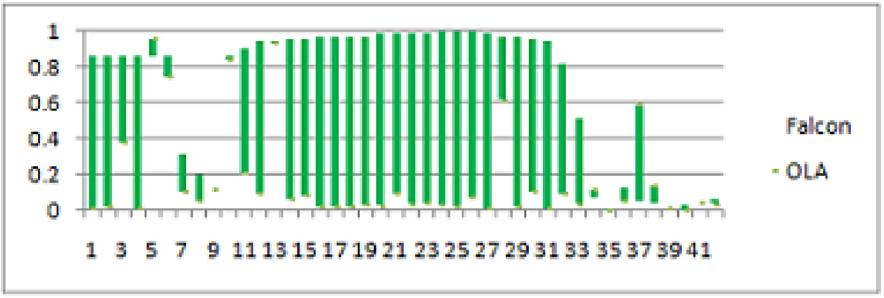


Fig. 4. Interval type 2 membership function for strong cluster considering two alignment tools. As Anchor flood assigned 1.0 for all of the alignments we exclude this tool here to clarify the necessity of considering mean in the selection criteria.

4.2 Family Ontologies

We performed another case study with a pair of ontologies in Family domain. We obtained 14 alignments and their similarity values after running the three alignment tools with these ontologies. We repeated our methodologies in the same way as we performed in the case of bibliographic text. Unfortunately, these two ontologies were similar enough to build a sparse membership function like bibliographic text. Most of the alignments received high range of uncertainty (more than 0.6), because the similarity measures of the alignments were high (more than 0.98) and we tried to divide them into three clusters in between 0.98 and 1.0. Therefore the centers of those clusters were close enough and this made the clustering decision difficult.

We could not find any set of alignments that could be used as a golden standard for this pair of ontologies. As they were small enough we performed the alignment ourselves and compared after sorting the alignments based on their calculated potential values. We found no significant difference between our approach and other alignment tools for the case of family ontologies.

4.3 Results

In order to justify our theory of calculating potentials we compared our results with the EON [12] ontology alignment contest results as our golden standard. But the problem was all existing alignments in EON have similarity value 1.0, whereas we have floating point similarity and potential values. For doing the comparison we termed an alignment "wrong" if it does not exist in the EON result. We sorted the 42 alignments (figure 2) based on their calculated potential values and counted the number of "wrong" alignments. Table 4.3 summarizes the results of comparing our method with other existing alignment tools. It clearly shows that sorting the alignments based on their potential values provides better results than all other alignment tools we considered, because the potential value has been calculated by aggregating multiple alignment tools. We have not included results of sorting the alignments based on Anchor Flood, as it assigns same similarity measure(1.0) for all.

Table 1. Comparing the number of wrong alignments found after sorting the alignments in descending order based on different methods. The table reports the number of wrong alignments present in the first 25.

Sorted based on	Wrong	Position
Potential	1	15*th*
Falcon similarity	2	2*nd* and 19*th*
OLA similarity	2	12*th* and 19*th*
Average of 3 tools	3	9*th*, 13*th* and 21*st*

5 Conclusion

Most of the alignment tools provide a similarity measure but it is hard to justify the quality of an alignment only by looking at its similarity measure. Also this similarity measure value is different in different alignment tools, that raises ambiguity and uncertainty. This paper provides an ensemble model that aggregates the results of multiple ontology alignment tools. As described in the methodologies section we applied Fuzzy C Means clustering and IT2MF in order to obtain an aggregated fuzzy membership function that combines the results of all alignment tools we considered. Finally we introduced a formula for calculating the potential of an alignment, where the potential addresses both the strength and uncertainty of an alignment. We have also proved that if we sort the alignments based on their potential it performs better than all other alignment tools. We would like to extend this work by applying more ontology alignment tools in the future.

References

1. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proceedings of the National Conference on Artificial Intelligence, pp. 450–455 (2000)
2. Jian, N., Hu, W., Cheng, G., Qu, Y.: Falcon-ao: Aligning ontologies with falcon. In: K-Cap 2005 Workshop on Integrating Ontologies, pp. 87–93 (2005)
3. Mendel, J.M.: Uncertainty Rule-Based Fuzzy Logic Systems. Prentice Hall PTR, Upper Saddle River (2001)
4. Seddiqui, M.H., Aono, M.: An effcient and scalable algorithm for segmented alignment of ontologies of arbitrary size. Web Semant. 7(4), 344–356 (2009)
5. Euzenat, J., Valtchev, P.: Similarity-Based Ontology Alignment in OWL-Lite. In: ECAI 2004, pp. 333–337 (2004)
6. Eckert, K., Meilicke, C., Stuckenschmidt, H.: Improving Ontology Matching Using Meta-Level Learning. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 158–172. Springer, Heidelberg (2009)
7. Dunn, J.C.: A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Cluster. Cybernetics and Systems 3(3), 32–57 (1973)
8. Niwattanakul, S., Martin, P., Eboueya, M., Khaimook, K.: Ontology Mapping based on Similarity Measure and Fuzzy Logic. In: Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Quebec City, Canada, 2007, pp. 6383–6387 (2009)
9. Ding, Z., Peng, Y., Pan, R.: A Bayesian Approach to Uncertainty Modelling in OWL Ontology. In: Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications, Luxembourg, Germany (2004)
10. Tordai, A., Ghazvinian, A., van Ossenbruggen, J., Musen, M.A., Noy, N.F.: Lost in Translation? Empirical Analysis of Mapping Compositions for Large Ontologies. In: The Fifth International Workshop on Ontology Matching (OM 2010), at ISWC 2010, Shanghai, China (2010)
11. Hathaway, R.J., Davenport, J.W., Bezdek, J.C.: Relational duals of the c-means algorithms. Pattern Recognition 22, 205–212 (1989)
12. EON Ontology Alignment Contest, http://oaei.ontologymatching.org/2004/Contest/

Author Index