Proceedings of

SWWS' 01

The First Semantic Web Working Symposium

Stanford University, California, USA
July 30 - August 1, 2001

We appreciate the constributions from our institutional sponsors:

National Science Foundation

OntoWeb Network

DARPA DAML

INRIA

We appreciate the contributions from our industrial sponsors:

Organizers

Isabel F. Cruz
U. Illinois at Chicago
USA
(ifc@cs.uic.edu)

Stefan Decker
Stanford U.
USA
(stefan@db.stanford.edu)

Jérôme Euzenat
INRIA
France
(Jerome.Euzenat@inrialpes.fr)

Deborah McGuinness
Stanford U.
USA
(dlm@ksl.stanford.edu)

PC Members

Dan Brickley, University of Bristol, UK
Tiziana Catarci, University of Rome "La Sapienza", Italy
Vassilis Christophides, ICS-FORTH, Greece
Steve Demurjian, University of Connecticut, USA
Max J. Egenhofer, University of Maine, USA
Peter Eklund, Griffith University, Australia
Dieter Fensel, Free University of Amsterdam, The Netherlands
Asunciòn Gomez-Perez, Universidad Politecnica de Madrid (UPM), Spain
Benjamin Grosof, MIT, USA
Natasha Fridman Noy, Stanford University, USA
Nicola Guarino, CNR, Italy
Pat Hayes, University of West Florida, USA
Jim Hendler, DARPA and University of Maryland, USA
Masahiro Hori, IBM Tokyo Research Laboratory, Japan
Ian Horrocks, University of M anchester, UK
Ora Lassila, Nokia Research, USA
Raphael Malyankar, Arizona State University, USA
Massimo Marchiori, W3C, University of Venice, USA, Italy
Brian McBride, Hewlett Packard, UK
Sheila McIlraith, Stanford University, USA
Robert Meersman, Free University Of Brussels, Belgium
Eric Miller, W3C, USA
Enrico Motta, The Open University, UK
Amedeo Napoli, LORIA, France
Dimitris Plexousakis, ICS-FORTH & Univ. of Crete, Greece
Peter Patel-Schneider, Lucent Technologies, USA
Guus Scheiber, University of Amsterdam, The Netherlands
Amit Sheth, University of Georgia and Taalee Inc, USA
Steffen Staab, University of Karlsruhe, Germany
Heiner Stuckenschmidt, University of Bremen, Germany
Frank van Harmelen, Free University of Amsterdam, The Netherlands

Reviewers

Sofia Alexaki
Cecilia Bastarrica
Peter Becker
Olivier Brunet
Paul Calnan
Farid Cerbah
Ying Ding
Michael Klein
Maurizio Lenzerini
Tarcisio Lima
Donald Needham
Borys Omelayenko
Charles E. Phillips, Jr.
Jeff Z. Pan
Margie Price
Mitchell Saba
Feng Zhang

Semantic Web Working Symposium
Program

MONDAY JULY 30	8:00-9:00	**SWWS Registration and Breakfast (at the TCSEQ Center)**			
	9:00-9:30	**Welcome (Organizers & Jim Hendler) (TCSEQ 200)**			
	9:30-10:30	**Invited Talk by Eric Miller (W3C Semantic Web Activity Leader) (TCSEQ 200)**			
	10:30-11:00	**Coffee Break (at the TCSEQ Center)**			
	11:00-12:30	**Working Track 1 (Room CIS-X101)** ***Ontology and Ontology Maintenance*** (Facilitators: Mark Tuttle, Apelon and Deborah McGuiness, KSL, Stanford University) *The Semantic Web As "Perfection Seeking": A View from Drug Terminology* Mark Tuttle, S. Brown, K. Campbell, J. Carter, K. Keck, M. Lincoln, S. Nelson, M. Stonebraker *Industrial Strength Ontology Management* Aseem Das, Wei Wu & Deborah McGuinness *OntoMap or How to Choose Upper-Model in One Day* Atanas Kirakov, Kiril Simov, Marin Dimitrov	**Working Track 2 (Packard 101)** ***Interoperability, Integration, Composition*** (Facilitator: Vipul Kashyap, Telcordia) *Towards Semantic Interoperability in Agent-based Coalition Command Systems* David Allsopp, Patrick Beautement, John Carson, and Michael Kirton *Object Interoperability for Geo spatial Applications* Isabel F. Cruz and Paul Calnan *Semantic Brokerage of Intellectual Property Rights* Roberto Garcia and Jaime Delgado	**Working Track 3 (Bio T175)** ***(Web-) Services and Applications*** (Facilitators: Jim Hendler, DARPA and Sheila McIlraith, KSL, Stanford University) *DAML-S: A Semantic Markup Language For Web Services* Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, Katia Sycara, Honglei Zeng *Serching for services on the semantic web using process ontologies* Mark Klein, Abraham Bernstein *Approach to Service Description for Matchmaking and Negotiation of Services* David Trastour, Claudio Bartolini	**Tutorial Track (TCSEQ 200)** (Chair: Charles Petrie) ***Ontology Engineering*** (Natalya F. Noy, SMI, Stanford University)

	12:30-02:00	**Lunch (at the TCSEQ Center), Demos (at Gates 104),** Demos by Verticalnet, Spirit-Soft, Mondeca, Empolis, SC4, Lastmileservice, UMBC, Stanford Medical Informatics, Griffith University, University of Bristol			
	02:00-03:30	**Working Track 1 (CIS-X101)** ***Ontology and Ontology Maintenance*** *The "Emergent" Semantic Web: An approach for derivation of semantic agreements on the Web* Clifford Behrens, Vipul Kashyap *Ontology versioning on the Semantic Web* Michel Klein & Dieter Fensel *Ontology Library Systems: The key for successful Ontology Reuse* Ying Ding & Dieter Fensel	**Working Track 2 (Packard 101)** ***Interoperability, Integration, Composition*** *Adding Multimedia to the Semantic Web: Building an MPEG-7 ontology* Jane Hunter *Overcoming Ontology Mismatches in Transactions with Self-Describing Agents* Drew McDermott, Mark Burstein and Douglas Smith	**Working Track 3 (Bio T175)** ***(Web-) Services and Applications*** *The Briefing Associate: A Role for COTS applications in the Semantic Web* Marcelo Tallis, Neil Goldman, Robert Balzer *ITTALKS: A Case Study in the Semantic Web and DAML* R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, Sovrin Tolia *Open Learning Repositories and Metadata Modeling* Hadhami Dhraief, Wolfgang Nejdl, Boris Wolf, Martin Wolpers	**Tutorial Track (TCSEQ 200)** ***Semantic B2B Integration*** (Christoph Bussler Oracle Corporation)
	03:30-04:00	**Coffee Break (at the TCSEQ Center)**			

	04:00-06:00	**Working Track 1** **(Room CIS-X101)** ***Ontology and Ontology Maintenance*** *UML and the Semantic Web* Stephen Cranefield *Metamodeling Architecture of Web Ontology Languages* Jeff Pan, Ian Horrocks *DAML+OIL is not Enough* Sean Bechhofer, Carole Goble, Ian Horrocks *Semantic Web Modeling and Programming with XDD* Chutiporn Anutariya, Vilas Wuwongse, Kiyoshi Akama, Vichit Wattanapailin	**Working Track 2** **(Packard 101)** ***Interoperability, Integration, Composition*** *A Framework for Ontology Integration* Diego Calvanese, Giuseppe De Giacomo and Maurizio Lenzerini *A Scalable Framework for Interoperation of Information Sources* Prasenjit Mitra, Gio Wiederhold and Stefan Decker *On the Integration of Topic Maps data with RDF data* Martin S. Lacher and Stefan Decker *A formal infrastructure for Interoperability on the Semantic Web* Jerome Euzenat	**Working Track 3** **(Bio T175)** ***(Web) Services and Applications*** *CREAM: Creating relational metadata with a component-based, ontology-driven annotation framework* Siegfried Handschuh, Steffen Staab, Alexander Maedche *OntoWebber: Model-Driven Ontology-Based Web Site Management* Yuhui Jin, Stefan Decker, Gio Wiederhold *Indexing a web site with a terminology oriented ontology* E. Desmontils, C. Jacquin *A semantic model for specifying data-intensive Web applications using WebML* Sara Comai, Piero Fraternali	**Tutorial Track** **TCSEQ 200** ***Demos*** (Facilitator: Natalya F. Noy, SMI, Stanford University) *Verticalnet* - Aseem Das *Spirit-Soft* - Steve Ross-Talbot *Mondeca* - Bernard Vatant *Empolis* - Hans Holger Rath *LastMileService* - Raj Bapna *Stanford Medical Informatics* - Monica Crubezy, Natalya F. Noy *DSTC/Griffith University* Peter Eklund
	07:00	**Banquet at the Faculty Club**			
TUESDAY	08:00-09:00	**SWWS Registration and Breakfast (at the TCSEQ Center)**			
	09:00-10:00	**Invited Talk: Michel Biezunski, Steven Newcomb on TopicMaps (Room TCSEQ 200)**			
	10:00-10:30	**Coffee Break (at the TCSEQ Center)**			

JULY 31					
	10:30-12:00	**Panel: Emerging Semantics (Room TCSEQ 200)** **Vipul Kashyap** **Panelists:** *Ora Lassila (NOKIA Research)* *Jim Hendler (DARPA)* *Dieter Fensel (Free University of Amsterdam)* *Umeshwar Dayal (Hewlett-Packard)* *Clifford A Behrens (Telcordia)*			
	12:00-01:30	**Lunch (at the TCSEQ Center), Demos (at Gates 104),** Demos by Verticalnet, Spirit-Soft, Mondeca, Empolis, SC4, Lastmileservice, UMBC, Stanford Medical Informatics, Griffith University, University of Bristol			
	01:30-03:30	**Working Track 1** **(Room CIS-X101)** ***Ontology and Ontology Maintenance*** *Utilizing Host-Formalisms to Formally Extend RDF-Semantics* Wolfram Conen, Reinhold Klapsing *RDF model revisited - or: how to make the most out of Reifications and Containers* Wolfram Conen, Reinhold Klapsing *Track summary and discussion* Mark Tuttle, Deborah McGuinness and Stuart Nelson	**Working Track 2** **(Packard 101)** ***Interoperability, Integration, Composition*** *Describing Computation within RDF* Chris Goad *Design Rationale for RuleML: A Markup Language for Semantic Web Rules* Harold Boley, Said Tabet and Gerd Wagner *Enabling Semantic Web Programming by Integrating RDF and Common Lisp* Ora Lassila *Track Summary and Discussion* Vipul Kashyap	**Working Track 3** **(Braun Auditorium)** ***(Web) Services and Applications*** ***to be announced***	**Tutorial Track** **(TCSEQ 200)** ***Models and Languages for Describing and Discovering E-services*** (Fabio Casati and Ming-Chien Shan, Hewlett-Packard)
	03:30-04:00	**Coffee Break (at the TCSEQ Center)**			
	04:00-06:00	**Facilitators Report and Announcement of BOF Sessions (TCSEQ 200)**			
	07:00	**Joint Reception with ICCS at the Faculty Club**			

WEDNESDAY AUGUST 1	8:00-9:00	**Breakfast (at the TCSEQ Center)**
	09:00-10:30	**Birds of the Feather Sessions in Gates 104 (40) Packard 101 (100) TCSEQ 200 CIS-101 (100) Joint Session with ICCS/DL**
	10:30-11:00	**Coffee Break (at the TCSEQ Center)**
	11:00-12:00	**BOF Wrap Up, Follow-up actions and Farewell (TCSEQ 200)**


STANFORD UNIVERSITY
PARKING AND CIRCULATION MAP
SCALE (Approx.) 5.2 in. = 1 mile (1 : 12,000)
1/4 mile
indicates areas under construction
Make sure to check the actual signs in the lot before you park. The parking designations on this map, along with all our conditions and regulations, are subject to change at any time.
Also note that many lots are signed only at the entrance. In "shared" lots – i.e., those posted "C/East Residences" – you may park with either permit designated.
Meter or time limit 6 am to 4 pm (M-F)
A permit 6 am to 4 pm (M-F)
A and C permits 6 am to 4 pm (M-F)
Motorcycle permit 6 am to 4 pm (M-F)
These permits valid in Parking Structures 1 & 2
A, C, and Z permits 6 am to 4 pm (M-F)
'A Carpool' permits only 6 am to 4 pm (M-F)
Pedestrian Zone and paths – Unauthorized motor vehicles may be cited with moving violation.
Daily parking permit sales: 855 Serra St., Memorial Aud. visitor ctr, Hospital Security, Tresidder info Window, Bookstore.
Disabled Person permits are valid in all marked spaces on campus. DP spaces are available at & around most buildings.
Resident student permit all hours
EA East
SO South
WE West
ES Escondido
IE Inner East
SH Stanford Housing
ST Stern
INSET
Parking and Transportation info: 723-9362
Marguerite Shuttle info: 723-9362
University Telephone Operator: 723-2300
REVISED FEBRUARY, 1997
COPYRIGHT © 1997 BOARD OF TRUSTEES OF LELAND STANFORD, JR. UNIVERSITY. REPRODUCTION PERMITTED ONLY WITH PERMISSION OF THE OFFICE OF TRANSPORTATION
Bragg Auditorium
Working Track 3
(Services and Applications) on
Tuesday
Lunch Demo Sessions
Gates 104
CIS-X101 Track 2
(Interoperation)
Packard 101
Track 1
(Ontology)
BIO-T175
Working Track 3
(Services and Applications) on
Monday
TSEQ Center
Plenary Sessions
and Tutorials
Faculty Club
Social Events
Lake Lagunita
CAMPUS DRIVE EAST
CAMPUS DRIVE WEST
JUNIPERO SERRA BLVD
SAND HILL RD
EL CAMINO REAL
PALM DRIVE
STANFORD STADIUM
Stanford Shopping Center

Table of Contents

Foreword

The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. In order to make this vision a reality for the Web, supporting standards, technologies and policies must be designed to enable machines to make more sense of the Web, with the result of making the Web more useful for humans. Facilities and technologies to put machine-understandable data on the Web are rapidly becoming a high priority for many communities. For the Web to scale, programs must be able to share and process data even when these programs have been designed totally independently. The Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people.

The First International Semantic Web Working Symposium (SWWS) took place in Stanford California, July 30 through August 1, 2001. The technical program of SWWS presented the state of the art in the development of the principles and technology that will allow for the Semantic Web to become a reality. There were two invited talks, one by Eric Miller and the other one by Michel Biezunski and Steven Newcomb, and one panel chaired by Vipul Kashyap. Three parallel tracks consisted of 35 technical presentations selected from 58 submissions, and a fourth track consisted of tutorials. The tutorials were presented by Natalya Friedman-Noy (Ontology Engineering), Christoph Bussler (Semantic B2B Integration), Fabio Casati and Ming-Chien Shan (Models and Languages for Describing and Discovering E-services). Of the 58 submissions, 29 came from Europe, 21 from the USA, 3 from Australia and New Zealand, 2 from China, Japan/Thailand, and 3 from unidentified countries. The rate of acceptance was approximately two out of three for all the groups except for the last two mentioned (respectively 50% and 0%). The four tracks were: "Ontology and Ontology Maintenance", facilitated by Deborah McGuinness and Mark Tuttle, "Interoperability, Integration, and Composition", facilitated by Vipul Kashyap, and "Services and Applications", facilitated by Jim Hendler and Sheila McIlraith. Charles Petrie chaired the tutorial track. In a joint session, the track facilitators presented a report on their respective tracks. In collaboration with DL 2001, birds of a feather working groups met during the last morning. The social program included a banquet at the Stanford Faculty Club and a joint reception with ICCS 2001.

The organizers would like to thank everybody who made possible the unique discussion of ideas and contributions during the two and half days of the Semantic Web Working Conference–the authors of the papers, the invited speakers, facilitators, and panelists, and the members of the Program Committee: Dan Brickley, Tiziana Catarci, Vassilis Christophides, Steve Demurjian, Max J. Egenhofer, Peter Eklund, Dieter Fensel, Asunción Gómez-Pérez, Benjamin Grosof, Natasha Fridman Noy, Nicola Guarino, Pat Hayes, Jim Hendler, Masahiro Hori, Ian Horrocks, Ora Lassila, Raphael Malyankar, Massimo Marchiori, Brian McBride, Sheila McIlraith, Robert Meersman, Eric Miller, Enrico Motta, Amedeo Napoli, Dimitris Plexousakis, Peter Patel-Schneider, Guus Scheiber, Amit Sheth, Steffen Staab, Heiner Stuckenschmidt, and Frank van Harmelen. We especially would also like to thank the additional reviewer: Sofia Alexaki Cecilia Bastarrica, Peter Becker, Olivier Brunet, Paul Calnan, Farid Cerbah, Ying Ding, Michael Klein, Maurizio Lenzerini, Tarcisio Lima, Donald Needham, Borys Omelayenko, Charles E. Phillips, Jr., Jeff Z. Pan, Margie Price, Mitchell Saba, and Feng Zhang The organization of this event in a very short time would have not been possible without the help of Arturo Crespo, Jennifer Espinoza, Martin Lacher, Annemarie Feely, Marianne Siroker, Bob Spillers, Prasenjit Mitra, Sarah Weden, and Seregey Melnik. Last, but not least, we would like to thank the generous financial support of the National Science Foundation, DARPA, the Ontoweb Network, INRIA, and of the following companies: VerticalNet, NOKIA, SpiritSoft, ENIGMATEC, Empolis, Connotate, Mondeca, Language and Computing, SC4, NetworkInference, Ontoprise, and LastMileServices.

Isabel F. Cruz	Stefan Decker	Jérôme Euzenat	Deborah McGuinness
U. Illinois at Chicago	Stanford U.	INRIA	Stanford U.
USA	USA	France	USA

Tutorials

Tutorial 1: Ontology Engineering

Natalya F. Noy
Stanford University
USA

Abstract
In recent years the development of ontologies - explicit formal specifications of the terms in the domain and relations among them -has been moving from the realm of Artificial-Intelligence laboratories to the desktops of domain experts. Ontologies have also become common on the World-Wide Web. The ontologies on the Web range from large taxonomies categorizing Web sites (such as on Yahoo!) to categorizations of products for sale and their features (such as on Amazon.com). On the Web and in many large applications ontologies serve a variety of purposes: making the knowledge about a particular domain explicit, sharing and reusing this knowledge, analyzing domain knowledge. A number of languages for defining ontologies on the Web, such as RDF(S) and DAML+OIL, are under development. In this tutorial we will discuss why one would build an ontology and present a methodology for creating ontologies based on declarative knowledge representation systems. We will present ontology examples, discuss common problems and pitfalls in ontology development and approaches to solving the problems. We will also give a brief overview of the current research issues in ontology engineering and compare some Web-based ontology-representation languages.

About The Speaker
Natalya F. Noy is a research scientist in the Stanford Medical Informatics laboratory at Stanford University. Her research focuses on ontology development and evaluation, semantic integration of ontologies, and making ontology-development accessible to experts in noncomputer-science domains. She is a member of the Protégé group at Stanford University, which develops a graphical and extensible software environment for ontology editing. She has received a PhD degree from Northeastern University concentrating on the challenges of ontology development in experimental sciences.

Tutorial 2: Semantic B2B Integration - Concepts, Architecture, Implementation and Deployment

Dr. Christoph Bussler
Oracle Corporation
USA

Abstract

This tutorial will give an introduction to the field of business-to-business (B2B) integration from a technical viewpoint with the focus on semantic integration aspects. The set of B2B integration concepts is introduced as well as their implementation in form of a technical semantic B2B integration architecture. A mix of examples is taken illustrating the problems that need to be solved in semantic B2B integration projects. The tutorial enables the audience to identify semantic B2B integration problems as well as to determine the benefits and deficiencies of various technical integration architecture approaches or B2B integration technologies.

About The Speaker

Christoph Bussler is Member of Oracle's Integration Platform Architecture Group based in Redwood Shores, CA. He is responsible for the architecture of Oracle's next generation integration platform product. Prior to joining Oracle he was at Jamcracker, Cupertino, CA, responsible for defining Jamcracker's ASP aggregation architecture, Netfish Technologies, Santa Clara, CA, responsible for Netfish's B2B integration server, The Boeing Company, Seattle, WA, leading Boeing's workflow research and Digital Equipment, Mountain View, CA, defining the policy resolution component of Digital's workflow product. He has a Ph.D. in computer science from the University of Erlangen, Germany and a Master in computer science from the University of Munich, Germany.

Tutorial 3: Models and Languages for Describing and Discovering E-services

Dr. Fabio Casati
Dr. Ming-Chien Shan

Hewlett Packard Laboratories
USA

Abstract

E-services are business functions made available via the Internet by service providers, and accessible by clients that could be human users or software applications. The main benefit of the e-services environment is that clients are able to dynamically discover the available e-service that best meets their needs, to examine its properties and capabilities, and to determine if and how to access it. However, in order to deliver e-services to clients, service providers are faced with several challenges. In particular, they need to describe e-services in a way that is accessible and understandable by the clients and to advertise them in web directories, so that they can be discovered by brokers as well as by end-users. In this tutorial we discuss the main requirements for models and languages for service description and discovery, and we present relevant approaches proposed by standardization consortia.

About The Speakers

Fabio Casati is a researcher at HP Labs, Palo Alto. He got his PhD from Politecnico di Milano (Italy) in 1999. His research interests include workflow management, e-services, mobile environments, and business process intelligence. He is author of more than 30 papers in international conferences and journals, and has served as organizer and program committee member in several conferences. He is also a lecturer of "Technologies for e-business" at San Jose State University.

Ming-Chien Shan is a research program manager in the Hewlett Packard Laboratories, Palo Alto, California. He joined IBM DB2 team in 1978 working on query optimization, data definition manager and distributed DBMS. He then joined HP in 1985 and managed various research projects, including object-oriented DBMS, heterogeneous DBMS, workflow and telecom service provisioning. Currently, he is the manager of e-business solutions program. Ming-Chien received his PhD degree in computer science from University of California, Berkeley in 1980. He has published more than 50 technical papers and been granted 15 software patents.

The Semantic Web As "Perfection Seeking:" A View from Drug Terminology

Mark S. Tuttle, *Apelon, Inc.*, mtuttle@apelon.com
Steven H. Brown, MD, *Vanderbilt University / Veterans Administration*
Keith E. Campbell, MD, PhD, *Inoveon, Inc.*
John S. Carter, *University of Utah / Apelon, Inc.*
Kevin D. Keck, *Keck Labs*
Michael Lincoln, MD, *University of Utah / Veterans Administration*
Stuart J. Nelson, MD, *National Library of Medicine*
Michael Stonebraker, PhD, *Massachusetts Institute of Technology*

Abstract. To date, the Semantic Web has viewed formal terminology, or ontology, as either immutable, or something that can change but that has no past and no future – only a present. Change, or process – such as "perfection seeking," is outside the scope of the proposed "semantics," except in so far as it is represented in attributes. In contrast, current U.S. Government efforts to formalize drug (medication) terminology are being driven by the need to manage changes in this terminology asynchronously and longitudinally. For example, each year the FDA (Federal Drug Administration) approves about 150 new drugs and thousands of changes to the "label" of existing drugs, the VHA (Veterans Health Administration) must manage new drugs, label changes, and tens of thousands of drug "packaging" changes, and the NLM (National Library of Medicine) must maintain a current index of references to proposed or approved medications in the world's biomedical literature. We propose that an emerging multi-federal-agency reference terminology model for medications, mRT, be used to drive development of the necessary repertoire of "semantic" change management mechanisms for the Semantic Web, and that these "process" mechanisms be organized into an ontology of change.

1. Overview – Using mRT to drive the development of Semantic Web change management

Creating standards, especially standards that create information industry infrastructure, is difficult, time-consuming and at constant risk for irrelevance and failure. One way to mitigate this risk, and secure the participation of the diverse interest groups required to make such standards a success is to focus on process – as in the process that produces and maintains a good standard. This is in contrast to an approach that says some existing artifact selected from a list will be THE standard, and all the others will NOT be the standard. An observation that we attribute to Betsy Humphreys from the National Library of Medicine in the context of biomedical terminology standards is that it doesn't matter where you start, i.e., it doesn't much matter which terminology or terminologies one selects as a starting point; instead what does matter is the process by which the proposed standard evolves to achieve and sustain the desired degree of quality, comprehensiveness, and functionality. The process is what determines where the standard ends up.

Seen in this light, change, even a large amount of change, will be a feature of successful formal terminologies, or ontologies. We hope to demonstrate the feasibility and utility of this approach. The challenge in the context of the Semantic Web is to choose a representation for change that makes it explicit. Viewed this way the Semantic Web would be "perfection seeking,"[1] and the ongoing changes would be part of the semantics. The challenge with this approach is the formulation of the units of change and the creation of an ontology of these change units. This follows a Semantic Web notion expressed by Tim Berners-Lee in a discussion of Metadata Architecture [1] "… metadata itself may have attributes such as ownership and an expiry date, and so there is meta-metadata but we don't distinguish many levels, we just say that metadata is data and that from that it follows that it can have other data about itself. This gives the Web a certain consistency." Making change part of the Semantic Web would preserve that consistency.

One way to focus the development of the desired units, inter-relationships, and uses is to solve real problems and gain experience from deployments of these solutions; we propose to do this by formulating, deploying and evaluating what we now call "The New Drug Transaction." This transaction needs to supply diverse, operating healthcare and biomedical information systems with the requisite formal definition of a new drug, given a reference model, and do so at Web scale. The main challenge is how to do this in a way that first avoids breaking working applications that use the drug terminology and second preserves the longitudinal value of existing and future patient descriptions of medication use.

More generally, healthcare and biomedicine undergo constant change – some of it perfection seeking and some of it clerical – and the relevant terminology needs to change in parallel. Again, the challenge is to the extent possible to accommodate change without breaking what already works, and without losing the value of historical data.

A simple, large-scale model of longitudinal change management is that used by MEDLINE, the National Library of Medicine's citation database (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi). The formal "semantics" of MEDLINE are supported by MeSH (Medical Subject Headings), a concept-based biomedical terminology that is updated annually (http://www.nlm.nih.gov/mesh/meshhome.html). Each year, rules are written that transform citations indexed using the previous year's MeSH into citations indexed using the new version of MeSH. In this way, by "re-writing history," old citations can be retrieved as appropriate using current terminology. As can be appreciated, formulating the rules requires manual intervention and testing, but more than 11 million citations, each tagged with about a dozen index terms selected from some 18,000 concepts, are maintained longitudinally in this way. While MEDLINE has always been a pre-eminent and exemplar information retrieval system, the notion of "history re-writing" implies a loss of information; the declining cost of secondary storage may eliminate one of the reasons for such information loss, a theme that will be re-examined below.

[1] Peri Schuyler, then head of the MeSH (Medical Subject Headings) at the NLM, used this term in the context of the UMLS (Unified Medical Language System) Project in 1988.

2. Background - The Semantic Web is a generalization of formalization efforts already underway in healthcare and biomedicine

In his recent Scientific American article Berners-Lee argues that the Semantic Web is infrastructure, and not an application [2]. We couldn't agree more. To us, this view is a top-down and horizontal approach to Semantic Web objectives, and it is this kind of disciplined thinking that made the Web the success that it is today.

In parallel with this effort, progress toward related goals is occurring in healthcare and biomedicine and we think of this progress as bottom-up and vertical. Thus, at present, healthcare and biomedicine have a repertoire of standard terminologies and standard messages and, in some instances, their use is or will be mandated by law.[2] While current deployments of these artifacts lack the formality required for the Semantic Web they nevertheless represent a rehearsal of many of the processes that the Semantic Web will require. Further, as will be described in a later section, the shortfalls of current healthcare terminology and message standards are driving a new generation of healthcare terminologies and messages that do have some of the desired formal properties. All this is part of a gradual evolution in healthcare information technology that is changing its focus from "systems" to "data," [3] [4] a trend predicted in [5]. The authors believe that the major forcing function for this evolution is the need to "scale" healthcare information technology to ever larger enterprises and collections of individuals and enterprises; while this trend began before the Web, the presence of the Web has accelerated the change.

What is missing in healthcare and biomedicine is a way to link its relevant progress and experience with that occurring in the Semantic Web community. The Web influences healthcare information technology, but the Web is little influenced by lessons learned in healthcare IT. We believe that medications represent a domain in which these two activities can be joined productively. The potential significance of such a joining cannot be over-estimated. Healthcare now costs the U.S. more than $1 trillion/year, and medications are the largest single category of cost and the fastest growing category of cost.[3] They are also involved in a significant number of life-threatening medical errors. [6]

At a deeper level, we believe that the Semantic Web is an opportunity to shrink the "formalization gap" described by Marsden S. Blois, PhD, MD (1918-88). Blois argued that overcoming this gap was the fundamental challenge of medical informatics: 'This discontinuity in formalization between a manual (human) medical information process and the machine code necessary to accomplish comparable ends begins at a very high descriptive level and it is not itself a concern of computer science. If this concern is to be given a name at all, it must be regarded as concerning medical applications, and it is increasingly being referred to as "medical information science" in the United States, and as "medical informatics" in Europe. It will be the task of this new discipline to better understand and define the medical information processes we have considered here, in order that appropriate activities will be chosen for computerization, and to improve the man-machine system.' [7] One rationale for a "perfection seeking" approach to the

[2] HIPAA (Health Insurance Portability and Accountability Act).

[3] Last year, VHA (Veterans Health Association) spent about $2.5 billion on medications, and MHS (Military Health System – covering active duty personnel and their dependents) spent about $1.5 billion. Personal conversation, Donald Lees, RPh, 6/01.

Semantic Web is the difficulty of getting the formalizations right, and of maintaining them, and the patient descriptions based on them, in the face of change.

3. A model - semantic definitions for medication active ingredients

If change management were not such a critical issue, already complete approximations of the medication reference model shown in ***Figure 1*** could be used by Semantic Web developers to test proposed representations. Carter, et al. [8] describe how about 1,000 active ingredients were given "Aristotelian" definitions represented in Description Logic and "published" in XML. One result of this effort was a focus on the emerging importance of "The New Drug Transaction" as a necessary conjunct to expansion of the model to cover all important active ingredients, and to trial deployments.

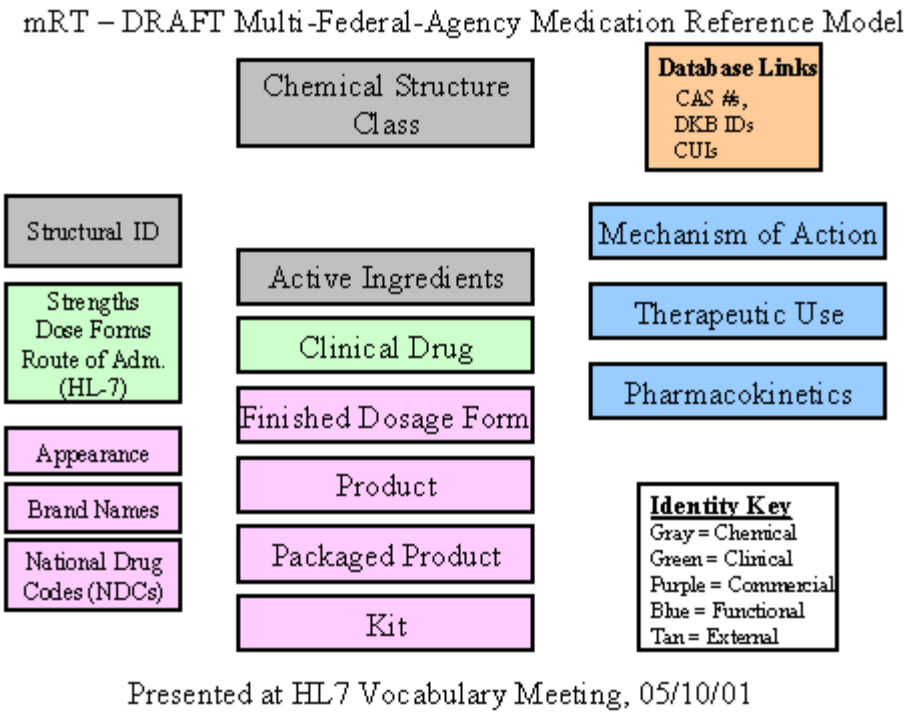


Figure 1 – DRAFT formal model of medications for potential use by three Federal Agencies: Active ingredients have "Aristotelian" definitions represented using Description Logic; these definitions will place each Active Ingredient in an IS_A hierarchy of Chemical Structure Classes, and describe each Active Ingredient using named relationships into reference taxonomies for, respectively, Mechanism of Action, Therapeutic Use, and Pharmacokinetics. Each Active Ingredient (molecule) will also have a machine-processible three-dimensional structural description (identifier). Not shown are inactive ingredients and other necessary details.

This model, developed over the last few years, has proven remarkably robust in the face of multi-disciplinary and multi-institutional inspection, and sample instantiations. Its next test will be to represent portions of various order-entry formularies used by the public and private sectors. A typical formulary covers about 10,000 – 100,000 "orderables" and the goal will be to produce "useful" definitions of the active ingredients contained in these orderables using early versions of the reference taxonomies for Chemical Structure, Mechanism of Action, Therapeutic Use, and Pharmacokinetics. This test will also allow us to gain experience assembling and formalizing medication information obtained from multiple authorities and disciplines that is used for related but still different purposes. For example, there will be at least three different kinds of "Therapeutic Use," also called "indications" – "FDA approved", "VA approved"(generally a superset of FDA approved), and "Described in the Literature".[4] The whole notion of "orderables" will also force clarification of the boundary between the so-called "terminology model" (categories and hierarchies) and the "instance" or "database" model (the orderables themselves, along with all their attributes). Everyone agrees that that the former is a good way to organize the latter, and that there should be a boundary between the two models – that is, the two models are similar and related but not the same, but few agree on where the implementation boundary should be, especially in light of emerging interoperation requirements based on re-usable objects. This dilemma should resonate with those working on the Semantic Web.

4. A process – embracing change and making it explicit

The model presented in Figure 1 is little more than an academic exercise without accompanying productive change management. Currently, excepting MeSH and MEDLINE (described above), change management in authoritative, deployed biomedical terminologies is at best primitive. [9] [10] As a result, there are few "warehouses" of patient descriptions that can be searched over time, that is across changes in the terminologies used to formalize the descriptions. Of the few patient description repositories that support such "time travel" no two do so in the same way, and none use existing or proposed standards. An explicit goal of the mRT project is to begin to overcome this shortfall at least in the context of medications.

The view of change management presented here is a synthesis of current and emerging practices in healthcare terminology, e.g., the use of Description Logic, earlier and current work on the handling of time-oriented data in database system models, e.g., POSTGRES [11] and T-SQL [12] [13], and our current understanding of the Semantic Web. This synthesis can be summed up by the conclusion that "Process is more important than representation."

4.1 A "new drug" transaction

The first step in making formal terminology changes into a terminology/ontology "thing," or unit, is to create a unit of change that has the same general properties as any

[4] An important side-effect of this effort will be an authoritative collection of so-called "off-label" uses of medications; such uses represent legal, but not FDA-approved, medication indications.

other "thing-ness" unit. For example, given the appropriate reference taxonomies, used to (in the Description Logic sense) "classify" medications, one can create the desired reference terminology – mRT – by "adding" the (Aristotelian) definitions of each drug, one drug at a time. But, of course, this ignores, among many other things, the fact that the reference taxonomies need to be changed, too. Frequently, new drugs come with new mechanisms of action and new indications (therapeutic objectives), and thus the corresponding "new drug transaction" may need to update the reference taxonomies before adding the definition of the new drug. These latter cases will be covered in "Other transactions" below.

To make the simple case more tangible, here is one potential near term future of the kind of "New Drug Transaction" that does not require updating the reference taxonomies:

1) The FDA will approve a new drug and "publish," as XML, a newly "structured" version of the traditional package insert, or "label," designed to "explain" that drug to both humans and computers. (One can think of this document as a "contract" between the FDA and the drug manufacturer.) The data that will appear in the new drug transaction is the result of processes now in place at the FDA; regulations are pending that will increase the degree of machine-processibility and formality of this data. [14]
2) The NLM will further process and enhance the parts of the label that can be processed usefully by computers, and then "publish" it, once again in XML. The "enhancements" may include connections to the biomedical literature, related terminology and foreign language names.
3) Applications or servers electing to process the new drug transaction will see that the XML indicates that it is an "add," the simplest kind of transaction to process. That is, the transaction will add a new concept – the new drug, the appropriate relationships to other concepts in the various reference taxonomies, and attributes of the new drug. (In every formulary or medication reference terminology known to the authors this is done manually, at present.)

It is not hard to imagine that most applications, e.g., drug order-entry systems, would be tolerant of such an insertion and subsequently "do the right thing." However, the problem with this simple form of the new drug transaction is that, as described by domain experts, most new drugs represent "changes in understanding," and it is not at all clear how existing applications can deal with such changes in understanding automatically, or know when they need help from humans. An extreme instance of such new understanding would be a drug that triggered a reorganization of the Aristotelian classification of existing drugs. (Changes in understanding due to pharmacogenetics may cause these kinds of "re-organizing" updates.)

4.2 Other transactions

In this context, "changes in understanding" are represented by changes in the reference taxonomies, e.g., for chemical structure, mechanism of action, pharmacokinetics, and therapeutic use. That is, a typical "new drug transaction" will need to include one or more changes to the reference taxonomies along with the (simple) "add" described above,

and these changes will represent "changes in understanding." It can be assumed that changes to the reference taxonomies will "break" existing applications, e.g., the decision support that operates in conjunction with order entry. The authors claim that to the degree that we can overcome this problem in the context of medication terminology maintenance that we are solving a problem that will be faced by the Semantic Web.

As presently planned our solution will be built on two foundations: First, mRT will not "overwrite" information; that is, per POSTGRES [15] any "garbage collection" or "archiving" will be handled asynchronously with new drug transactions, the practical effect being that an explicit, time-oriented, history of mRT is available to applications at all times. Second, appropriate use of Description Logic permits consistency-preserving updates; for example, if prior to execution of the new drug transaction an off-line, an updated copy of mRT is "reclassified" successfully (in the DL sense), then, in principle, mechanisms exist that can correctly update a run-time database (terminology server) "incrementally" (and thus quickly). Thus, such updates represent one useful repertoire of units of change.

Per earlier work of Stonebraker, et al. and more recent work of Snodgrass, et al., one can view a "database" as a time-oriented accumulation of changes. Thus the current "state" of the database is acquired through a computation, or "view," on the transactions accumulated over time. (See [13], for an enumeration of the many subtleties implicit here.) Part of the desired functionality is implemented, currently, in the MEME (Metathesaurus Enhancement and Maintenance Environment) deployed at the NLM. [16] The Metathesaurus (http://www.nlm.nih.gov/pubs/factsheets/umlsmeta.html) is a gigabyte+ synthesis of most authoritative biomedical terminologies, now released multiple times per year. Increasingly, a "release" is becoming a report on a time-oriented database.[5] Gradually, the whole notion of a "release" will become less important, and, instead, the Metathesaurus will be seen as a time-oriented record – a no-information-loss history – of authoritative terminologies. Of interest to those trying to deploy solutions on the Semantic Web, in run-time systems, use of incremental "write-once / read-many" databases make locking and error recovery significantly simpler.

We expect that the simple new drug transaction will be the easiest formal unit of change to specify. Quantitatively, the most important unit of change will be a transaction that introduces a change to the definition of an existing medication. For practical reasons the latter transactions are both the most important to accommodate in existing medication order entry systems and the most difficult. Frequently, they affect how drugs are to be used, e.g., the change may be a new contraindication.

Complicating this approach are the presence of larger changes-in-understanding – so-called "lumps" and "splits" - that, seemingly, violate the "axioms" implicit or explicit in DL tools. "Splits" occur when a concept is "split" into two or more concepts, typically because of the emergence of new knowledge. The latter may be new concepts or existing concepts. And the original concept that is split may be retired, or it may be retained as a subsumer of the "split" concepts. Splits are most often prompted by new information system needs, related to the emergence of new knowledge. Similarly, "lumps" – the merging of two previously distinct concepts – is usually prompted by the detection of clerical errors, or by the discovery that two things we thought were different proved not to be. As will be appreciated by those who favor the use of Description Logic (DL) in

[5] Brian Carlsen, personal conversation.

these contexts, a feature of DL, namely its support for formal definitions, helps to decrease the number of inadvertent "missed synonyms". Alternatively, some less mature domains, e.g., Bioinformatics, avoid the problem by using terminologies in which terms are freely lumped or split as needs dictate.

4.3 An ontology of change

If we view a formal terminology or ontology as a corpus of "facts," or assertions, collected over time, then one can contemplate an ontology of such facts, or changes. This much is straightforward. The difficulty is defining and implementing the semantics to be attached to each type of "change unit." One step toward such semantics is the simple expedient of tagging each terminologic unit – concept, term, relationship, and attribute - with a "Start Date" and (any) "End Date"; then, in principle, an application can know the state of the terminology at any point in time. More disciplined and complete forms of such semantics are what are needed to preserve the longitudinal functionality of systems that use the ontology, and what will be needed to transfer knowledge gained from a successful test of the new drug transaction to the Semantic Web.

In the MEDLINE - "rewriting history" - example described above, semi-automated methods accommodate the effects of new concepts, retired concepts, split concepts and lumped concepts in MeSH, as best as can be done each year. Thus, one "blunt instrument" approach to the analogous problem in the Semantic Web is for every repository of historical information to have a companion "warehouse" that is consistent with the current relevant ontologies. The semantics of change are then implemented in the potentially frequent re-computation of this warehouse, as appropriate. The companion argument here is that so-called Clinical Data Repositories (CDRs) and some biomedical research databases are being implemented as "write-only" databases because they represent the authoritative archive of record. Any so-called "data-healing" is done outside the CDR in adjacent data warehouses that are built from queries that run against the authoritative archive. Such pragmatics may evolve into functional requirements for the Semantic Web.

Regardless, the challenge posed by "ontologizing" these units of change is to represent what, for example, should be inherited or shared by other units. Thus, the new drug transaction is a specialized version of the change transaction and thus should inherit any properties of the former. At present, it is not clear how "split" and "lump" should be handled, formally.

4.4 "Perfection Seeking"

While the notion of "perfection seeking" has been very helpful in that it helps those in an inter-disciplinary project "satisfice" in particular domains so as to make progress toward the over-all goal, it has not yet been formalized, e.g., in the form of a metric. At present, terminology and terminology process are bereft of quality metrics. One exception is some work by Campbell, et al., that measured the degree to which lexically implied subsumption (one term appearing within, or sharing sub-strings with, another term) had been represented logically, i.e., in DL, in a large healthcare terminology. [17] While the metric was aimed at measuring the yield of "lexically suggested logical

closure" it also revealed the degree to which the lexical suggestions were not converted to logical relationships, e.g., because of linguistic ambiguity.

A related hypothesis was that expressed by Blois, namely that conceptualization and naming was more stable and predictable at "lower" biological levels, e.g., for molecules. [18] Thus, we would expect fewer synonyms and fewer changes to the Chemical Structure portion of the formal definitions of ingredients.

The fact remains, however, that we've yet to "formalize" (or even measure) perfection-seeking to any useful degree. It is still an entirely human process. However, there is some evidence that tools can aid formalization and while doing so improve conceptualization. [19] [20] Specifically, when a user, in this case a physician, is given the task of entering a formal term for a patient "problem," an interface that displays potentially related formal terms in response to the input of a casual term can help the user better conceptualize the concept being entered. Thus, even when the user interface returns an exact equivalent for the casual term, users may choose a "better" formal term from the displayed semantic neighborhood. The simple explanation for this phenomenon is that humans are better at recognition than recall. Those developing ontologies will be familiar with the phenomenon; once domain experts can "see" a domain model they can almost always make it better.

4.5 Architecture, tools and the local enhancement problem

Implicit in much that has been written here is the architectural notion of vocabulary servers, or in this context, formal terminology or ontology servers. That is, such servers "normalize" terminology functions for enterprises, some at Web scale. [See for example the National Cancer Institute's EVS (Enterprise Vocabulary Server) http://ncievs.nci.nih.gov/NCI-Metaphrase.html] We believe that such servers will be essential to the support of the Semantic Web, and as usual on the Web, the challenge will be how to maintain them in loose synchrony as appropriate.

A clear result of experience to date shows that terminology development, especially formal terminology development cannot be undertaken for long without non-trivial supporting tools and software. Foremost among the required tools is a scalable terminology editing workstation, one evolutionary sequence of which was begun by Mays, et al. [21] The fact that formal terminologies will almost always be constructed and maintained by geographically separated domain experts implies additional requirements for "configuration management," conflict resolution, and the like. One approach to these problems is described in [22]. Further, experience in both the U.S. and United Kingdom has shown that the rate-limiting factor for large-scale terminology development is workflow management, rather than the editing work itself.

One short-term reality is the need for what we call "local enhancement." In the healthcare domain, enterprises will have some locally produced, i.e. "locally mixed and prepared," medications for the foreseeable future, and academic medical centers will always have new terms and concepts in substantive use locally. For these and other reasons, an authoritative reference terminology will need to be enhanced locally. The so-called "update paradox" is that those who add the greatest quantity of local enhancements incur the greatest maintenance burden as the external terminology authority evolves. This tradeoff is made more complex by external reimbursement and reporting requirements.

5. Additional exemplars - reference terminologies and semantic messages in healthcare and biomedicine

In response to the shortfalls of current authoritative biomedical terminologies a number of efforts are underway focused on the development of so-called "principled" reference terminologies. For the purposes of this paper the "principles" in question are those that are computer-empowering, indeed the whole point of a reference terminology is to empower computers, particularly, as with the Semantic Web, to empower computer-to-computer interoperation. Several examples are represented in Figure 2.

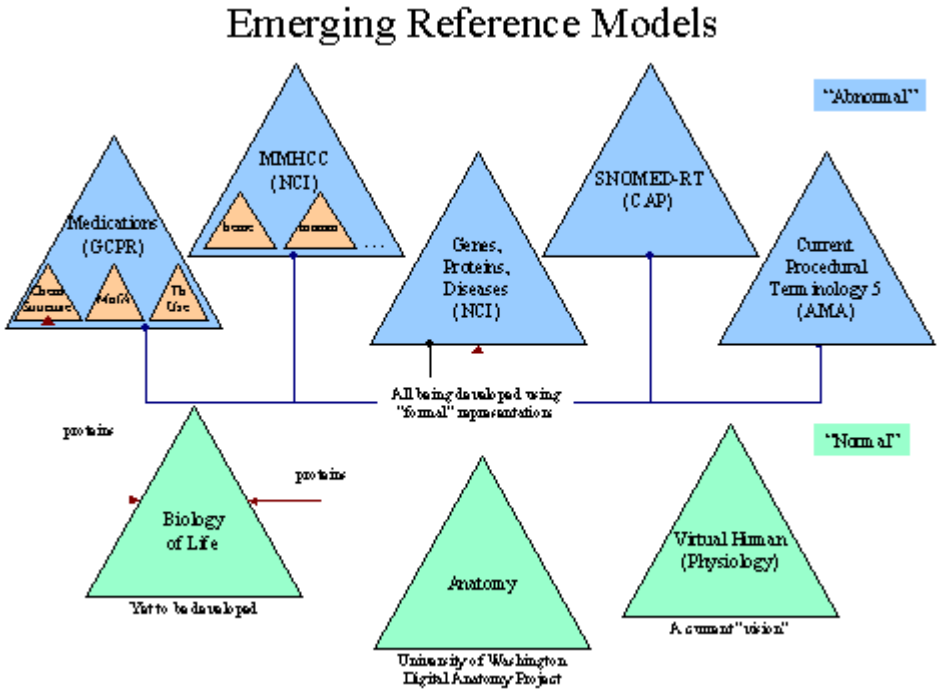


Figure 2 – Emerging Reference Terminologies in Biomedicine: The GCPRMedications reference terminology defined some 1,000 medication active ingredients in terms of Chemical Structure Class, Mechanism of Action, and Therapeutic Use. The NCI MMHCC (Mouse Models of Human Cancer Consortium) is developing detailed diagnostic terminologies for eight organ sites in mice, as a prelude to "certification" of the models as representative of human cancer behavior. NCI is also "modeling" about 250 genes known to be associated with cancer; in particular the association between these genes, the proteins they produce (or do not produce), and diseases is being made explicit. SNOMED-RT is a large (100K+ concept) effort by CAP (College of American Pathologists) and Kaiser Permanente Healthcare to "formalize" SNOMED International (SNOMED = Systematic Nomenclature of Medicine). The AMA (American Medical Association) is formalizing CPT-4 (Current Procedural Terminology). Each of these efforts employs a Description-Logic-based representation. The modular approach implied by this repertoire of reference terminologies in turn creates a need for a reference terminology for Biology that would represent the considerable commonality in, for instance, mice and humans. Similarly, a formal model of human anatomy being developed by Rosse, et al., at the University of Washington may evolve into a reference terminology for vertebrate anatomy as a way to, again, capture inter-species commonality for reuse in other models. A terminology model of Physiology, now being contemplated by some groups, may represent another piece of the "normal" reference model. Not shown is a laboratory testing method terminology being developed by the CDC (Centers for Disease Control and Prevention) .[23]

As recently as a few years ago such a (relative) "explosion" of formal terminology efforts would have been inconceivable. Now such efforts are taking on, in specific domains, the challenge implied by the Semantic Web, namely the development of ontologies for specified domains. Early versions of some of these terminologies are being deployed this year.

HL7, version 3 (http://www.hl7.org/page.cfm?p=524), is a proposed standard for semantic messages in healthcare. It builds on the widely deployed HL7, version 2, standard syntax by using "value sets" taken from external, authoritative, formal terminologies.

6. Summary – healthcare and biomedicine are a rehearsal for the Semantic Web

We are building on our experience with the use of formalization processes for update management in critical working systems. We believe that the challenges we face are specialized equivalents of challenges to be faced by Semantic Web developers as more and more sophisticated systems are deployed and become critical. Among other things these experiences reveal the critical role of process, and that this process needs to be made explicit and intrinsic. We are attempting to fulfill this requirement through the development of an ontology of change, and a recognition that process is more important than representation. If successful, the Semantic Web community may be able to generalize this ontology sufficiently to allow it to be migrated into the "horizontal" Semantic Web infrastructure, and support a "perfection-seeking" Semantic Web.

Acknowledgements

This work was partially supported by Contracts with NLM (National Library of Medicine), NCI (National Cancer Institute), and GCPR (Government Computer-based Patient Record). The GCPR (Government Computer-based Patient Record) medication Reference Terminology Model project team also included Ha Nguyen and Joanne Wong, University of California Berkeley, Munn Maung, University of California Davis, Maung Than, Tun Tun Naing, Apelon, Richard Dixon, MD, FACP and Joe Awad, MD, Vanderbilt. Also contributing were Betsy Humphreys, MLS, William T. Hole, MD, Suresh Srinivasan, PhD, Alexa McCray, PhD, Frank Hartel, PhD, Sherri De Coronado, Robert Cardiff, MD, PhD, Scott Kogan, MD, Mark Erlbaum, MD, David Sperzel, MD, David Sherertz, Brian Carlsen, Cornelius Rosse, MD, DrSc, and Randy Levin, MD, and several others at the FDA (Food and Drug Administration), though as customary they should not be held responsible. Eric Mays, PhD and Bob Dionne from Apelon clarified some of the utility of Description Logic. Important feedback regarding the utility of the model presented in Figure 1 was received from the HL7 Vocabulary Committee and from NCVHS (National Committee on Vital and Health Statistics), including assistance from Jeff Blair, Stan Huff, MD, Christopher Chute, MD, DrPH, James Cimino, MD, Jeff Blair, Ed Hammond, PhD, Michael Fitzmaurice, PhD, Joan Kapusnick-Uner, PharmD, and William Braithwaite, MD. Related material was presented at Pharmacology Grand Rounds Vanderbilt University, Informatics Seminars at the University of Utah, the University of California San Francisco, and the University of California Davis, and also at the Department of Defense, the FDA (Federal Drug Administration), the HL7 (Health Level 7) Vocabulary Standards Committee, and TEPR (Toward Electronic Patient Record) 2001.

References

[1] Berners-Lee, T., "Metadata Architecture," W3C, January 6, 1997.

[2] Berners-Lee, T., Hendler, James, Lassila, Dra, "The Semantic Web," Scientific American, May, 2001.

[3] Tuttle, MS, Information technology outside health care: what does it matter to us? J Am Med Inform Assoc. 1999 Sep-Oct;6(5):354-60.

[4] Kuhn, KA, Guise, DA, "Review: From Hospital Information Systems to Health Information Systems – Problems, Challenges, Perspectives," Haux, R, Kulikowski, C., Editors, *Yearbook of Medical Informatics 2001*, Schattuer, pp 63-76.

[5] Brodie, Michael L., M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach*, Freeman, 1995.

[6] Kohn, L., J. Corrigan, and M. Donaldson, *To Err is Human: Building a Safer Health System*. 2000, Institute of Medicine Report, Washington, DC: National Academy Press, 2000.

[7] Blois, M.S., ***Information and Medicine: The Nature of Medical Descriptions*** (University of California Press, 1984), p. 284.

[8] Carter, J., et al., "The Creation and Use of a Reference Terminology for Inter-Agency Computer-based Patient Records: The GCPR RTM Demonstration Project," Fall AMIA Symposium, 2001 (to be presented).

[9] Cimino, JJ, An approach to coping with the annual changes in ICD9-CM., Methods Inf Med. 1996 Sep;35(3):220.

[10] Tuttle, MS, Nelson, SJ, A poor precedent. Methods Inf Med. 1996 Sep;35(3):211-7. (A companion article to [9] above.)

[11] Michael Stonebraker. ``The POSTGRES Next-Generation Database Management System", ***CACM***, 34(10)78-93, October 1991. (Available on-line at http://www.acm.org/pubs/articles/journals/cacm/1991-34-10/p78-stonebraker/p78-stonebraker.pdf.)

[12] Snodgrass, RT, ed., *The TSQL2 Temporal Query Language*, Kluwer, Boston, 1995. http://www.wkap.nl/kapis/CGI-BIN/WORLD/book.htm?0-7923-9614-6

[13] Snodgrass, RT, *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufman, 1999.

[14] Randy Levin, MD, personal conversation.

[15] Michael Stonebraker, "The Design of the POSTGRES Storage System," Proceedings of the 13th VLDB Conference, Brighton 1987, pp. 289-300. (Available on-line at http://www.informatik.uni-trier.de/~ley/db/conf/vldb/Stonebraker87.html.)

[16] Suarez-Munist, O, et al., "MEME-II supports the cooperative management of terminology. Proc AMIA Annu Fall Symp. 1996;:84-8.

[17] Campbell, KE, Tuttle, MS, Spackman, KA, A "lexically-suggested logical closure" metric for medical terminology maturity. Proc AMIA Symp 1998;:785-9.

[18] Blois, MS, Medicine and the nature of vertical reasoning. N Engl J Med. 1988 Mar 31;318(13):847-51.

[19] Tuttle, MS, et al., "Metaphrase: an aid to the clinical conceptualization and formalization of patient problems in healthcare enterprises" Methods Inf Med. 1998 Nov;37(4-5):373-83.

[20] Cimino, JJ, Patel, VL, Kushniruk, AW, Studying the human-computer-terminology interface. J Am Med Inform Assoc. 2001 Mar-Apr;8(2):163-73.

[21] Mays, E, et al., Scalable and expressive medical terminologies. Proc AMIA Annu Fall Symp. 1996;:259-63.

[22] Campbell, KE, *Distributed Development of a Logic-Based Controlled Medical Terminology* (CS-TR-97-1596). Dissertation available from elib.Stanford.ledu, Stanford: Stanford University.

[23] Steindel, Steven, Granade, SE, "Monitoring Quality Requires Knowing Similarity: The NICLTS Experience," Fall AMIA Symposium, 2001 (to be presented).

Industrial Strength Ontology Management

Aseem Das[1], Wei Wu[1] & Deborah L. McGuinness[2]

[1]*VerticalNet Inc., {adas, wwu}@verticalnet.com*
[2]*Knowledge Systems Laboratory, Stanford University, dlm@ksl.stanford.edu*

Abstract. Ontologies are becoming increasingly prevalent and important in a wide range of e-commerce applications. E-commerce applications are using ontologies to support parametric searches, enhanced navigation and browsing, interoperable heterogeneous information systems, supplier enablement, configuration management, and transaction discovery. Applications such as information and service discovery and autonomous agents that are built on top of the emerging Semantic Web for the WWW also require extensive use of ontologies. Ontology-enhanced commercial applications, such as these and others require ontology management that is scalable (supporting thousands of simultaneous distributed users), available (running 365x24x7), fast, and reliable. This level of ontology management is necessary not only for the initial development and maintenance of ontologies, but is essential during deployment, when scalability, availability, reliability and performance are absolutely critical. VerticalNet's Ontology Builder and Ontology Server products are specifically designed to provide the ontology management infrastructure needed for e-commerce applications. These tools bring the best ontology and knowledge representation practices together with the best enterprise solutions architecture to provide a robust and scalable ontology management solution.

1 Introduction

Ontology Builder and Ontology Server were developed in response to the business needs for ontologies in VerticalNet's e-commerce and B2B applications. They provide a scalable and distributed ontology environment, which is a component critical to the success of e-commerce applications. More broadly, however, this component is also critical to the success of any architecture, which leverages background information, such as the Semantic Web. The next generation web – commonly referred to as the Semantic Web – obtains its power and "intelligence" from utilizing markup information on content sources along with background information on terms and content. The success of such an endeavor relies on environments that support creation and maintenance of background

information, while working in a broadly distributed environment like the web. Ontology Builder/Server provide such an environment in an industrial strength implementation.

Vertical Net currently hosts 59 industry-specific e-marketplaces that span diverse industries such as manufacturing, communications, energy, and healthcare. Each e-marketplace acts as an industry-specific comprehensive resource that provides businesses and professionals with information on products, technology, industry regulations, and news and allows buyers and sellers to exchange information, source, buy, and sell products.

The primary challenge in developing these e-marketplaces is integrating the disparate sources of information in a way that presents buyers with a single, coherent browsing and navigation experience that includes contextually relevant information from all of the available sources. Suppliers have to be able to display their products on the e-marketplace in a way that enables buyers to purchase electronically, even though the suppliers maintain their product databases and availability and price information in their own vocabulary. For example, different suppliers might use the terms *memory device*, *passives*, and *RAM* to refer the same product and have very different internal vocabularies.

The use of ontologies was seen as the best solution not only to solve these particular problems [18, 19], but also to provide a common knowledge infrastructure for other e-commerce applications like service discovery, auctions, and request for proposal. Most of VerticalNet's e-commerce applications are now knowledge-enabled and use ontologies to drive their services.

2 Requirements

An extensive requirement gathering process was undertaken to compile requirements for VerticalNet's ontology management solutions. We identified the following key requirements for ontology management for VerticalNet:

1 Scalability, Availability, Reliability and Performance – These were considered essential for any ontology management solution in the commercial industrial space, both during the development and maintenance phase and the ontology deployment phase. The ontology management solution needed to allow distributed development of large-scale ontologies concurrently and collaboratively by multiple users with a high level of reliability and performance. For the deployment phase, this requirement was considered to be even more important. Applications accessing ontological data need to be up 365x24x7, support thousands of concurrent users, and be both reliable and fast.

2 Ease of Use – The ontology development and maintenance process had to be simple, and the tools usable by ontologists as well as domain experts and business analysts.

3 Extensible and Flexible Knowledge Representation – The knowledge model needed to incorporate the best knowledge representation practices available in the industry and be flexible and extensible enough to easily incorporate new representational features and incorporate and interoperate with different knowledge models such as RDF(S) [2, 15] or DAML [11]/DAML+OIL [8].

4 Distributed Multi-User Collaboration – Collaboration was seen as a key to knowledge sharing and building. Ontologists, domain experts, and business analysts need a tool that allows them to work collaboratively to create and maintain ontologies even if they work in different geographic locations.

5 Security Management – The system needed to be secure to protect the integrity of the data, prevent unauthorized access, and support multiple access levels. Supporting different levels of access for different types of users would protect the integrity of data while providing an effective means of partitioning tasks and controlling changes.

6 Difference and Merging – Merging facilitates knowledge reuse and sharing by enabling existing knowledge to be easily incorporated into an ontology. The ability to merge ontologies is also needed during the ontology development process to integrate versions created by different individuals into a single, consistent ontology.

7 XML interfaces – Because XML is becoming widely-used for supporting interoperability and sharing information between applications, the ontology solution needed to provide XML interfaces to enable interaction and interoperability with other applications.

8 Internationalization – The World Wide Web enables a global marketplace and e-commerce applications using ontological data have to serve users around the world. The ontology management solution needed to allow users to create ontologies in different languages and support the display or retrieval of ontologies using different locales based on the user's geographical location. (For example, the transportation ontology would be displayed in Japanese, French, German, or English depending on the geographical locale of the user.)

9 Versioning – Since ontologies continue to change and evolve, a versioning system for ontologies is critical. As an ontology changes over time, applications need to know what version of the ontology they are accessing and how it has changed from one version to another so that they can perform accordingly. (For example, if a supplier's database is mapped to a particular version of an ontology and the ontology changes, the database needs to be remapped to the updated ontology, either manually or using an automated tool.)

The requirements of scalability, reliability, availability, security, internationalization and versioning were considered to be the most important for an industrial strength ontology management solution.

3 Existing Ontology Environments

Given the above requirements, several existing ontology management environments were evaluated[1]:

[1] The evaluation was done in Fall'99 and hence does not include ontology management environments such as OntoEdit (http://www.ontoprise.de), WebODE (http://delicias.dia.fi.upm.es/webODE/), and OILEd (http://img.cs.man.ac.uk/oil/), which were available for use after Fall'99.

- Ontolingua/Chimaera [6, 16]
- Protégé/PROMPT [10, 20]
- WebOnto/Tadzebao [4]
- OntoSaurus, a web browser for Loom [12] (http://www.isi.edu/isd/ontosaurus.html)

Some of these environments have already been compared based on different criteria than those formulated at VerticalNet [5]. Figure 1, shows a feature set matrix and our evaluation[2] of the tools based on VerticalNet's requirements. To keep the evaluation simple, a three level (+, 0, -) scale was used, where (+) indicates a requirement was surpassed, (0) indicates the requirement was met and (-) indicates that the tool failed to meet the requirement. Although, none of the existing ontology development environments provide all of the required features, they are nevertheless strong in particular features and have different but very expressive underlying knowledge representation models.

	Scalable Available Reliable	Ease of Use	Knowledge Representation	Multi User Collaboration	Security Management	Diff & Merge	International ization	Versioning
Ontolingua/ Chimaera	-	-	+	0	-	+	-	-
Protégé/ PROMPT	-	0	+	-	-	+	-	-
OntoWeb/ Tadzebao	-	0	+	+	-	-	-	-
OntoSaurus/ Loom	-	-	+	0	-	-	-	-

Figure 1: Comparison of Some Ontology Environments

Ontolingua provides a very powerful and expressive representation with its frame language and its support for KIF [9] – a first order logic representation. In combination with its theorem prover (ATP), Ontolingua provides extensive reasoning capabilities and with Chimaera [16], it supports ontology merging and diagnostics. Ontolingua also provides expressive and operational power not found in other environments such as support for generating and modifying disjoint covering partitions of classes.

WebOnto/Tadzebao provides very rich collaborative support for browsing, creating and editing ontologies, together with the ability to collaboratively annotate and hold synchronous and asynchronous ontology related discussions using the Tadzebao tool.

OntoSaurus provides a graphical hyperlinked interface to Loom knowledge bases. Loom provides expressive knowledge representation, automatic consistency checking and deductive support via its deductive engine – the classifier.

Protégé is the easiest to use and supports the construction of knowledge-acquisition interfaces based on ontological data. It also has a component framework for easily integrating other components via plugins. Protégé already provides several plugins including PAL, a first order logical language for expressing constraints, and SMART/PROMPT [20], a tool for merging and alignment of ontologies

[2] This was not a formal evaluation with published, unambiguous evaluation criteria. It was however a good faith effort to evaluate VerticalNet requirements as understood in the various tools.

However, despite their strengths, all of the ontology solutions fell short on the scalability, reliability, and performance requirements, perhaps because industrial strength, commercial scalability was not seen as a important aspect of ontology management since most of the ontology usage until recently has been restricted to research and academia. Also, none of the tools provided security, internationalization, or versioning support – requirements considered critical for e-commerce applications.

After evaluating these solutions against our requirements, we decided to build our own ontology management solution with the goal of bringing the best ontology and knowledge representation practices together with the best enterprise solutions architecture to satisfy the requirements of ontology-driven e-commerce applications.

4 Ontology Builder

Ontology Builder is a multi-user collaborative ontology generation and maintenance tool designed to incorporate the best features of existing ontology toolkits in order to provide a simple, powerful and yet broadly usable tool. Ontology Builder uses a frame-based representation based on the OKBC Knowledge Model [3]. OKBC was developed recognizing the wide general acceptance of frame-based systems [13] and provides an API (Applications Programming Interface) for frame-like systems. Written entirely in Java, Ontology Builder can run on multiple platforms. It is based on the J2EE (Java 2 Enterprise Edition) platform (http://java.sun.com/j2ee), which is a standard for implementing and deploying enterprise applications. Ontology Builder also provides:

- Import and export based on XOL (XML-based Ontology Exchange Language) [14][3]
- A verification engine designed to maintain consistency of terms stated in the language
- A role-based security model for data security and ontology access
- An ontological difference and merging engine

[3] At the time of design and development, a DAML option did not exist. Today there are plans to support DAML+OIL and RDF as well.

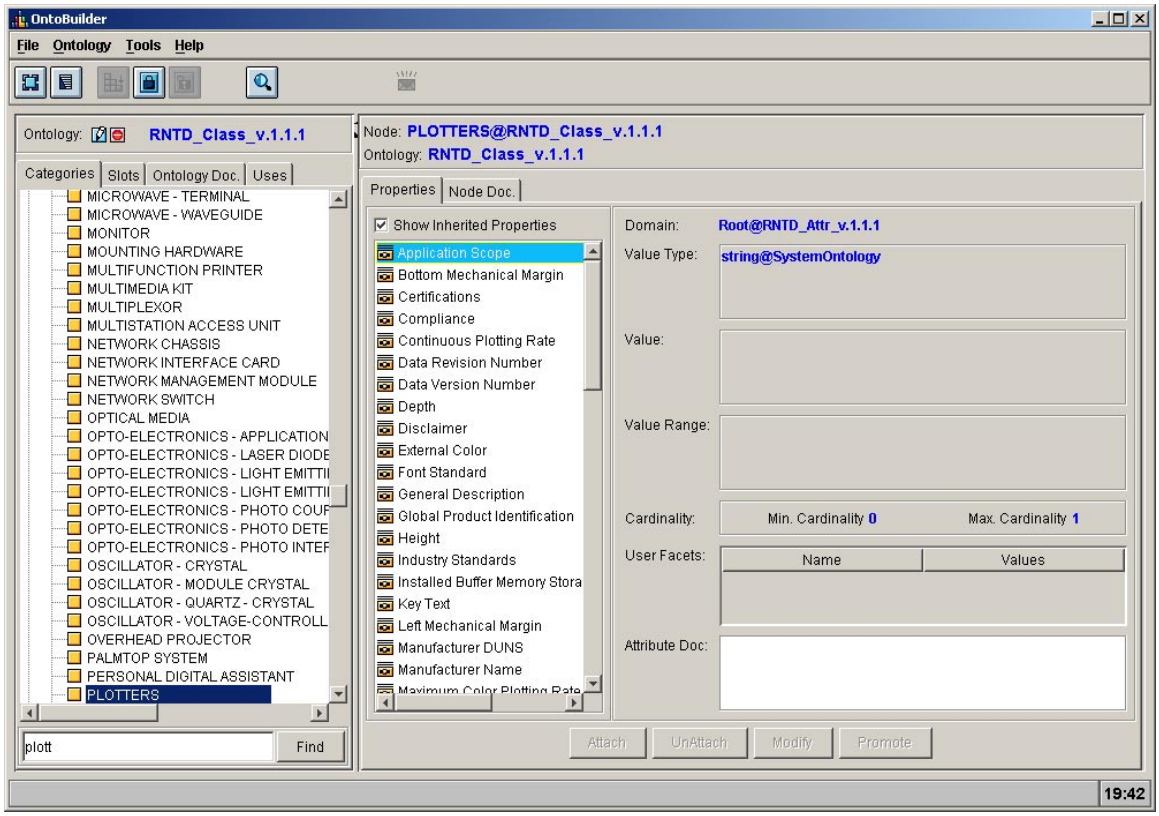


Figure 2: Ontology Builder Main Screen

4.1 Architecture

Ontology Builder is based on the J2EE (Java 2 Enterprise Edition) platform, a standard for implementing and deploying "enterprise" applications. The term "enterprise" implies highly-scalable, highly-available, highly-reliable, highly-secure, transactional, distributed applications. The J2EE technology is designed to support the rigorous demands of large-scale, distributed, mission-critical application systems and provides support for multi-tier application architecture. Multi-tier applications are typically configured to include:

- A client tier to provide the user interface
- One or more middle-tier modules that provide client services and business logic for an application
- A backend enterprise information system data tier that provides data management

The client tier is a very "thin" tier, that contains only presentation logic. The business and data logic are usually partitioned into separate components and deployed on one or more application servers. This partitioning of the application into multiple server components allows components to be easily replicated and distributed across the system, ensuring scalability, availability, reliability and performance.

Central to the J2EE platform architecture are application servers, which encapsulate the business and data logic and provide runtime support for responding to client requests, automated support for transactions, security, persistence, resource allocation, life-cycle management, and as well as lookup and other services.

Ontology Builder uses a 4-tier architecture comprised of a presentation tier, web tier, service tier, and data tier. This architecture, shown in Figure 3, can be deployed using a single application server. The application server encapsulates the service tier, which consists of the business and data logic. A single server can support many simultaneous connections and multiple servers can be easily clustered as needed for scalability, load balancing, and fault tolerance. Within the presentation tier, a client can be either a Java applet or application. The clients have easy-to-use interfaces written using the Java Swing APIs. Both applet and application-based clients communicate with the web tier via the HTTP protocol. The web-tier communicates with the service tier using RMI (Java Remote Method Invocation) (http://java.sun.com/products/rmi-iiop/index.html). The service tier communicates with the data tier through the JDBC (Java Data Base Connectivity) protocol (http://java.sun.com/products/jdbc). Collaboration is implemented using a JSDT (Java Shared Data Toolkit) server (http://java.sun.com/products/java-media/jsdt), which forwards all communication and change events to the respective clients.

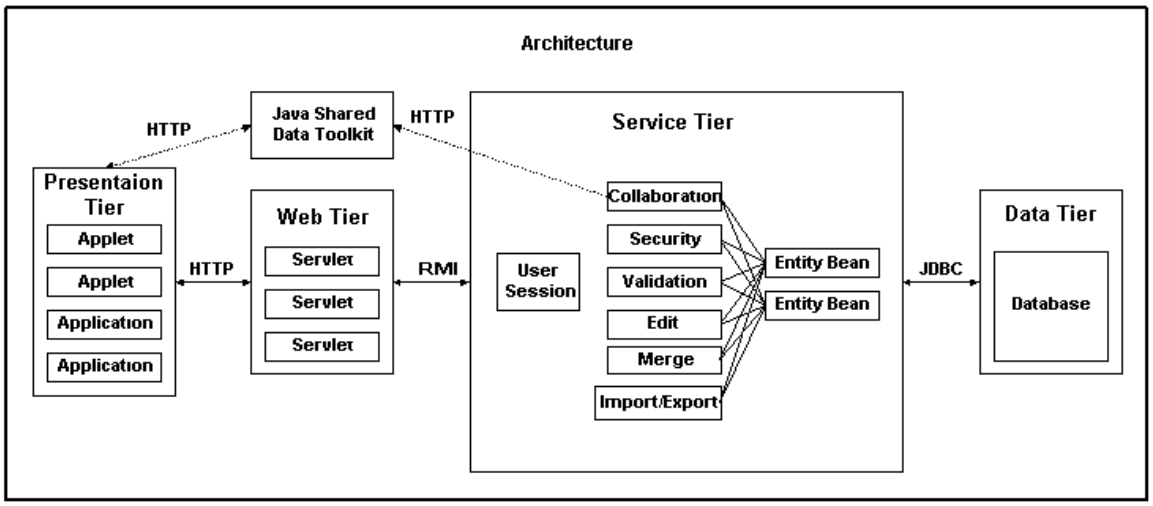


Figure 3: The Architecture of Ontology Builder

4.2 Knowledge Representation

Ontology Builder uses an object-oriented knowledge representation model based on and compatible with the OKBC knowledge model and is designed to use the best practices from other frame-based systems. Ontology Builder implementation supports the OKBC operations on classes, slots, facets, and individuals. Currently, however, no external interfaces are exposed to enable other knowledge systems to use Ontology Builder as an OKBC compliant server. Interoperability, knowledge sharing, and reuse are important goals and our future plans call for making Ontology Builder work as a fully compliant OKBC server.

Ontology Builder supports a metaclass architecture to allow the introduction of flexible and customizable behaviors into an ontology. This could potentially be used for incorporating other knowledge models or extending the existing knowledge model within Ontology Builder. Ontology Builder predefines certain system constants, classes, and

primitives in a default upper ontology, which can be extended or refined to change the knowledge model and behaviors within the system. The main predefined concepts are:

- CLASS - the default metaclass for all classes, CLASS is an instance of itself
- SLOT – the default metaclass for all slots and an instance of CLASS
- T – the root in the default upper ontology (sometimes referred to as THING in other ontologies)
- INDIVIDUAL – the class of ground objects. Operationally, every entity that is not a class is an instance of INDIVIDUAL.[4]
- Predefined slots – slot-minimum-cardinality, slot-maximum-cardinality, slot-value-type, slot-value-range and domain. These are template slots on the class SLOT.
- Predefined facets– minimum-cardinality, maximum-cardinality, value-type, value-range and documentation-in-frame. These define the specific values for the slot as associated with either a class or a slot frame.
- Predefined primitive data types – boolean, string, integer, float, date, etc.

An ontology is composed of **classes**, **slots**, **individuals** and **facets**, which are all implemented as frames. **Ontology** itself is also defined as a frame and contains information such as author, date created and documentation. Both classes and slots support multiple-inheritance in an Ontology Builder ontology.

Classes are all instances of the metaclass CLASS by default, which is changeable by the user. Classes can be instances of multiple metaclasses and they may be subclasses of multiple superclasses.

Slots are defined independently of any class and are instances of the metaclass SLOT by default, which is also changeable by the user. They can also be instances of multiple metaclasses and parent classes. Like classes, slots also support a multiple-inheritance hierarchy. Slot hierarchies can be used to model naturally hierarchical relationships between terms. For example, you might need to model the notion of price along with the subrelations of wholesale-price, retail-price, and discount-price.

Slots can be attached to a class frame or a slot frame, as slots are themselves first-class objects and when attached describe the properties of the frame. A slot can be attached either as a **template** slot or as an **own** slot. Own slots cannot be directly attached to a frame, but are acquired by the frame (class, slot or individual) being an instance of another class. Template slots can be directly attached to either a class or a slot frame. The domain own slot (acquired by a slot frame from being an instance of class SLOT) is useful for limiting the applicability of the slot only to the specified domain class and its subclasses. If a slot does not define a domain, it can be applied to all classes in an ontology. This flexibility is often useful during the early stage of ontology development when the slots used in an ontology are still being refined. Later however, it is often useful to define a domain for slots so that they are only used in specific contexts.

Facets specify the specific values for a slot-class or a slot-slot association. A facet is considered associated with a frame-slot pair, if the facet has a value for that association. The predefined facets (value-type, value-range, minimum-cardinality, maximum-

[4] Note: Slots and facets are instances of CLASSES. Currently, all entities are either CLASSES or INDIVIDUALS but for extensibility, we are not stating that INDIVIDUALS and CLASSES form a covering partition for all things.

cardinality etc.) hold the values given to a slot's own slots (slot-value-type, slot-value-range, etc.) when the slot is associated with a frame. The facet values can only be a specialization of the slot frame's own slot values. For example, if slot *color* is defined to have a slot-value-type of "color", when it's attached to a frame, the value can only be changed to a specialization of "color", "rgbcolor" or "hsvcolor". If the value is changed, then the "value-type" facet will hold the changed value. In addition to predefined facets, Ontology Builder supports the creation and use of **user-defined facets**. A user-defined facet can be created and attached to a slot when the slot is attached to a frame. For example, a user-defined facet might be used to specify whether or not a slot is "displayable".

4.3 Ontology Inclusion (Uses Relationship)

Ontology construction is time consuming and expensive. To lower development and maintenance cost, it is beneficial to build reusable and modular ontologies so that new ontologies can be created and assembled quickly by mixing and matching existing validated ontologies. Both Ontolingua and Protégé have the capability to include ontologies for the purpose of reuse [7, 22]. Protégé allows projects to be included, but the included projects cannot be easily removed and no duplicated names can exist across projects used (included projects plus the current working project) due to the requirement that names must be unique. This unique name requirement in Protégé is limiting because duplicate names occur in practice. Ontolingua provides facilities that allow flexible combination of axioms and definitions of multiple ontologies. Ontolingua eliminates symbol conflicts among ontologies in its internal representation by providing a local name space for symbols defined in each ontology.

Ontology Builder supports concepts reuse and ontology inclusion through the "uses" relationship. The "uses" relationship allows all classes, instances, slots, and facets from the included ontology to be visible and used by an ontology. For example, if ontology A "uses" ontology B, all of the concepts defined in ontology B (classes, instances, slots and facets) can be referenced from ontology A. A class in ontology A can be a subclass of a class in ontology B, and any class in A can use any slots defined in ontology B. The "uses" relationship can be added or removed easily from an ontology. When a "uses" relationship is removed, inconsistencies might exist in the current working ontology because concepts defined in the removed "uses" ontology still are being referenced, even though the ontology is not being used. Changes made to an ontology are propagated in real-time to all ontologies that use that ontology. Although this ensures that the latest concepts are available for use, it might also cause inconsistencies. Verification can be performed to diagnose and identify frames that have inconsistencies

The "uses" relationship is transitive. If ontology A "uses" ontology B, and ontology B "uses" ontology C, then ontology A "uses" ontology C automatically. Ontology Builder also allows cyclical "uses" relationship, that is ontologies A and B can both use each other. Concepts are unambiguously identified by using a globally unique identifier that is generated automatically when a concept is first created; or by using a fully qualified name. A fully qualified name is the concept name concatenated together with the "@" and the ontology name. For example, car@transportation. The fully qualified name is guaranteed to be unique as a concept name is enforced to be to be unique within a

specific ontology and ontology names are unique across all ontologies in the knowledge base. The fully qualified names are used automatically when working with concepts in ontologies other than the ontology where they are initially defined.

4.4 Data Storage and Knowledge-Relational Mapping

Knowledge-base systems traditionally used the computer's main memory for storing the knowledge needed at run-time. The amount of information that can be stored is limited by the available memory and there might be an initial delay in loading all of the entities into memory from a flat file. Moreover, the storing of the knowledge model in flat files is not secure, is error-prone, and quickly becomes unmanageable as the size of the knowledge base increases. Object-Oriented Database Systems (OODS) can also be used to store the knowledge model and provide superior modeling for representing the relations and hierarchies within an ontology. However, when compared to relational DBMS (RDBMS), OODS lack in performance, enterprise usage and acceptance, internationalization support, and other features. RDBMS are still the storage mechanism of choice in enterprise computing when it comes to storing large amounts of performance-critical data. RDBMS can store gigabytes of data, search several million rows of data extremely quickly, and also support data replication and redundancy.

Ontology Builder uses an enterprise-class RDBMS so that very large-scale ontologies and large numbers of ontologies can be stored and retrieved quickly and efficiently. Several other knowledge based systems SOPHIA [1] and an environment for large ontologies motivated by PARKA [23] have also used RDBMS for these and other similar reasons. Ontology Builder currently supports the Oracle 8 and Microsoft SQL Server RDBMSs for data storage.

Ontology Builder employs a sophisticated database schema to represent the OKBC based knowledge model and can support all OKBC-defined operations that could be performed on classes, instances, slots and facets, as well as the operations specified by the OKBC ask/tell interface. The multiple-table database schema also supports internationalization, which permits ontologies to be developed in any language. Multiple translations of the same ontology can coexist in the same database and can be used to view the same ontology in different locales. The schema is normalized; each piece of information is stored in only one location so that modifications to a concept are automatically propagated to all entities that use that concept.

Knowledge-relational mapping is accomplished via a high-performance persistence layer that converts relational data to and from in-memory Java objects that represent the different entities and relationships of the knowledge model. Information retrieval is optimized to retrieve information about multiple concepts via one JDBC database call, which dramatically improves performance. Moreover, a lazy-loading algorithm is used to retrieve information on an as-need basis. For example, when an ontology is first loaded, only the classes and the class hierarchy are loaded; attached slots, slot values, and facet values are only loaded when a user decides to browse or edit a particular class.

4.5 Multi User Collaboration & Locking

Ontology construction is often a collaborative endeavor where the participants in the ontology building process share their knowledge to come to a common understanding and representation of the ontology. These participants might be geographically separated and for collaboration require the ability to hold discussions and view the changes made to the ontology by other collaborators. Ontology Builder provides this type of multi-user collaborative environment. Collaborators can hold discussions individually or in a group and see changes made to the ontology by other collaborators in real time.

Collaboration is implemented via the Java Data Shared Toolkit (JSDT), which provides the communication, messaging, and session management infrastructure for collaboration within Ontology Builder. As they log into the system, each user is registered with the JSDT server in a default "global" discussion room. Messages sent by any user in this discussion room are received by all other current users of the system. Each ontology also defines its own discussion room, which is created the first time any user opens the ontology for browsing or editing. Users who open the same ontology are added to that ontology's discussion room automatically and can see the messages from and collaborate with other users within that ontology's discussion room. A user can also open a private chat session with any other user who is logged on to the system.

Edits to any ontology in the system are broadcasted to all users, regardless of their interest. The change record indicates the type of edit operation, the affected concept and ontology, and the user who performed the action. Figure 4 is a snapshot of the collaboration window that shows the system log and a discussion between collaborators. Any changes to the ontology are committed to the database immediately, so that the changes are available to all other users in real time. An icon is displayed automatically next to the concepts within an open ontology that have been modified by other users, indicating to the user that the information currently displayed in the Ontology Builder client is no longer accurate. The user might already know what has changed based on the discussion with other collaborators or can look in the system messages to see exactly what was changed in the affected concept. An ontology can be refreshed at any point to retrieve the latest state.

Since multiple collaborators can make changes to the same ontology, some kind of locking scheme is necessary to prevent users from overwriting each other's changes. Ontology Builder uses a pessimistic locking strategy that requires an explicit lock to be acquired by a collaborator before any edits are allowed to a concept. Explicitly locking a concept implicitly locks all of the parents and children of the locked concept, preventing other users from editing either the children or the parents of the locked node. Explicitly locking a concept still allows other users to edit the siblings of the locked concept. Locked concepts are shown with a locked icon in all of the clients, indicating which concepts are currently being edited. This locking strategy enables multi-user collaboration and reduces inconsistencies generated from multiple collaborators working on the same ontology.

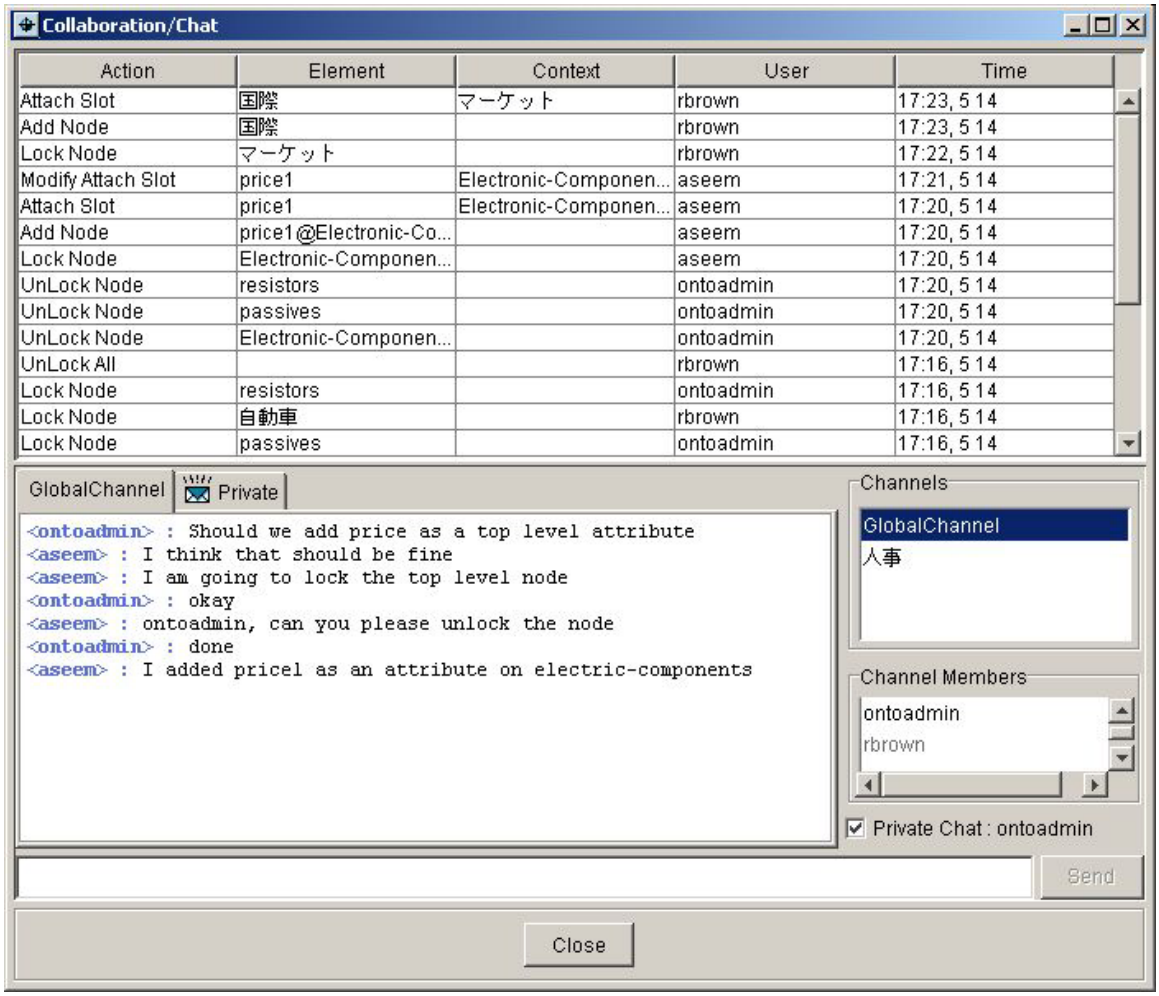


Figure 4: Collaboration Window in Ontology Builder

4.6 Verification

Ontology Builder provides a verification engine to resolve any inconsistencies that might have been introduced during the ontology development and maintenance process. Maintaining consistency is not only critical during the development process where a particular ontology might "use" other ontologies, it is even more critical during the deployment phase where the ontologies have to be valid and consistent so that they can be used by applications without any errors. Real-time verification is a fairly complex task and requires a truth maintenance system (TMS) of some sort in order to have acceptable performance. If a TMS is not used, thorough checks of all of the elements of the ontology need to be done, which is not acceptable from a performance perspective. Ontology Builder does some real-time verification during the edit/creation process itself (for example, it checks for value-type and cardinality violations), but for a full consistency check, the verification engine needs to be explicitly invoked by the user. The verification engine checks for:

- Cycles
- Domain of slots is valid for the classes to which they are attached

- Minimum cardinality <= maximum cardinality
- Minimum cardinality <= num of values <= maximum cardinality
- Values are of specified value-types
- Undefined symbols – symbols that are being used but not defined in the current ontology or any of the ontologies it uses
- Attached slots are consistent with the slot definition (Specialization of value-types, value-ranges and cardinalities is checked for consistency)

4.7 Difference & Merging

Merging ontologies becomes necessary when there is a need to consolidate concepts defined in multiple ontologies, often developed by different teams or gathered from various sources, into a consistent and unified ontology that can be deployed with e-commerce applications. Because the general task of merging ontologies can become arbitrarily difficult, extensive human intervention and negotiation are required. Chimaera [17] and PROMPT [21] provide semi-automated tools to facilitate the merging process. The merging tools in Chimaera and PROMPT suggest a list of merging candidates and present available operations on the candidate frames. Once a user finishes a particular merge operation, more suggestions could be generated and the tool guides the users to finish the merging process. Chimaera also provides diagnostics on the results of merging and other ontology modifications.

Ontology Builder follows a different path in that the initial list of merging candidate frames is not generated. Instead, Ontology Builder relies on the user to decide where to start the merging process. Essentially the user determines when two concepts mean the same thing semantically. The rationale behind the decision is that in practice a user often knows the structures and contents of the ontologies to be merged, and thus has the knowledge to determine where to start the merging process. The goal of the difference and merge service in Ontology Builder is to speed up the merge process once the initial merging candidate frames have been chosen, rather than being a general-purpose merging tool like those provided by Chimaera and PROMPT.

In Ontology Builder, the merge operation does not generate a third ontology that contains the merged results from two input ontologies. Instead, Ontology Builder defines a base ontology and merge ontology where the differences between the two ontologies can be initially identified and then, if desired, the differences can be merged into the base ontology.

Ontology Builder currently has a simplistic algorithm for reporting the differences between two ontologies. Differences are reported for the two concepts selected for comparison as well as for their children that have matching names. If there are no matching names, the differencing stops. Ontology Builder reports the following differences:

- Missing children/parents
- Missing slots
- Value, value-type, value-range, domain, documentation, and cardinality differences for matched concepts

If desired, the differences can be merged. The merge operation

- Copies missing children recursively to the base ontology
- Copies missing slots to the base ontology
- Merges documentation, slot values, value-types, value-ranges and cardinalities for the matched concepts

The difference and merge feature of Ontology Builder is simple compared to the merging features available in other tools like PROMPT or Chimaera, but future plans call for enhancing this functionality based on further requirements and proposed usage.

4.8 Role Based Security

Ontology Builder provides a flexible security model designed to allow client access to the back-end services. Every user has an account on the system and is only allowed to access the back-end services if properly authenticated. Each user is assigned a role, which denotes the level of access for ontology management. Users assigned a particular role can only perform the operations allowed by that role, however, users can be assigned different roles for different ontologies. The security model also enables a much finer-grained permissions system where individual edit operations in an ontology (such as modify-documentation) can be enabled for particular users.

By protecting ontology data and controlling access to back-end services, Ontology Builder's security model meets one of the critical requirements for enterprise class applications.

4.9 Internationalization

Ontology Builder is fully internationalized and can support the browsing and editing of ontologies in multiple locales. A single representation of the ontology is maintained for all locales. Names from each of the locales are linked to this one representation so that changes in ontology structure in one locale are propagated and available in all the other locales. Concepts, which have not been translated in a particular locale, are shown in the locale in which they were initially created. For example, if the ontology was initially created in English and then partially translated into Japanese, browsing it in Japanese will show the names in English for the concepts that have not yet been translated. Ontology Builder also provides support for translating from one locale into another locale. Hooks are provided to use a translation tool or service if desired to semi-automate the translation process. The snapshot in Figure 5 shows a Japanese ontology with some untranslated words in English and French.

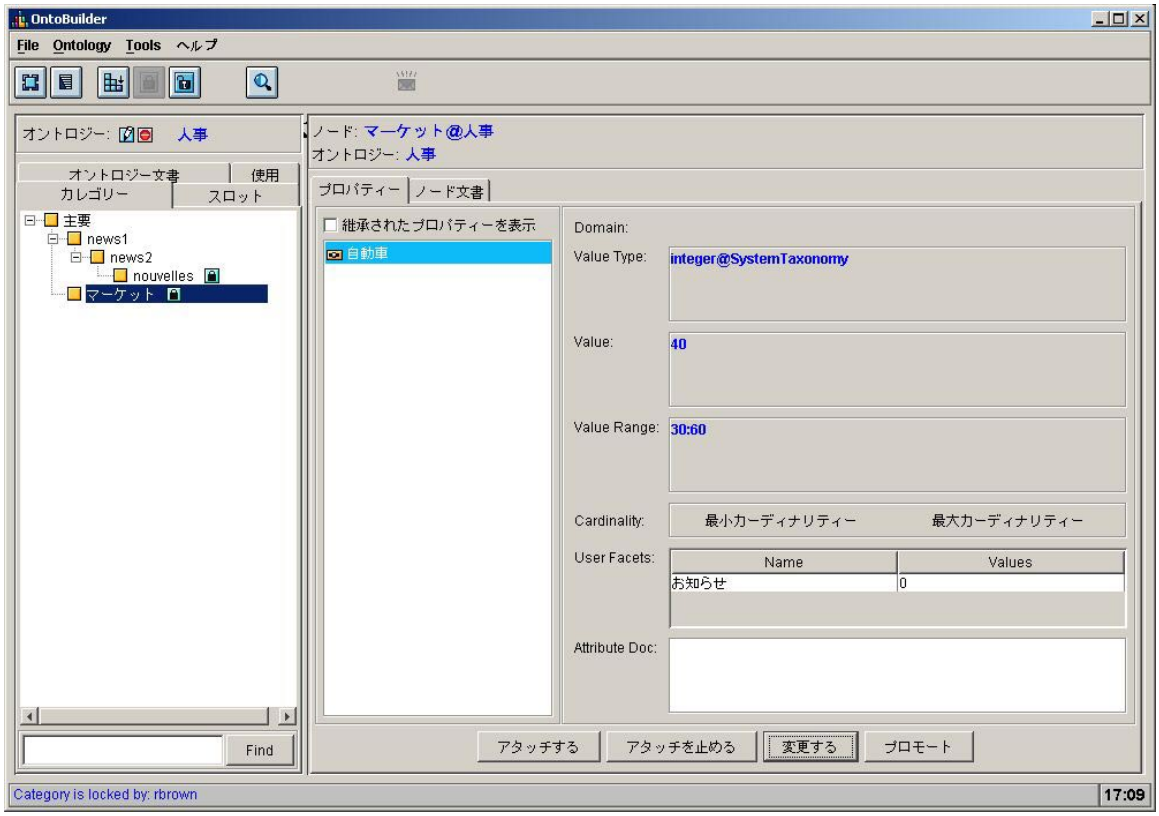


Figure 5: Ontology creation in Japanese

4.10 Import & Export

Ontology Builder provides import and export functionality based on XOL (XML based Ontology Exchange Language) [14]. XOL is based on OKBC-Lite, a simplified form of the OKBC knowledge model, and is "designed to provide a mechanism for encoding ontologies within a flat file that may be easily published on the WWW for exchange among a set of application developers." In Fall' 99, when the decision to use XOL was made, XOL was considered to be an emerging standard for exchange and publication of ontologies. Since, then other ontology representation and exchange standards such as RDF and DAML+OIL have emerged and we plan to support these standards in the near future. The XOL DTD used by Ontology Builder has been modified to support internationalization, metaclass, uses, and facet definitions, which are not part of the original DTD.

5 Ontology Server

Ontology Server is a scalable, high-performance server and is a critical component for e-commerce applications that require ontologies to drive their services. It provides a very scalable, available, reliable, and high-performance solution. Ontology Server uses exactly the same architecture and representation as Ontology Builder and provides XML and Java RMI interfaces for access to the ontological data. It is optimized for read-only access, which facilitates the use of data-caching mechanisms to enhance performance,

which is critical for e-commerce applications. Ontology Server defines its own interfaces, which are simpler and more suitable for e-commerce applications than the general OKBC interface.

6 Usage & Performance

Ontology Builder was released internally for use by VerticalNet ontologists and domain experts in April 2000, following a beta release in February 2000. The server - a Sun Ultra 1/60, 1 Gigabyte of RAM, with Oracle 8.0.4 - is hosted out of Palo Alto and accessed mainly from Horsham, Pennsylvania but it is also accessed from several other locations. Over the past year 84 different users have created 974 ontologies on the server. Concurrent usage peaked at about 20 users using the system at one time. The current database has over 5 million records, consisting of 650,000 classes, 480,000 slots, 680,000 frame-slot relations, 220,000 frame-slot-facet relations, 650,000 parent-child relations and 1,100,000 meta-class relations.

Ontology Builder and Ontology Server both use the same architecture and back-end services. However, Ontology Server is optimized for read-only access to the ontological data and gives better performance than Ontology Builder for read operations. Figure 6, shows the performance graph for read operations for Ontology Server. 25 to 1000 clients were simulated accessing 100 different frames, each frame being accessed by each client 100 times. The performance tests were done on a Windows 2000 Pentium III (800 mHz) machine with 512 megabytes of RAM, using SQLServer 2000 default configuration without any tuning. Multiple clients were simulated using multiple threads on a Windows 2000 Pentium III (800 mHz) machine. The performance data is given for **average response time** - the time experienced by a client to retrieve a frame, including server processing time, networking delay, lookup and Java serialization/deserialization and for **overall requests per second** – the number of frame accesses per second or the server throughput.

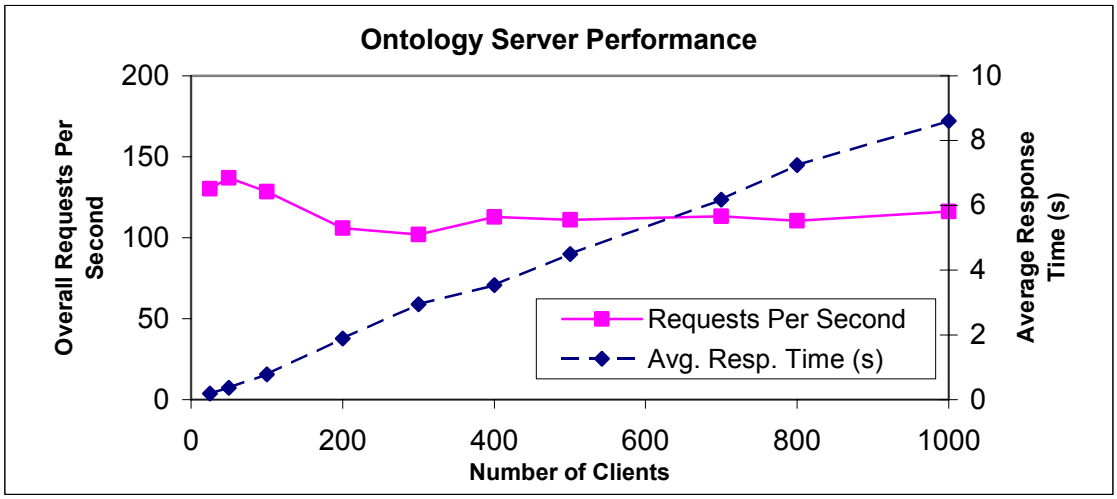


Figure 6: Performance graph for Ontology Server

The graph shows that as the number of clients increases, the throughput remains almost the same but the average response time increases, as now clients have to wait for previous requests from other clients to complete. The average response time for 200 users is about 2 seconds, but as the number of users increases the response time gets much longer, which may not be acceptable. To allow a more scalable solution multiple servers can be clustered together to handle thousands of users concurrently with a reasonable response time. The choice of application server can also significantly impact the response time and the server throughput as some application servers provide better performance and scalability than others. The choice of database and fine-tuning of the database can also increase performance and scalability.

Excluding the networking, serialization and lookup time, Ontology Server's **actual processing time** is only 1-3 milliseconds and does not vary significantly with the number of clients, once the frame has been initially loaded from the database. The initial loading time is about 20–250 milliseconds for each frame, depending on the number of slots, facets, class, parents, children and metaclass relations to be retrieved. Once retrieved, the application server caches the frame and subsequent requests to retrieve that frame take only 1-3 milliseconds regardless of the client requesting the frame. The number of frames to be cached can be specified as a parameter. Frames not being accessed for a while are cached out and replaced with the newly requested frames as the caching limit is reached. Since, all of our tables use primary keys, the size of the database and tables does not significantly increase the initial loading time of the frame. Figure 7, shows the access time in milliseconds for retrieving a bare frame (with no relational information) from the frame table with different sizes.

Num. Of Rows	Min. Time	Max. Time	Avg. Time	Iterations
1000	3.12	14.45	7.2	200
10,000	3.84	17.12	7.75	200
100,000	3.23	15.78	9.35	200
1,000,000	4.52	19.35	11.85	200

Figure 7: Access time for retrieving from database table with different sizes

Ontology Builder does not use caching for retrieving ontological data, but uses lazy loading to retrieve information as needed. Each piece of information is retrieved from the database every time it is requested. For the same machine configuration as described above, the **average response time** to retrieve a simple frame with parents, children, metaclasses and slots (without slot values and frame-slot-facets) is about 50 milliseconds, which translates into 20 read transactions per second. The average time to create a simple frame in Ontology Builder is about 35 milliseconds, which translates into 30 write transactions per second. In practice this level of performance for Ontology Builder has proved to be acceptable, as the ontology development and maintenance is not a performance intensive process. Clustering multiple servers, choice of application server and tuning the database can further improve Ontology Builder's performance.

7 Discussion

Ontologies are becoming much more common as a core component of e-commerce applications. Industrial strength solutions are needed and, in fact, critical for the success and longevity of these applications. We have presented two Vertical Net products: Ontology Builder and Ontology Server. We believe these products bring together the best knowledge management and ontology practices and the best enterprise architectures to provide industrial-strength solutions for ontology creation, maintenance, and deployment.

When evaluated against our initial product requirements, Ontology Builder and Ontology Server meet or surpass most of the requirements. Figure 8, shows this evaluation and compares Ontology Builder with the ontology environments compared in Figure 1. Even though we have provided reasonable solutions to most requirements, designated by a 0, we believe there is still considerable room for improvement and plan to continue to enhance functionality in these particular areas.

	Scalable Available Reliable	Ease of Use	Knowledge Representation	Multi User Collaboration	Security	Diff & Merge	Internationalization	Versioning
Ontolingua/ Chimaera	-	-	+	0	-	+	-	-
Protégé/ PROMPT	-	0	+	-	-	+	-	-
OntoWeb Tadzebao	-	0	+	+	-	-	-	-
OntoSaurus/ Loom	-	-	+	0	-	-	-	-
Ontology Builder	+	0	0	0	0	0	+	-

Figure 8: Comparison of Ontology Builder with other Ontology Environments

We believe we have delivered a robust solution for our most critical requirements – scalability, availability, reliability and performance. By using an enterprise architecture (J2EE) and an enterprise RDBMS as the back end storage, we have provided an enterprise-class scalable, reliable, available, and high-performance ontology management solution.

The Ontology Builder client provides an easy-to-use interface for ontologists, domain experts, and business analysts. Though, we have not done formal usability studies, many domain experts and analysts have been able to use the tool productively, with a minimum of training. However, we believe, there is always room for improvement in user-interface design and usability and we plan additional work on usability in response to user studies and needs analysis.

Our knowledge model is based on the OKBC knowledge model and provides flexibility and extensibility for incorporating new features and existing knowledge models. However, Ontology Builder does not support axioms yet and does not include a full reasoning component. While we do support internal consistency checking and propagation of implicit information, we do not provide an OKBC interface and thus do not support full OKBC compliance. We plan to extend our knowledge model to support axiomatic reasoning and also plan to implement an OKBC interface. Our current

import/export format is XOL, future plans include support for other common formats such as RDF and DAML+OIL.

We have provided a multi-user collaborative environment to facilitate the ontology building, sharing, and maintenance process. Collaborators can hold discussions and see changes committed by other users. The collaborative environment could be further improved by providing optimistic locking (where a frame is not allowed to be edited, only when it is being updated) instead of pessimistic locking. We are also investigating a more complete conferencing and whiteboarding solution, perhaps by integrating a third party tool like Microsoft NetMeeting (http://www.microsoft.com/windows/netmeeting/default.asp) or Netscape Conference (http://home.netscape.com/communicator/conference/v4.0).

Our role-based security model provides data security, data integrity, user authentication and multiple levels of user access. A fine-grained model in which a set of permissions could be assigned to a user of a particular ontology has also been designed.

The difference and merging engine currently uses a simple algorithm. Future plans call for a more sophisticated difference and merging algorithm

Ontology Builder is fully internationalized and can be used in multiple languages and ontologies can be created and displayed in multiple locales.

Ontology Builder currently does not provide any versioning support. Versioning of ontologies is needed so that changes from one version to another can be tracked and managed and so that applications can determine what specific version of an ontology is being accessed. We hope to provide fine-grain versioning control functionality in the future.

8 Acknowledgements

We like to thank the many people who have contributed to these products - Mark Yang for design, Howard Liu, Don McKay, Keith Thurston, Lisa Colvin, Patrick Cassidy, Mike Malloy, Leo Orbst, Eric Elias, Craig Schlenoff, Eric Peterson for their use of the products and feedback, Joel Nava, Faisal Aslam, Hammad Sophie, Doug Cheseney, Nigel McKay for implementation and Hugo Daley,Adam Cheyer for their support.

9 References

[1] Neil F. Abernethy, Russ B. Altman, “SOPHIA: Providing basic knowledge services with a common DBMS”, *Proceedings of the 5th KRDB Workshop*, Seattle, WA, 1998.

[2] Dan Brickley & R.V.Guha, "Resource Description Framework (RDF) Schema Specification 1.0", World Wide Web Consortium, Cambridge, MA, 1999

[3] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter Karp, James Rice, “Open Knowledge Base Connectivity 2.0”, Knowledge Systems Laboratory, 1998.

[4] J. Domingue, “Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on theWeb”, *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada, 1998.

[5] J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa & V. R. Benjamins, “WonderTools? A comparative study of ontological engineering tools”, *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada, 1999.

[6] Adam Farquhar, Richard Fikes, James Rice, “The Ontolingua Server: a Tool for Collaboartive Ontology Construction”, *International Journal of Human-Computer Studies,* 46, 707-727, 1997.

[7] Adam Farquhar, Richard Fikes, James Rice, “Tools for assembling modular ontologies in Ontolingua”, Knowledge Systems Laboratory, Stanford University, April, 1997.

[8] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, and Peter F. Patel-Schneider. ``OIL: An Ontology Infrastructure for the Semantic Web ". In IEEE Intelligent Systems, Vol. 16, No. 2, March/April 2001.

[9] Michael Genesereth and Richard Fikes, “Knowledge Interchange Format, Version 3.0 Reference Manual”, Knowledge System Laboratory, Stanford University, 1992.

[10] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, & M. A. Musen, “Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000)”. *Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management*. Banff, Alberta, 1999.

[11] James Hendler and Deborah L. McGuinness, ``The DARPA Agent Markup Language". *IEEE Intelligent Systems*, Vol. 15, No. 6, November/December 2000, pages 67-73.

[12] ISX Corporation (1991). "LOOM Users Guide, Version 1.4".

[13] Peter D. Karp, "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center, 1992.

[14] Peter D. Karp, Vinay K. Chaudhri, and Jerome F. Thomere, "XOL: An XML-Based Ontology Exchange Language," Technical Note 559, AI Center, SRI International, 1999.

[15] Ora Lassila & Ralph Swick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation 22 February 1999, World Wide Web Consortium, Cambridge (MA); available on-line as http://www.w3.org/TR/REC-rdf-syntax/.

[16] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder, "An Environment for Merging and Testing Large Ontologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning,* Breckenridge, Colorado, April 2000.

[17] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder, "The Chimaera Ontology Environment", *Proceedings of the The Seventeenth National Conference on Artificial Intelligence,* Austin, Texas, July 2000.

[18] Deborah L. McGuinness ``Ontologies and Online Commerce". In IEEE Intelligent Systems, Vol. 16, No. 1, January/February 2001, pages 8-14.

[19] Deborah L. McGuinness. "Ontologies Come of Age". To appear in D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (editors). Semantic Web Technology, MIT Press, Boston, Mass., 2001.

[20] Natalya F. Noy & Mark A. Musen, "SMART: Automated Support for Ontology Merging and Alignment", *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada, July 1999.

[21] Natalya F. Noy & Mark A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", *Seventeenth National Conference on Artificial Intelligence*, Austin, Texas, 2000.

[22] Protégé Users Guide, http://www.smi.stanford.edu/projects/protege/doc/users_guide/index.html

[23] Kilian Stoffel, Merwyn Taylor, James Hendler, "Efficient Management of Very Large Ontologies", *Proceedings of American Association for Artificial Intelligence Conference*, (AAAI-97), AAAI/MIT Press 1997.

OntoMap – the Guide to the Upper-Level

Atanas K. Kiryakov, Marin Dimitrov
OntoText Lab., Sirma AI EOOD
Chr. Botev 38A, 1000 Sofia, Bulgaria, {naso,marin}@sirma.bg

Kiril Iv. Simov
Linguistic Modelling Laboratory, Bulgarian Academy of Sciences
Akad. G. Bontchev Str. 25A, 1113 Sofia, Bulgaria, kivs@bgcict.acad.bg

Abstract. The upper-level ontologies are theories that capture the most common concepts, which are relevant for many of the tasks involving knowledge extraction, representation, and reasoning. These ontologies use to represent the skeleton of the human common-sense in such a formal way that covers as much aspects (or "dimensions") of the knowledge as possible. Often the result is relatively complex and abstract philosophic theory.

Currently the evaluation of the feasibility of such ontologies is pretty expensive mostly because of technical problems such as different representations and terminologies used. Additionally, there are no formal mappings between the upper-level ontologies that could make easier their understanding, study, and comparison. As a result, the upper-level models are not widely used. We present OntoMap — a project with the pragmatic goal to facilitate an easy access, understanding, and reuse of such resources in order to make them useful for experts outside the ontology/knowledge engineering community.

A semantic framework on the so called conceptual level that is small and easy enough to be learned on-the-fly is designed and used for representation. Technically OntoMap is a web-site (http://www.ontomap.org) that provides access to upper-level ontologies and hand-crafted mappings between them. Currently it supports just online browsing and DAML+OIL export. The next step will be to provide the resources in various formats, including an application server giving an uniform access to the resources via OKBC. This way OntoMap will become part of the semantic web, i.e. machine-understandable rather than just human-readable.

1 Introduction

The structure of the paper is as follows: the next section makes an introduction to the Upper-Level Ontologies, including a principal comparison to the domain-specific ones, discussion on their relations with the lexical knowledge bases and some incompatibility issues. Section 3 discusses the representation languages and primitives with subsections about the most significant paradigms and a short overview of the approaches for their unification. Section 4 describes the primitives used in the OntoMap project – the OntoMapO ontology. Section 5 focuses on the OntoMapO methodology for mapping concepts between ontologies. More technically section 6 enumerates the formats in which OntoMap will provide all the ontologies and mappings. The initial set of ontologies to be hosted and the mappings between them are discussed in section 7. The next section 8 demonstrates the OntoMap usability with a

sample snapshot. Section 9 provides an idea about the services that could be provided on top of the results of the project. The next section concludes the paper.

2 Upper-Level Ontologies

The upper-level ontologies capture mostly concepts that are basic for the human understanding of the world. They are ”grounded” in (supported by, wired to) the common sense that makes it hard to formalize a strict definition for them. They represent the so called prototypical knowledge.

For example, what should be a formal KL-ONE-style or Frame-style definition of a ”table”. Most of the tables have 4 legs, however, there are pretty obvious exceptions for tables with three legs, single leg or even without anything to be considered as a leg. There could be also a ”serious” table with 6 legs. What should be the minimum and maximum cardinality for the slot/role leg? And what should be the type restriction? This is the reason to have most of the upper-level concepts being primitive in KL-ONE terms – they can only have partial definitions, some necessary conditions that involve other partially defined concepts. This is the practical reason to have the upper-level ontologies (for example, Upper Cyc Ontology, SENSUS, MikroKosmos) defined mainly in terms of taxonomic relations. An attempt to strongly use attributes in their definitions could be hard, expensive, and usually leads to involvement of default reasoning or other similar mechanisms that cause intractability.

2.1 Domain-Specific vs. Upper-Level Ontologies

This pseudo-dilemma seems to be mostly a question of goal and scope of the developers of the ontology rather than a representational or management problem. Of course, there exists a significant real difference between the two types of ontologies. The domain-specific ontologies that are trying to capture, for example, a market segment or certain scientific area typically consist of well-defined concepts. For example, in the natural sciences (Mathematics, Physics, Chemistry, Biology, Medicine) the knowledge is usually easy to formalize because it is more or less systematic — it could be expressed using well-defined scientific terms. In such cases, the objects in the universe of discourse are either purely abstract either they are some idealized/simplified models of the real phenomena in the world. For instance, a triangle is nothing more than a polygon with three angles.

2.2 Lexical Knowledge Bases

The so-called lexical knowledge bases (LKB, such as WordNet) are lexicons, thesauri, or dictionaries that attempt to formalize the lexical semantics — the meanings of the words in one or more natural languages. Similar to the upper-level concepts, the meanings of the words are grounded in the common understanding of huge populations — there are no formal definitions, the words can bear a number of different meanings often based on associations, typical uses, collocations, and prototypical knowledge. Going further, the meanings of many words are just primitive concepts. Historically the LKBs and the upper-level ontologies seriously influenced each other. Some upper-level ontologies were developed on the basis of a LKB — such example is the SENSUS ontology ([10]). Other upper-level ontologies were developed

in order to give formal semantics to a LKB — such an example is the EuroWordnet Top Ontology, [15]. This is the reason to have a number of LKB semantic resources included in the initial set of ontologies to be hosted in OntoMap.

2.3 Philosophical Diversity

The existence of several upper-level ontologies that disagree on the most basic concepts about the entities in the world demonstrates a significant philosophical diversity. The practical goals OntoMap project is after seem to require clarification of these basic discrepancies. Which properties of the entities in the world are the most basic ones? What follows from different choices on this level? On which level of generality the differences disappear if they disappear at all? For example, the top of the MikroKosmos ontology (see [14]) demonstrates a typical top-level:

```
ALL
    PROPERTY
        ATTRIBUTE
        RELATION
    OBJECT
        SOCIAL-OBJECT
        PHYSICAL-OBJECT
        MENTAL-OBJECT
        INTANGIBLE-OBJECT
    EVENT
        SOCIAL-EVENT
        PHYSICAL-EVENT
        MENTAL-EVENT
```

However, it ignores the stuff/object (countable) distinction that is very basic in Cyc (see [2]) and other upper-level ontologies.

Our understanding is that OntoMap should not try to choose the best upper-model or to produce a new one. The upper-level have to be chosen according to the specific application – we just want to make the comparison easier.

2.4 Some Disadvantages of the Automatic Mapping

Automatic mapping or merging of ontologies is not involved in OntoMap — we start with the assumption that there either exist some hand-crafted mappings, or such can be developed. Even though it seems that automatic mapping could reduce the efforts, the typical heuristics employed (see [11], [1]) can have a very limited role in the case of upper-level ontologies because of a number of reasons:

- there is relatively small number of upper-level resources, because they are complex, expensive, and (potentially) more reusable than the domain-specific ones;
- they are more complex than the domain-specific ones, because they handle more abstract and partially defined concepts. Much of the semantics is represented just as a free text gloss, rather than as a formula in some knowledge representation formalism. So, in many

cases the equivalence (or mapping) between two concepts could be judged only by interpretation of the glosses;

- the mappings between upper-level ontologies are more re-usable because the ontologies are more reusable;
- the quality of the mapping is extremely important because a mistake in the upper-levels of an ontology can have terrible effect on the lower levels.

3 Unified Representation Needed

In order to provide a uniform representation of the ontologies and the mappings between them, a relatively simple meta-ontology (let us call it OntoMapO) of property types and relation types should be defined. Before presenting OntoMapO we will make an overview of the related problems and approaches.

3.1 Terminological Diversity

There are number of different notions (or terminologies) that are currently used in knowledge management community. The differences (both phraseological and conceptual) are rooted in the main paradigms in the knowledge representation. Here is a non-exhaustive overview of the most popular ”languages” used by the ontologists:

- **concepts, relations, properties** — these are usually the terms inspired by the early semantic networks (let us use the abbreviation **SemNet** below), mathematics and philosophy. *Concepts* are used to express any kind of static and cognitively autonomous semantic phenomena. They classify the entities in the domain of discourse — each entity either belongs to a certain concept’s interpretation, either not. In other words, the information carried out by the concept is either true for the entity either not. The entities that belong to the interpretation of the concept are called *instances* of the concept. Typical concepts are Person, Food, Meeting, Idea. The *properties* come to represent characteristics, aspects, or attributes of the entities, as well, as relations between them. They are further separated into *attributes* and *relations*. Typical representatives are: color, gender (attributes); loves, causes (relations).
- **classes, slots, facets and frames** — obviously, this is the frame-based terminology (to be referred as **Frames** below). Here *classes* correspond to the concepts while the notion for the instances remains the same. The *slots* (especially as they evolved in the last years) correspond to the properties. Slots are further distinguished into *template-slots* (class- or concept-attributes in SemNet) and *own-slots* (instance-attributes in SemNet). The template-slots are defined on a class level — for example, Color is a template-slot for the class Car. In contrast, own-slots connect some values of the template-slots to certain instances of the class, say Colour(Ferarri, Red). *Facets* are properties of the slots, for example, Domain, Range, Min-Cardinality, Documentation. Formally speaking, they are own-slots of the slots. There is no clear favorite for a single term corresponding to the facet notion in the rest of the paradigms;

- **concepts, roles, individuals** — this is the terminology used in the so called description logics (DL), the descendants of the KL-One knowledge representation language (CLASSIC, LOOM, KRIS, SHIQ). This paradigm is pretty close to the one used in the semantic networks. Strictly speaking, it is developed to make them more precise on the epistemological level. The *roles* correspond to the properties, the *individuals* correspond to the instances;

- **classes, objects, attributes** - this is the terminology used in the **object-oriented** paradigm (OO), mostly popular for the purposes of the software engineering. The *classes* correspond to the concepts while the *attributes* (also called data members) correspond to properties. Equivalent of the class-attributes are the *static data members*. The *objects* are always instances of certain class;

- **collections, individuals, predicates, constants** - this is the terminology used by the *cyclists*, the people developing the Cyc knowledge base at Cyc Corp. Roughly, the *collections* correspond to concepts, *individuals* to the instances, and binary predicates (that are kind of collections themselves) correspond to properties. The *constants* are names of collections, individuals, or predicates.

3.2 The Conceptual Level

There are number of attempts to resolve the terminological diversity by managing ontologies in a representation-independent fashion on the so called knowledge-level or conceptual level. Two of the most popular approaches are reported in [4] (ODE) and [12] (OntoEdit). Even sticking to the frame-based terminology the knowledge-model of Protégé-2000 (see [13]) is also a good example for a self-contained and well designed conceptualization that provides sufficient expressive power to capture ontologies encoded in different languages.

A comprehensive classification of the different kinds of properties is reported in [7] — according to different combinations of the meta-properties *identity*, *rigidity* and *dependence* it introduces seven different notions corresponding to ”Concept” in ODE. The primitives used in Cyc (see [2] and the previous sub-section) are interesting at least because the approach is proven in a really large-scale knowledge base.

4 OntoMapO: The OntoMap Primitives

OntoMap is trying to use the minimal useful set of primitives. We were led by the understanding that the oversimplification is not that fatal for the overall usability as a complex system of primitives could be. We tried to design OntoMapO to capture most of the semantics usually encoded in upper-level models. The guideline was to give access to 80% of the ”knowledge” content of of the upper-level ontologies within 20% of the complexity of the representation needed to capture all of it.

We developed a minimalistic meta-ontology that is also as self-describing as possible. Thus, most of the primitives are defined just in terms of the rest of the OntoMapO primitives.

OntoMapO ontology could be also seen as a language. A simple language that provides some expressive power via single kind of expressions – binary relations between concepts. We are intentionally not providing specific syntax in order to keep it as representation independent as possible. Further in this paper we will use a LISP-like syntax to serialize the

relations, however, it is obvious that many other notations (say XML) could perform equally well.

4.1 Comparison with Other Approaches

We will try first to give an impression about OntoMapO by quickly comparing it to two well known approaches for ontology representation.

4.1.1 OntoMap vs. Ontolingua

Each concept is represented in Ontolingua with some twenty slots, many of which are not obvious for people that do not understand frames. For example, if somebody wants to understand the definition of `Corporation` in the Enterprise Ontology as it is represented in Ontolingua (see [18] and [17]) s/he has to bother about the meaning of slots like `Set-Cardinality` and `Relation-Universe`.

We undertake an approach opposite to the one employed in Ontolingua, [5], following the rationale that even though many distinctions could be clearly defined in Ontolingua the most of the semantic-model developers cannot understand them. Our vision is that the database designers, for example, should not be expected to learn complex frame-based theories.

4.1.2 OntoMap vs. RDFS and DAML+OIL

OntoMapO is much similar to the RDFS. The equivalent for `rdfs:Class` is `Concept` in OntoMap and again there are two basic relations: instantiation and inheritance (see subsection Instantiation in Addition to Inheritance). An equivalent of `rdf:Property` in OntoMap is `BinaryRel`.

We will try to outline just the major differences:

- in OntoMap `rdfs:Resource` is missing, actually there is no difference between classes and resources – they are both considered as concepts (resp. `rdfs:Class` and `Concept`).
- as a consequence the property types (resp. `rdf:Property` or `BinaryRel`) are considered as a sub-class of the concepts.
- there is no special relation for `rdfs:subPropertyOf` – the sub-class relation (resp. `rdfs:subClassOf` and `ChildOf`) is used for this purpose.

Similarly to the RDF triples, the basic expressive primitive in OntoMap are directed binary relations between the concepts that are labeled with other concepts.

The DAML+OIL language (see [8]) can be seen as extension of RDFS. OntoMapO is much similar and basically simpler than DAML+OIL. It is missing class expressions (no enumeration, boolean expressions, and property restrictions), Unique and Ambiguous properties. However, it is not the case that OntoMapO is a sub-language of DAML+OIL, in addition it has number of primitives for mapping (TopInstance, ExactClass, ParentAsInstance, and ChildAsClass) and meronymy (PartOf, MemberOf, and SubstanceOf).

4.2 Concepts, Relations, and Ontologies

Concept is the most basic primitive, so, we are leaving it to the reader's intuition. Just as a reference point the concepts could be compared to the constants in Cyc. The concepts could be related to each other by *binary relations*. Each binary relation has a type that is a concept. Each concept belongs to an ontology and, of course, there could be many different ontologies.

4.3 Instantiation in Addition to Inheritance

Our semantic framework got some inspiration from Cyc, Protégé-2000, and RDFS representation models (see [2], [13], and [16]) — in addition to the inheritance relations we also employ as a basic mechanism the instantiation. So, the concepts are not only described by their parents and children in the subsumption hierarchy but also from the classes that they belong to. The classes themselves are also concepts that could belong to other classes and so on. This way an infinite number of meta-levels could be defined.

We will use a simple set-theoretical semantics to explain the distinction between the inheritance and instantiation. Suppose that each concept is interpreted as a set of its instances. So, `(InstanceOf I C)` means that $\texttt{I} \in \texttt{C}$. In the same fashion `(ChildOf C1 C2)` means that $\texttt{C1} \subset \texttt{C2}$. This interpretation has some pretty reasonable consequences:

- the inheritance relations are transitive — if `(ChildOf C1 C2)` and `(ChildOf C2 C3)` it follows that `(ChildOf C1 C3)`. Really, from $\texttt{C1} \subset \texttt{C2}$ and $\texttt{C2} \subset \texttt{C3}$ it follows that $\texttt{C1} \subset \texttt{C3}$
- the instantiation is not-transitive — if $\texttt{I} \in \texttt{C}$ and $\texttt{C} \in \texttt{MetaC}$ it does NOT follows that $\texttt{I} \in \texttt{MetaC}$
- the instantiation is transitive with respect to inheritance — if `(InstanceOf I C1)` and `(ChildOf C1 C2)` it follows that `(InstanceOf I C2)`. Really, from $\texttt{I} \in \texttt{C1}$ and $\texttt{C1} \subset \texttt{C2}$ it follows that $\texttt{I} \in \texttt{C2}$

An obvious advantage of such extensive use of instantiation is that it makes the hierarchy less tangled avoiding multiple-inheritance on many places. As a design principle, instantiation should be used to express non-sortal properties of the concepts (see [7]). For example, in OntoMapO (see below) we represent the transitivity of a relation type (say, `ChildOf`) via instantiation. It is because the fact that certain relation is transitive does not determines its identity — it is just a rigid property, namely a Category for relations.

4.4 Relations

Each relation between two concepts is an instance of the concept representing its type. Let us extend the set-theoretical interpretation of our model — if `(RelA B C)` than the pair $< \texttt{B}, \texttt{C} > \in \texttt{RelA}$. Suppose there are two concepts `RelA` and `RelB` that represent relation types and the first one inherits the second one `(ChildOf RelA RelB)`. Our interpretation correctly predicts that `RelB` holds between all concepts where `RelA` holds. Let us show how it works:

- let have concepts `A` and `B` and there is a relation of type `RelA` between them `(RelA A B)`

- following our interpretation we can state that $< \texttt{A}, \texttt{B} > \in \texttt{RelA}$
- also `(ChildOf RelA RelB)` means that $\texttt{RelA} \subset \texttt{RelB}$
- now it is obvious that $< \texttt{A}, \texttt{B} > \in \texttt{RelB}$, that means that
- there is a relation of type `RelB` between the concepts `A` and `B`.

In OntoMapO all the concepts representing relation types should be instances of the `BinaryRel` concept or at least one of its children. Further, OntoMap inference engine considers a binary relation to be transitive iff it is an instance of `TransitiveRel` that is a child of `BinaryRel`. Examples for transitive relation are `ChildOf` and `Equivalent`. Analogously, a concept represents a symmetric relation type iff it is an instance of `SymmetricRel` — we can take `Inverse` relation (discussed below) as such example.

4.5 Each OntoMapO Relation Has an Inverse Relation

Another principle that we followed was to define an inverse relation for each of the OntoMapO relations except the symmetric ones, of course. The rationale behind this was twofold:

- to emphasize that the OntoMap relations (in contrast to the slot notion, for example) does not give any representational preference to the concept in the first place
- two make the relations easy to read and follow in both directions

So, `ChildOf` relation has its inverse `ParentOf` relation; `InstanceOf` is inverse to `ClassOf`. In order to keep some correspondence to the frame-based systems we defined `HasSlot` relation as an inverse to the `Domain` relation that could be defined between a relation type and the concept which instances could be first arguments of the relation. Analogously, `Reifies` is inverse to the `Range` relation that holds between a relation type and a concept which instances could be second arguments of the relation. Here are some real constraints that take place in OntoMapO:

- `(Domain Inverse BinaryRel)` and the equivalent statement that `(HasSlot BinaryRel Inverse)`
- `(Range Inverse BinaryRel)`
- `(Domain ChildOf Concept)`, equivalently `(HasSlot Concept ChildOf)`

4.6 How Are the Predefined Relations Special

Let us call *variants* of a relation `R` all its direct or indirect children (i.e. sub-relations or sub-properties) as well as the relations that are equivalent to it or one of its children.

OntoMap considers as an equivalence relation each relation that is variant of `Equivalent`. In a similar fashion, all the variants of `ChildOf` and `ParentOf` are treated as inheritance relations. Analogously, one relation is an instantiation relation iff it is a sub-relation of `InstanceOf` or `ClassOf` relations. Obviously, all the sub-relations of `Inverse` are properly interpreted as inversion by the OntoMap inference engine.

This approach makes the primitives that the OntoMap inference engine understands extensible. For example, when ”explaining” Cyc’s knowledge model to OntoMap it is easy to define `(Equivalent #$genls ChildOf)` – this way OntoMap automatically starts to understand this kind of Cyc inference relations without any need to translate them further.

There is an interesting implementation issue related to this extensibility. In order to infer all the inheritance relations, the engine should know all the equivalence relations (to be able to detect the variants of `ChildOf` and `ParentOf`). However, the opposite is also true. The appropriate algorithms were implemented.

4.7 The Hierarchy

The hierarchy below is basically an inheritance tree augmented with some instantiation information – after each concept name in brackets we have the (most specific) classes it belongs to.

```
Top (Concept)
   Concept (Concept)
      BinaryRel (Concept)
         TransitiveRel (Concept)
         SymmetricRel (Concept)
   Ontology (Concept)
      Context (Concept)
   ChildOf (TransitiveRel)
      MuchMoreSpecific (TransitiveRel)
   ParentOf (TransitiveRel)
      MuchMoreGeneral (TransitiveRel)
   ClassOf (BinaryRel)
      ExactClassOf (BynaryRel)
   InstanceOf (BinaryRel)
      TopInstanceOf (BinaryRel)
   SimilarTo (TransitiveRel, SymmetricRel)
      Equivalent(TransitiveRel,SymmetricRel)
   Inverse (SymmetricRel)
   ChildAsClass (BinaryRel)
   ParentAsInstance (BynaryRel)
   Domain (BinaryRel)
   Range (BinaryRel)
   HasSlot (BinaryRel)
   Reifies (BinaryRel)
   DisjointWith (BinaryRel)
   IsPartOf (TransitiveRel)
   HasPart (TransitiveRel)
   MadeOf (BinaryRel)
   SubstanceOf (BinaryRel)
   MemberOf (BinaryRel)
   GroupOf (BinaryRel)
```

The full definition of OntoMapO in DAML+OIL (version from March, 2001) together with descriptions of each of the concepts is available at
`http://www.ontomap.org/2001/07/ontomapo`

5 Ontology-Mapping Primitives

There is no formal difference between the relations that can be used inside an ontology and those to be used for mapping of concepts in different ontologies. However there are number of relations that are not expected to be used inside well defined ontology — those should be used for handling structural differences between different ontologies. For example, the `(TopInstance A B)` should be used when a concept `A` from one ontology exists as a concept `B` on the upper level of denotation in another ontology, i.e. `(ChildOf X A)` holds iff `(InstanceOf X B)` holds. Such design patterns should not be tolerated inside a single ontology. Here follow explanations for these relations:

- `MuchMoreSpecific`, `MuchMoreGeneral` – the first concept is much more specific (resp. general) than the second one. Both are transitive and inverse to each other and have to be used to ”constrain” the meaning of a concept that has not equivalent or even similar concept in another ontology;
- `TopInstance` – the first concept is the most general instance of the second one. Inverse to `ExactClass`;
- `ExactClass` – the first concept is a kind of meta-concept, the second concept is the most general instance of the first one. Inverse to `TopInstance`;
- `ParentAsInstance` – the first concept is more general than all the instances of the second one that is a meta-concept. Inverse to `ChildAsClass`
- `ChildAsClass` – the first concept is a meta-concept (class), all its instances are more specific than the second concept. Inverse to `ParentAsInstance`

5.1 Representing Ontologies in OntoMap

When an ontology representation has a well defined conceptualization our approach is to map its primitives to the OntoMapO primitives. For example, importing Upper-Cyc Model ([2]) we just defined that

```
(Equivalent #$genls ChildOf)
(Equivalent #$isa InstanceOf)
```

an so forth with the rest of the Cyc’s relations. While OntoMapO interprets each relation that is equivalent or more specific than `ChildOf` as inheritance relation it starts perfectly understand the inheritance in Cyc.

Analogously importing Protégé-2000 ([13]) meta ontology we can establish that:

```
(Equivalent :DIRECT-SUPERCLASSES ChildOf)
(Equivalent :DIRECT-SUBCLASSES ParentOf)
(ChildOf :DIRECT-INSTANCES InstanceOf)
(ChildOf :DIRECT-TYPE ClassOf)
```

First, let us answer why `:DIRECT-TYPE` is more specific than `ClassOf` — in Protégé each concept could be instance just of a single class. This limitation makes `:DIRECT-TYPE` more specific relation than `ClassOf`. Even with this complication, OntoMap will be able to interpret the instantiation as it is defined Protégé because `:DIRECT-TYPE` is a specification (sub-relation) of `ClassOf`, so `:DIRECT-TYPE` is an instantiation relation.

6 Formats and Representations

Publicly available ontologies (or parts of them) will be presented in a number of standard forms:

- PROLOG and KIF;
- DAML-OIL (already available);
- HTML - an online ontology browser as well as static pages available for download;
- SQL scripts for ORACLE and MS SQL Server;
- Ready-to-use files for MS Access (MDB);
- Online application server accessible via CORBA, EJB, RMI, and SOAP (an RPC protocol based on XML.)

  At present, the only the online ontology browser is implemented.

7 The Initial Set of Ontologies

The following ontologies will be hosted initially:

- Upper Cyc Ontology [UCYC]
- EuroWordnet Top Ontology [EWNTOP]
- EuroWordnet Clusters [EWNCLUST] - the clusters of EWN base concepts classified by top concepts. An extension of ETOP
- WordNet 1.5 unique beginners [WNUB5]
- WordNet 1.6 unique beginners [WNUB6]
- WordNet 1.7 unique beginners [WNUB7]
- CORELEX [CLEX] - made on top of WNUB5
- MikroKosmos top-level [MKOSTOP]

- SENSUS top-level [SENSTOP]

For each of the ontologies there will be available an ”executive summary” as well as the most important documents about it (papers, reports, guides and so on), URLs. Of course, the original ”distributives” provided by the creators will be also available. The following ontologies are already hosted on OntoMap: UCYC, EWNTOP, WNUB7, and MKOSTOP. Clear candidates for hosting are (or will be) also OpenCyc, any results of the SUO effort, and the Simple Core Ontology.

Mappings between some of the ontologies will be provided in order to ensure an easier understanding and comparison between them. So, the ontologies hosted will form an interconnected graph. Such mapping already exists between EWNTOP and UCYC (see [9]). The mapping between WNUB7 and MKOSTOP and the later two ontologies was recently developed. Some of the relations in the graph exist because of the nature of the ontologies:

- EWNCLUST and ETOP - the concepts of the former one are just conjunctions of those of the later one
- CLEX and WNUB5 - same as above
- WNUB5 and WNUB6 - there exist a mapping provided by the creators
- SENSTOP and UCYC - it is available as a part of the UCYC distributives as well as separately by the SENSUS developers.

The following mappings will be developed as a part of the project:

- EWNCLUST to UCYC
- CLEX to EWNCLUST (both directions)
- WNUB7 to UCYC
- WNUB7 to EWNTOP

The mappings will be available in the same formats as the ontologies. Actually, each mapping could be seen as an extension of the target ontology. For example, the mapping between EWNTOP and UCYC can be considered as an extension of the UCYC with the concepts of EWNTOP that are connected appropriately to the UCYC constants (see [9]). No ontologies and upper-level models will be developed under the project instead of the OntoMapO meta-ontology mentioned above.

8 One Usability Example

Here follows an example of how the OntoMap could help understanding a complex case in the Upper Cyc Ontology - the `#$MeetingTakingPlace` constant. The comprehension comes from the two sources: it is deeply positioned in the tangled subsumption hierarchy; and also some important information is encoded via instantiation. The most readable representation in [2] is:

The collection of human meeting events, in which #$Persons gather intentionally at a location in order to communicate or share some experience; business is often transacted at such a meeting. Examples include: a particular conference, a business lunch, etc.

isa: #$DefaultDisjointScriptType, #$ScriptType, #$TemporalObjectType
genls: #$SocialGathering
some subsets: (16 unpublished subsets)

The underlined text represents hyper-references to descriptions of the appropriate constants. Below follows the standard view on the same concept provided by OntoMap:

Concept: **#$MeetingTakingPlace** [UpperCyc]

Gloss: The collection of human meeting events, in which #$Persons gather intentionally at a location in order to communicate or share some experience; business is often transacted at such a meeting. Examples include: a particular conference, a business lunch, etc.

Super-concepts (parents): #$SocialGathering;
indirect: #$IntangibleIndividual, #$CompositePhysicalAndMentalEvent, #$TemporalThing, #$PhysicalEvent, #$MentalEvent, #$Intangible, #$Thing, #$SpatialThing,#$MentalActivity, #$PurposefulAction, #$Situation, #$HumanActivity, #$AnimalActivity, #$Event, #$Action, #$Individual, #$SocialOccurrence

Indirect parents in other ontologies:
Physical[EWN_Top], Top[EWN_Top], Mental[EWN_Top], Social[EWN_Top], 2ndOrderEntity[EWN_Top]

Instance of: #$DefaultDisjointScriptType #$TemporalObjectType
indirect: #$Collection, #$ObjectType, #$Thing, #$SituationType, #$SetOrCollection, #$Intangible, #$MathematicalOrComputationalThing, #$ScriptType

Sub-concepts (children): <none>

Direct instances: <none>

All direct relations:

```
#$genls: #$SocialGathering
#$isa: #$TemporalObjectType
#$isa: #$DefaultDisjointScriptType
```

This was a ”snap-shot” of the current on-line interface of OntoMap. Pay attention that both indirect parents and classes are displayed that is extremely useful — it requires serious efforts to reconstruct this indirect relations manually. Also, super-concepts in the EuroWordnet (EWN) Top Ontology can be seen, that provides a good impression about a possible position of `#$MeetingTakingPlace` there. These way people that are familiar with EWN top can get an idea about the meaning of the Cyc constant.

9 Ontology Unification Services

In parallel with the maintenance of the server and updates to the content we will be able to provide the following services for both domain specific and upper-level ontologies:

- Loading the ontology in OntoMap and hosting it there. This way it will become accessible in all the formats supported. Also its conformance profile could be determined (see [3] and Unified Representation section). A security subsystem will be developed, so the proprietary ontologies will not be publicly available.
- Developing of mappings to ontologies that are already hosted in OntoMap. The mappings themselves will be also available in all the supported formats.

10 Conclusion

OntoMap project is still in an early phase that makes it hard to evaluate it. The experience gathered providing the Upper Cyc Ontology as MS Access database is encouraging – even though the original resource is available for a long time more than two hundred people found it useful and downloaded it in this shape just for one year. We got a very positive feedback for another experiment of ours – developing a mapping between the Upper Cyc Ontology and the EuroWordnet Top Ontology and then providing it as a database as well as an online service. Even without significant theoretical innovation such facilitatory efforts seem to be important for the development of the semantic modeling community.

The first interesting result of the project is a Java-written inference engine that already supports the OntoMapO language – it is sound and complete with its support for inheritance, instantiation, inverse, transitive, and symmetric relations. The biggest ontology that we experimented with (Upper Cyc Ontology, about 3000 concepts) can be loaded in few seconds and than queried in real-time. The inference engine is used for support of an on-line ontology browser that is publicly available at `http:\\www.ontomap.org`. Apart from the engineering work needed to make available the remaining functionality of OntoMap portal we are constantly working on evaluation and development of the semantic framework – OntoMapO.

References

[1] Campbell, A.E. and Shapiro, S.C., *Algorithms for Ontological Mediation*, Technical Report 98-03, Dep. of CS and Engineering, State Univ. of New York at Buffalo, 1998.

[2] Cyc Ontology Guide: Introduction.
http://www.cyc.com/cyc-2-1/intro-public.html

[3] Genesereth, Michael R., and Fikes, Richard eds. *Knowledge Interchange Format draft proposed American National Standard (dpANS)*. NCITS.T2/98-004 http://logic.stanford.edu/kif/

[4] Gomez-Perez, A.; Fernandez, M.; Blazquez, M.; Garcia-Pinar, J. M. *Building Ontologies at the Knowledge Level using the Ontology Design Environment.* http://delicias.dia.fi.upm.es/articulos/ode/ode.html

[5] Gruber, Thomas R. *Ontolingua: A Mechanism to Support Portable Ontologies.* Technical Report KSL 92-66, Knowledge System Laboratory, Stanford University, 1991.

[6] Gruber, Thomas R. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing.* Technical Report KSL 93-04, Knowledge System Laboratory, Stanford University, 1993.

[7] Guarino, Nicola; Welty, Christopher *A Formal Ontology of Properties.* In the Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France. R. Dieg and O. Corby (Eds.): EKAW 2000, LNAI 1937, pp. 97-112, Springer Verlag, 2000.

[8] Ian Horrocks, Frank van Harmelen, Peter Patel-Schneider (eds.) *DAML+OIL (March 2001)* http://www.daml.org/2001/03/daml+oil-index.html

[9] Kiryakov, Atanas; Simov, Kiril Iv. *Mapping of EuroWordnet Top Ontology to Upper Cyc Ontology.* In: Proceedings of "Ontologies and Text" workshop, during EKAW 2000. Juan-les-Pins, French Riviera, Oct. 2, 2000. http://www.ontotext.com/publications/ index.html# KiryakovSimov2000b

[10] Knight, K. and Luk, S. *Building a Large Knowledge Base for Machine Translation.* Proceedings of the American Association of Artificial Intelligence Conference AAAI-94. Seattle, WA, 1994.

[11] McGuinness, Deborah L., Richard Fikes, James Rice, and Steve Wilder, *An Environment for Merging and Testing Large Ontologies*, Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000). Breckenridge, Colorado, USA. April 12-15, 2000.

[12] Maedche, A.; Schnurr, H.-P.; Staab, S.; and Studer, R. *Representation Language-Neutral Modeling of Ontologies.* In: Frank (ed.), Proceedings of the German Workshop "Modellierung" 2000. Koblenz, Germany, April, 5-7, 2000.

[13] Noy, Natalya F.; Fergerson, Ray W.; Musen, Mark A. *The Knowledge Model of Protege-2000: Combining Interoperability and Flexibility.* In the Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France. R. Dieg and O. Corby (Eds.): EKAW 2000, LNAI 1937, pp. 97-112, Springer Verlag, 2000.

[14] Ortiz, Antonio Moreno. *Managing conceptual and terminological information in a user-friendly environment.* In: Proceedings of "OntoLex 2000: Ontologies and Lexical Knowledge Bases", Sozopol, Sept. 8-10, 2000. (to appear)

[15] Vossen, Piek (ed.) *EuroWordNet General Document Version 3, Final*, July 19, 1999. http://www.hum.uva.nl/ ewn/

[16] World Wide Web Consortium; Brickley, Dan; Guha, R.V. (eds.) *Resource Description Framework (RDF) Schema Specification*, 1999. http://www.w3.org/TR/1998/WD-rdf-schema/

[17] Uschold, Mike; King, Martin; Moralee, Stuart; and Zorgios, Yannis *The Enterprise Ontology*, The Knowledge Engineering Review, 1998, Vol. 13, Special Issue on Putting Ontologies to Use (eds. Mike Uschold and Austin Tate).

[18] Uschold, Mike *Converting an Informal Ontology into Ontolingua: Some Experiences*, Univ. Edinburgh, Artificial Intelligence Application Institute (AIAI), AIAI-TR-192, March 1996.

The "Emergent" Semantic Web: A Consensus Approach for Deriving Semantic Knowledge on the Web[1]

Clifford Behrens and Vipul Kashyap
Telcordia Technologies, Inc
445 South Street
Morristown, NJ 07960-6438, USA
{cliff, kashyap}@research.telcordia.com

Abstract. The recent and growing interest in the Semantic Web has given rise to a flurry of activity in standardization bodies (such as the W3C) to specify semantics using formal languages and inference mechanisms. The real challenge, however, is to link formal semantics with deeper meaning as reflected by consensus discovered among users on the Semantic Web. We believe the process of deriving and formally describing ontologies for the Web (expressed using standardized languages) is necessarily a social-cultural one; hence, new consensus-based tools are required to derive shared semantic systems for different communities of interest. This paper introduces Consensus Analysis as a means for deriving semantic knowledge from the information provided by subject matter experts and describes the *Schemer System* prototype for acquiring and processing this information. The results of a trial application of this approach and prototype on technologists asked to identify current mass market consumer trends in the domain of Internet privacy and security are reported. These findings implicate Consensus Analysis as a powerful tool capable of enabling the semantic Web by yielding core knowledge such as controlled vocabularies and domain ontologies.

1. Introduction

Recently there has been a great interest in the Semantic Web and issues related to specification and exploitation of semantics on the WWW. In particular, shared ontologies are being proposed for representing the core knowledge that forms the foundation for semantic information on the Web. Fensel [1] has identified two broad research thrusts related to ontologies:

1) Approaches to standardize the formal semantics of information to enable machine processing. Work being done as a part of the W3C RDF working group [2] and the DAML+OIL initiative [3] falls within this category.
2) Approaches to define real world semantics linking machine processable content with meaning for humans based on consensual terminologies.

To realize the goals of the Semantic Web, there is a need to wed approaches centered on the formal representation of semantics with approaches to systematically acquire terminologies that best express shared systems of meaning among users. We believe the process of deriving and describing domain ontologies necessarily involves the search for consensus among domain experts and, therefore, is inherently a social-cultural one. As such, the proper approach to deriving knowledge, like domain ontologies, ought to be sensitive to the semantic context of information, and should be informed by the real-world bottom-up, decentralized process in which knowledge typically evolves. After all, decentralized models for consensus achievement better reflect the dynamic sociological characteristics of the Web (which have been the cause for its rapid acceptance and success). In this manner more meaningful ontologies (expressed in standardized formal languages) can emerge through more natural interactions of Web users within their respective communities. We agree with Fensel [1] who claims that the real challenge for making the semantic Web a reality is, "a model for driving the network that maintains the process of evolving ontologies."

Consistent with this claim, similar interest in Knowledge Management processes has motivated new research in automatic knowledge acquisition, classification, and representation [4]. Much of the discussion on Knowledge Management has focused on information technology, e.g., hardware, software and communications networks, but has not laid out in clear terms what notions of "knowledge" need to be supported by this technology. For example, there exists in the literature a recurring theme that knowledge is any information stored in a Knowledge Repository, and that this knowledge can somehow be acquired or "discovered" automatically from disparate, heterogeneous information sources, e.g., Web pages and networked document collections. This approach seems at best naïve as it ignores the context and intended purpose of source information. Without establishing this context and purpose, it seems unlikely that much useful "knowledge" can be discovered as it leaves matters pertaining to information's meaning and relationships with existing knowledge open to broad interpretation.

Within the literature there is expressed the idea that not all information is knowledge; information only becomes knowledge once it is mapped to a knowledge structure, i.e., it is organized in a way that makes it accessible and comprehensible to users [4]. In fact, this qualification suggests that there may exist many such structures for organizing the same information, again supporting the idea that context and purpose are essential for transforming information to knowledge. It also implicates the importance of knowing the "community of interest" (COI) for both the producer and consumer of information to enable this transformation since members of different COIs may set different contexts or use the information with very different intentions. In the emergent Semantic Web, it is critical to determine the “consensus” knowledge structures for a COI.

The term "community" is becoming ubiquitous, particularly in discussions related to delivery of personalized services on the Internet, yet there exist distinct usages of the term. For some, a community seems to consist of all who, because of a shared interest in certain kinds of information, frequent the same place, real or virtual, regardless of any interaction among them, e.g., all those who browse the same Web site [5]. A more social usage entails information exchanges among a collection of individuals, e.g., all those who exchange useful information about some topic of mutual interest through email or chat rooms [6]. A more sophisticated "cultural" notion of community refers to all who, in addition to meeting the two preceding conditions, share a vocabulary, *semantics* and theory for organizing information.

For members of this class of community there exists some common purpose and key concepts for communicating ideas and sharing experiences. In this paper, this last notion of community is adopted because, as it will be demonstrated, it provides an opportunity to analyze knowledge, and its variations among individuals, with greater formal rigor. It also helps to more clearly draw the lines operationally between information, individual knowledge, and what will be referred to later as "cultural knowledge."

Much research has already been conducted in the social sciences, particularly cognitive anthropology and cognitive psychology, on modeling knowledge domains, i.e., conceptual categories that include other semantically-related categories, and eliciting the information needed to build these models. However, most of these methods are extremely time-consuming, taxing the attention of a few SMEs (Subject Matter Experts), those recognized as experienced and possessing specialized domain knowledge. *Consensus building* is another approach to building knowledge representations that is gaining increasing popularity in the Information Processing standards community and elsewhere [7, 8]. New information technology could be applied to eliminate much of the need (and enormous cost) of face-to-face group decision-making meetings, e.g., read [9] and [10] for examples of IT approaches to collaborative knowledge construction.

Previous methods for building knowledge from consensus have been tried, e.g., Delphi approaches [11, 12], but these are typically iterative and require much human intervention. While the importance of consensus to achieving views that best represent collective thinking is often stressed, too often views are biased strongly by the force of individual personalities and are not representative of any particular COI. Other problems arise from the heterogeneous composition of decision-making groups whose members conceptualize the same problem from widely different perspectives, i.e., those of different COIs. Moreover, simple polling methods that only average expert opinion do not usually yield results with the depth and logical properties of real domain knowledge, nor do they exploit the contributions of the most competent SMEs. Thus, there is need for a different approach that derives, rather than forces, consensus views, does so without the need for much human intervention and many iterations, acquires useful information from SMEs (weighted by their competence) at their convenience, and is capable of yielding shared knowledge for a demonstrable domain of interest.

By combining new formal and more rigorous approaches to consensus-modeling, specifically powerful methods of Consensus Analysis that already have been tested successfully by Cognitive Anthropologists in numerous knowledge domains, the network services approach taken in this research overcomes the limitations of previous computer-assisted approaches. This is accomplished by (1) incrementally refining or "evolving" knowledge, (2) providing *metrics* for evaluating the cultural saliency of a domain and the knowledge-based competency of SMEs in a COI, (3) dynamically assigning SMEs to the most “appropriate” COI and (4) not only spreading the task of knowledge acquisition among many SMEs, rather than just a few, but also leveraging Web infrastructure to engage them at their convenience.

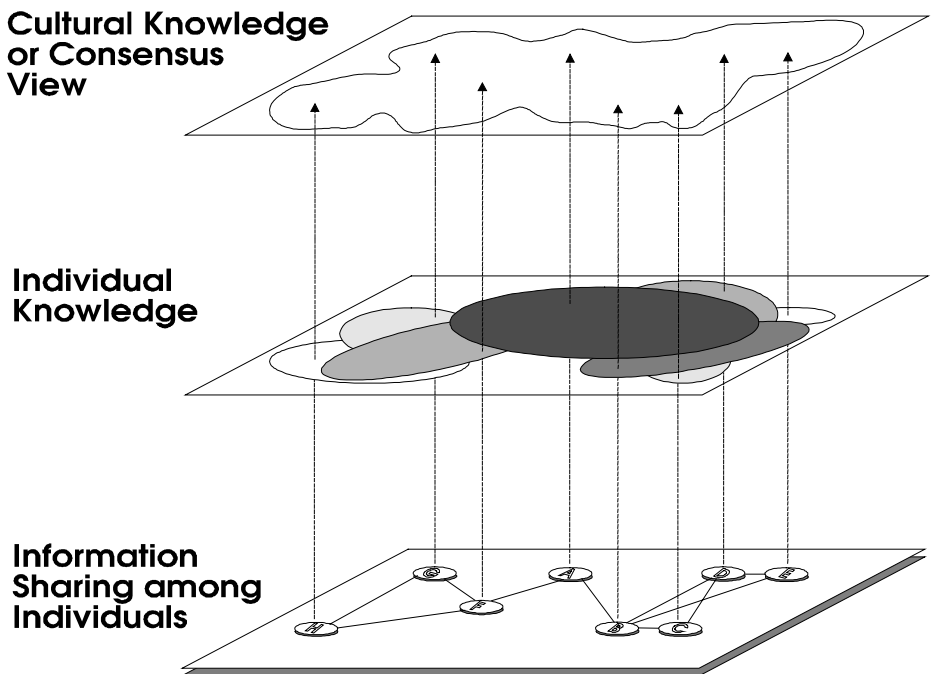


Figure 1. Knowledge distribution, knowledge sharing and consensus.

2. Cultural Knowledge and Consensus Analysis

Consensus Analysis is based on a few simple, but powerful, ideas , i.e., *knowledge is both distributed and shared* [13]. For any knowledge domain, and any group of subjects "expert" in this domain, so-called "SMEs" possess different experiences; hence, they know different things, and some of them know more than others (see Fig. 1). Information sharing, e.g., among individuals A-H in the figure, facilitates the availability of a much larger pool of information with non-uniform distribution of knowledge across SMEs. For example, many information standards groups are composed of data providers, data users, librarians and software vendors. These groups tend to possess different experiences with data, and apply their own unique views and semantics to describe these data. Yet, certain individuals (the hi-tech "gurus") are recognized as being more knowledgeable than others, i.e., there exist recognized domain "experts." Because of their widely regarded and highly-valued knowledge, these experts are frequently requested to share what they know with others as consultants or as leaders in standards-setting groups, or render opinions about how best to describe or classify information in their domain of expertise. Hence, one typically finds within any COI that there is differential expertise among its members, but also some knowledge that is widely-shared and recognized as being "essential." In fact, this knowledge may be so fundamental and its use so widespread that, over time, it becomes logically well-structured or canonical, e.g., even published as a metadata content standard. *The process of mapping information onto such a consensus standard is the essence of cultural knowledge creation.*

Cultural knowledge is not all that one knows (e.g., the set of knowledge for each individual represented in the middle layer of Figure 1); nor is it the sum total of what

everybody knows (e.g., the union of individual knowledge sets in the middle layer). Rather, it is an abstraction, knowledge shared in its "broad design and deeper principles" by members of a society or community [14]. In other words, while its entire details are not usually known (or can be always be articulated explicitly) by anyone, cultural knowledge consists of those things that all members of a COI understand all others hold to be true.

Kroeber [15] referred to this highly-structured, rich form of knowledge as a "systemic culture pattern," a coherent subsystem of knowledge that tends to persist as a unit. This unit features a semantic system, consisting of an appropriate vocabulary and grammar, for classifying and talking about elements within a knowledge domain. Examples of cultural knowledge are: a kinship terminology [16, 17], or perhaps a metadata content standard [18], a consensus statement for screening cancer [19], or a set of software requirements [20]. It is this shared pool of structured information, acquired primarily by learning, which constitutes cultural knowledge [21].

2.1 Robustness of the Consensus Model

The significance of information sharing and distribution of cultural knowledge has encouraged some researchers to exploit consensus, measured by intersubject agreement, as an indicator of knowledge. The method of Consensus Analysis was first presented in several seminal papers [13, 22, 23]. In addition to introducing the formal foundation for Consensus Analysis (reviewed later in more detail), the initial papers cited above also provided examples of its application to modeling knowledge of general information among US college students, and the classification of illness concepts among urban Guatemalans. Other more recent applications of Consensus Analysis have focussed on measuring cultural diversity within organizations [24]. These successes, obtained for a wide variety of domains and social-cultural contexts, indicate that the following three explicit assumptions, upon which Consensus Analysis is based, are extremely robust [13]:

i) *Common Truth*. There is core knowledge (expressed in a highly probable set of answers to questions or "items") that is "applicable" to all SMEs or, put another way, all SMEs are members of the same COI and generally share a common perspective or "cultural reality."

ii) *Local Independence*. The information or responses provided by each SME are acquired independently from those of other SMEs, i.e., SME item response random variables satisfy conditional independence for all possible response profiles and the core answer set.

iii) *Homogeneity of Items*. Each SME has a fixed level of "competence" or "expertise" across all items, i.e., items used to sample what SME's know are equally difficult and provide representative coverage of a coherent domain. In practice, this assumption has been found to be quite robust and requires only that those SMEs who are most knowledgeable in a domain consistently outperform non-experts.

From these assumptions, it is possible to derive a method for estimating three properties of interest: (1) a measure of the overall saliency of the knowledge domain represented by the pool of items, (2) the level of domain expertise or "cultural competence" for each SME based on the amount of consensus or agreement between his/her responses to items with those of all other SMEs, and (3) the most probable set of "correct answers," inferred from the responses

of each SME and weighted by their respective competence measures, i.e., the consensus view.

2.2 Statistical Methodology

As mentioned earlier, the Consensus Analysis Model can be derived formally from the three assumptions given in section 2.1. This formal model consists of a data matrix X containing the responses X_{ik} of SMEs *1..i..N* on items *1..k..M*. From this matrix another matrix M^* is estimated and it holds the empirical point estimates M^*_{ij}, the proportion of matching responses on all items between SMEs *i* and *j*, *corrected for guessing* (if appropriate), on off-diagonal elements (with $M^*_{ij} = M^*_{ji}$ for all pairs of SMEs *i* and *j*). Alternatively, another matrix C^*, which contains the observed covariances C^*_{ij} between the responses of SMEs *i* and *j*, corrected for variance among SME answers, may be substituted for M^* [25]. To obtain D^*_i , an estimate of the proportion of answers SME *i* "actually" knows and the main diagonal entries of M^* (or C^*), a solution to the following system of equations is sought:

$$M^* = D^* D^{*\prime} \text{ or alternatively,} \quad (1)$$

$$C^* = D^* D^{*\prime} \quad (2)$$

where D^* is a column vector containing estimates of individual competencies $D_1..D_i..D_N$ and $D^{*\prime}$ is merely its transpose. Since equation 1 (or 2) represents an overspecified set of equations and because of sampling variability, an exact solution is unlikely. However, an approximate solution yielding estimates of the individual SME competencies (the D^*_i) can be obtained by applying *Minimum Residual Factor Analysis* [26], a least squares approach, to fit equation 1 (or 2) and solve for the main diagonal values. The relative magnitude of eigenvalues (the first eigenvalue λ_1 at least three times greater than the second) is used to determine whether a single factor solution was extracted. All values of the first eigenvector, v_1, should also range between 0 and 1. These results test the validity of the Common Truth assumption.

If the criteria above are satisfied, then the individual SME competencies can be estimated with:

$$D^*_i = v_{1i} \sqrt{\{\lambda_1\}} \quad (3)$$

The D^*_i, then, are the loadings for all SMEs on the first factor. These estimates are required to complete the analysis, i.e., to infer the "best" answers to the items. The estimated competency values (D^*_i) and the profile of responses for item k ($X_{ik,l}$) are used to compute the Bayesian *a posteriori* probabilities for each possible answer. The formula for the probability that an answer is "correct" follows:

$$Pr(\langle X_{ik} \rangle\ i=1 \mid Z_k=l) = \prod_{i=1}^{N} [D^*_i + (1-D^*_i)/L]^{Xik,l}\ [(1-D^*_i)(L-1)/L]^{1-Xik,l} \quad (4)$$

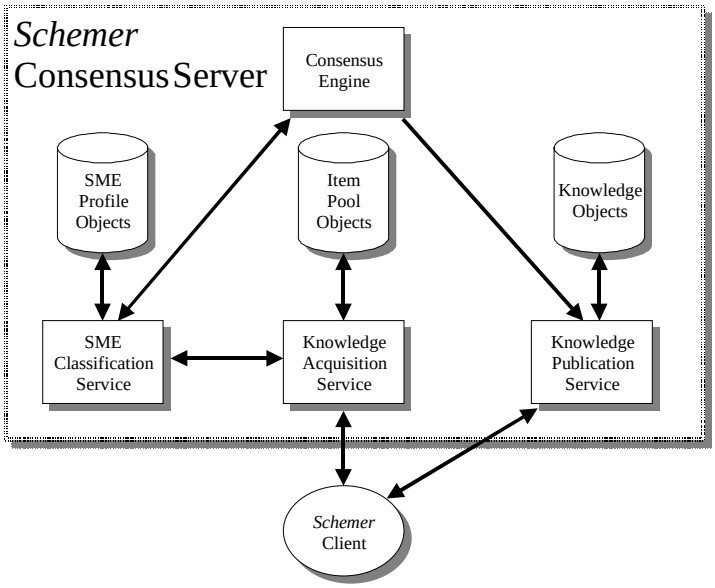


Figure 2. *Schemer* system architecture.

where Z_k is the "correct" answer to item k, l is the l^{th} response to item k, and L is the total number of possible responses ($l_1 ... l_L$) to item k. Again, it should be mentioned that the "correctness" of an answer is relative to the perspective shared by members of a particular COI, i.e., the one sampled. Equations 1-4 provide formal motivation for the approach taken in this research, and indicate algorithms that need to be implemented in software as part of a network-enabled consensus server.

3. System Architecture and Prototype

Telcordia researchers have begun to design a software prototype called the *Schemer System*, shown in Fig. 2. Key software components in this design have already been implemented to communicate better some of the objectives of the approach, stimulate greater interest in it, and demonstrate the feasibility of automating Knowledge Acquisition and Consensus Analysis modeling. Future work will include development of Publication Services and a fuller integration of software components in a continuous Web-based service.

In our current design, the *Schemer System* consists of a *Schemer Client* and *Schemer Server*; however the latter really involves the interaction of four services: a *Subject Matter Expert Classification Service*, a *Knowledge Acquisition Service*, a *Consensus Engine*, and a *Knowledge Publication Service*. These services read and write information to several data bases, one storing information about SMEs, another storing pools of items used to acquire information from SMEs, and another which stores the derived knowledge structures, i.e., the controlled vocabularies, forecasts, ontologies, classification schemes, or productions systems. Next, each of these services is examined in more detail.

Client. The job of the *Schemer Client* is to provide a graphical/text interface through which a user communicates with the *Schemer Server*. It presents information sent by the Server such as item forms and knowledge visualizations, both textual and graphic.

SME Classification Service. This service determines a knowledge domain of interest for a SME, and assigns a SME to his/her proper COI. Classification is necessary to present a SME with meaningful items and knowledge derived from a consensus analysis of peer responses. Knowledge domain identification may be determined either by asking a SME to select a known knowledge domain from a list or, if unknown to *Schemer*, the SME is asked to input the name of this knowledge domain. COI classification may be accomplished in two stages, *a priori* and *post hoc*. If nothing is known about the SME, then preliminary COI classification is made by asking the SME to choose a COI from a list of COIs already known to *Schemer*. If, however, the SME cannot find an appropriate COI in this list (or if the knowledge domain is unknown to *Schemer*), then he/she is prompted for a list of key terms frequently used to describe objects in the knowledge domain of interest. These terms are matched against key term lists (if they exist) for known COIs to determine the "best" COI classification for this SME. But once an item form has been constructed for this SME and used to acquire more information, *post hoc* analysis of results obtained from Consensus Analysis may be used to reclassify this SME, if he/she so chooses. To compare new information obtained from the SME with information known for SMEs already classified, the *Subject Matter Expert Classification Service* reads the data it needs from a repository of SME profile objects.

Knowledge Acquisition Service. Based on the knowledge domain and preliminary COI classification obtained for a SME, this service selects an appropriate item pool object and composes an instrument or form that is used to determine what the SME knows about the domain. Items published on these forms are read from a repository of item pool objects, each identified by knowledge domain and COI. This service sends the form to the Client where the SME enters his/her responses to items on the form, then sends these back to the *Knowledge Acquisition Service*. The SME's response pattern, along with his/her ID, is stored in a repository of SME profile objects, also grouped by COI and knowledge domain.

Consensus Engine. This service performs a Consensus Analysis of data collected for a knowledge domain/COI grouping each time new responses are added to a SME's profile, and stores the updated result in a *Knowledge Repository*, along with ancillary statistics, e.g., Goodness-of-fit indices. Not only does the *Consensus Engine* analyze data read from SME profiles, but it also adds information to these, e.g., a SME's competency score.

Knowledge Publication Service. On a user's request, this service constructs forms with textual and graphical representations of derived knowledge, stored in the *Knowledge Repository*, for presentation on the *Schemer Client*. Access to information stored in this repository is also provided by this service so that a user can retrieve a knowledge object for use with his/her own software application.

To date, a skeleton *Knowledge Acquisition Service* has been built, capable of taking as input from a SME's Web browser a knowledge domain and COI value, then return a form with an appropriate item set for this knowledge domain/COI combination. Currently, only dichotomous (True/False) formats are supported. Once a SME completes this form and submits his/her responses to the *Knowledge Acquisition Service*, it notifies the *Consensus Engine* that a SME's profile has been updated. The *Consensus Engine* processes all of the response vectors for SMEs in the same knowledge domain/COI data base, then stores the

results, e.g., eigenvalues, SME competency scores and the estimated answer key, in the knowledge base. All of these services have been implemented in Java® and the R® statistical programming environment, so can run under Unix® or Windows®.

4. Experiment

The remainder of this paper describes an experiment that was conducted among Telcordia technologists to derive a consensus view of mass-market consumer trends related to Internet security and privacy. While no attempt will be made to derive a domain ontology from this experiment, our intent is to demonstrate how Consensus Analysis works and to further suggest that it seems well-suited for this purpose.

A prototype of the *Schemer* system was built and used to deliver a questionnaire consisting of sixty-seven items related to privacy and security of information on the Internet (see Appendix A). These items were derived from Georgia Tech's *10th World Wide Web User Survey*, which includes a section entitled, "Online Privacy and Security". A request was mailed electronically to Research Scientists belonging to two labs within Telcordia Technologies Applied Research. These sample SMEs were asked to answer items on the questionnaire *as if they were domain experts* being asked for their opinions about *mass market consumer trends within the Web user community*, not necessarily with their personal opinion. Along with responses to the questionnaire, SMEs were asked for their employee ID and a list of no more than twenty descriptors that they believed best represented their professional area of expertise. The former was used as a pointer to other ancillary information about the SME, e.g., lab, department, group, and office location, while the latter was collected to help associate the domain expertise of a SME with that of others in the sample. A total of thirty-six Research Scientists responded to the request above. This sample was opportunistic, not random; moreover, a special request was made to members of Telcordia's *Computer Networking Research* department, which specializes in Internet security issues, so that the responses of these domain experts could be compared to others in the sample.

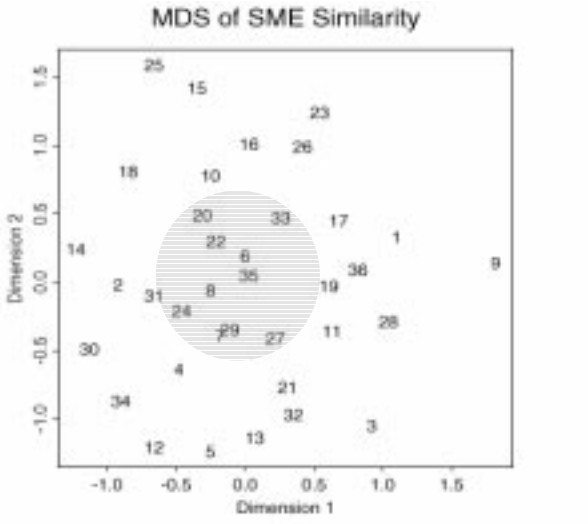


Figure 3. Plot of MDS results showing similarities in responses among SMEs. Similar SMEs are plotted close to one another. Stress= 0.260 after 19 iterations.

The similarity or agreement among SME response patterns can be explored in Figure 3. This two-dimensional plot was obtained from a Multidimensional Scaling of only the off-diagonal entries of the consensus matrix (M_{ij}* in Equation 1) calculated for the thirty-six Telcordia SMEs [27]. In this visualization, the SMEs with similar responses are plotted closest to one another. The absence of clustering in this plot suggests that the study sample was drawn from a single COI whose members share core domain knowledge about "Online Privacy and Security." This notion was tested more rigorously by estimating a consensus model for these data.

4.1 Knowledge Domain Validation

As the review of Consensus Analysis pointed out, knowledge derivation rests on establishing the validity of the domain to those SMEs in the sample. This is accomplished by inspecting the relative magnitudes of the eigenvalues for the first factors extracted from the consensus matrix using Minimal Residuals Factor Analysis. Again, the "rule-of-thumb" is that the eigenvalue of the first factor must be at least three times greater than the second; moreover, subsequent eigenvalues should all be small and roughly equivalent. Inspection of the eigenvalues for the first three factors extracted from the response set collected from Telcordia SMEs reveals that the first is over six times greater than the second, and the second and third eigenvalues are almost equal (see Table 1). This lends strong support to the claim that the items on the questionnaire are sampling a single, coherent knowledge domain, and that this domain has salience for the sample of respondents. Moreover, the high Pseudo-Reliability Coefficient (0.944) also obtained suggests that these results are stable and would likely be the same ones obtained with repeated sampling [13].

Table 1. Eigenvalues for testing saliency of "Online Privacy and Security" knowledge domain.

Factor	Eigenvalue	Percent	Cumulative %	Ratio
1	11.902	77.8	77.8	6.500
2	1.831	12.0	89.8	1.175
3	1.559	10.2	100.0	
	15.292	100.0		

4.2 Estimation of SME Competence

Having established the saliency of "Online Privacy and Security" as a knowledge domain for SMEs in the sample, it is possible to estimate each one's competency in this domain. The competencies for this sample of SMEs are listed in Table 2. This metric can be interpreted as the probability that a SME would correctly answer an item. Competencies for this sample range from 0.32-0.76 with a mean of 0.56± 0.11. With a sample size of thirty-six, and average competency level of 0.56, it ought to be possible to correctly classify (as either

“true” or “false”) at least 95% of the items on the "Online Privacy and Security" questionnaire with a 0.999 confidence level [13].

Table 2. Estimates of competency for thirty-six SME’s questioned about "Online Privacy and Security."

SME	Competency	Organization	Location
1	0.48	C2E	M3B
2	0.60	C2E	M3B
3	0.41	C2E	M3B
4	0.56	C2E	M3R
5	0.48	C8E	M3B
6	0.75	ICI	N3X
7	0.75	Missing	Missing
8	0.67	I9B	M2R
9	0.32	C7H	M3B
10	0.58	C8I	N1X
11	0.61	C1B	M3B
12	0.47	C2I	M3B
13	0.52	I0B	M2R
14	0.50	C2I	M3B
15	0.42	C2F	M3R
16	0.52	C8E	M3R
17	0.59	I5I	M2B
18	0.45	C7E	M3B
19	0.59	C8E	M3B
20	0.67	C2I	M3R
21	0.55	C2A	M3B
22	0.67	I5I	M2R
23	0.46	C8F	M3B
24	0.76	C8B	N3Z
25	0.35	I5B	M3R
26	0.51	C8F	M3B
27	0.67	C8F	M3B
28	0.52	I5H	M2R
29	0.72	C7H	M3B
30	0.52	A4B	M2R
31	0.63	Missing	Missing
32	0.58	I9D	M2B
33	0.64	C2A	M3R
34	0.51	I9I	M2B
35	0.69	C2A	M3B
36	0.59	C2A	M3B

4.3 Knowledge Derivation

By using SME competencies as weights, the most probable set of answers can be estimated from SME responses with Bayes' formulation in Equation 4. In Table 3 the answers obtained in this way are compared to the dominant responses given by the 1,482 respondents who completed the GVU survey. Pearson's Chi-square (with Yate's continuity correction) was calculated to test for independence between the two sets of answers [28]. A Chi-square value of 11.852, with one degree of freedom, was obtained from the test, and with a p-value<0.001, there is strong support to conclude that the answers estimated through Consensus Analysis are not different from those obtained for the GVU sample. A Yule's Q= 0.78 also indicates that this association is a reasonably strong one.

Table 3. Cross tabulation comparing majority responses on GVU survey to those estimated from responses of Telcordia SMEs with Consensus Analysis.

	Telcordia SMEs		**Marginal Totals**
GVU Survey	False	True	
False	20	14	**34**
True	5	28	**33**
Marginal Totals	**25**	**42**	**67**

4.4 SME Classification

With estimates of SME competencies in hand, the spatial arrangement of points plotted in Figure 3 can be given a particularly nice intuitive interpretation. Those SMEs who knew the most about "Online Privacy and Security" are plotted in the center of this figure; in fact, those ten SMEs with the highest competency scores fall within the shaded area; while those with the lowest scores are located at the periphery of this plot. However, there also exists idiosyncratic variation among these SMEs in what they know about this domain, and so domain expertise seems to cross organizational boundaries. This idea was tested in several ways.

The terms that SMEs provided to describe their technical areas of expertise were carefully enumerated. Surprisingly, while the frequent use of "hot buzz words" was anticipated, it seems that SMEs exploited free-listing as an opportunity to create very specialized identities. In fact this sample of SMEs applied 189 unique descriptors (each consisting of one or more terms) to characterize their expertise. The number of descriptors listed by SMEs ranged from 0-16 with an average list size of 5.25 descriptors. Four SMEs listed no terms. Only nineteen of the 189 descriptors were listed by more than one SME and all but two of these nineteen were listed only twice, further suggesting a reason for the absence of any discernable clustering of points in Figure 3. However, the five most competent SMEs (24, 6, 7, 29, and 35) identified themselves as knowing more about business and marketing aspects of telecommunications, e.g., "Business planning", "economics", "market-oriented programming"; and used terms such as "system administration" and operations "hand-offs",

implying greater familiarity with consumer or user-oriented perspectives. The only shared concepts expressed in the free lists of those SMEs (9, 25, 3, 15, and 18) with the lowest competency scores were “distributed computing”, “Internet”, “Internet Protocols”, and to some degree more abstract interests, e.g., “mathematics”, “formal methods”, and “theory of distributed systems”. It seems that this group is focused more on privacy and security from a network engineering or design perspective, rather than from a consumer's point-of-view.

More rigorous statistical tests of the organizational and locational basis for knowledge distribution among these SMEs were also made. For these tests, two other symmetrical distance matrices were constructed: the first from SME organization numbers and the second from their office locations (both listed in Table 2). The Organization Code consists of three characters that identify a SME's lab, department and group, respectively. A matrix, whose cells express the organizational distance between SMEs, was constructed from this information in the following manner: a “0” was assigned to all cells along the superdiagonal, a “1” was entered into a cell for SMEs belonging to the same lab, department and group, a “2” for SMEs belonging only to the same lab and department, a “3” to those SME's belonging only to the same lab, and a “4” to those SMEs in different labs. The Office Address also consists of three characters identifying a SME's office site (two possible sites separated by about 50 miles), floor and wing. A locational distance matrix for SMEs was calculated in the following manner: a “0” was assigned to all entries along the superdiagonal, a “1” was entered in a cell for two SMEs located at the same site, on the same floor, and in the same office wing, a “2” for SMEs occupying only the same site and floor, a “3” for those SMEs only located at the same site, and a “4” to those SMEs located at different sites.

The strength of association between these two distance matrices and each of the two distance matrices and the consensus matrix was tested using Quadratic Assignment [29]. With Quadratic Assignment a correlation statistic γ is computed between the corresponding cells in two matrices of observed data. Then one of these matrices is repeatedly permuted randomly, each time computing a new γ. A p-value for this randomization test is determined by counting the proportion of times the value of γ computed for the data permutations equaled or exceeded the value calculated for the observed data. The results obtained from Quadratic Assignment testing with the Consensus matrix, and the Organizational and Locational distance matrices, after 1,000 permutations, are given in Table 4.

Table 4. Results from significance testing of relationships between organizational distance, inter-office distance and amount of consensus among SME responses. (Quadratic Assignment with 1,000 permutations used for tests.)

Association	γ (observed)	Proportion As Large
Organization/Location	0.586	0.000
Organization/Consensus	-0.388	0.810
Location/Consensus	-0.187	0.160

Several conclusions can be drawn from these tests. As one might expect, there does seem to be some association between a SME's organizational affiliation and the location of his/her office. However, there exists little evidence to support the claim that either their

organizational affiliation or the location of their office has much to do with what they know about "Online Privacy and Security," though location does seem to influence more what one knows than organizational affiliation, i.e., a possible "water cooler" effect. Another way of putting this is that cross-organizational forums and informal sharing of information among those who experience greater face-to-face contact may contribute more to learning and knowledge formation than hiring practices and interactions structured more strictly along organizational boundaries.

5. Conclusions

The experimental results obtained for the *Schemer* prototype are promising, especially considering that the "correct" answers obtained for the GVU sample were in many cases tentative due to a large, heterogeneous sample. Moreover, some of these answers were derived statistically, with no prior analysis to weed-out items with near equal frequencies of "true" and "false" responses. This finding implies that meaningful answers to difficult and "fuzzy" problems might be obtained more quickly, and with less effort and cost, from the information provided by a few competent SME's, rather than from a very much larger survey sample [13].

So, what does this experiment have to do with the Semantic Web? We believe that it demonstrates a potentially powerful use of consensus for deriving semantically-relevant ontologies from domain experts. While this experiment asked SMEs to evaluate items pertaining to Internet security and privacy, they might instead have been requested to rate terms in a list on the basis of their salience to a knowledge domain, or to rate the relative strength of semantic relationships between terms on this list. The protocol adopted for the present experiment could be applied to analyze SME responses to these items to determine (1) those terms that should be part of a controlled vocabulary, and (2) a standard set of semantic relationships between terms in this vocabulary. Based on these consensus views other items could be developed and evaluated by SMEs to derive defining attributes for terms in the ontology. At each step in this process, Consensus Analysis provides important "reality" checks. The metrics it yields, as computed in this study, more clearly indicate the saliency of the targeted domain to SMEs and provide an opportunity to assess how much domain knowledge is possessed by each SME in the sample. We conclude with the conjecture that, by interviewing even a small number of competent SMEs, ontologies for Web catalog and directory services can be similarly constructed in a manner that best represents the collective wisdom of semantically-specialized communities-of-interest.

6. Future Work

This study provides motivation for future research in four key areas: Information acquisition, knowledge derivation, knowledge representation and knowledge reuse.

Information acquisition. The ubiquity of the Web is encouraging some in the Knowledge Management community to consider the automation of tried-and-tested information gathering techniques, e.g., repertory grids [30]. Many such techniques exist, and it isn't always clear when application of any particular one is appropriate, e.g., see [31, 32, 33]. Thus, there is a need to consider which of the many available information acquisition

techniques are appropriate for gathering the information needed to derive different types of knowledge, e.g., controlled vocabularies, ontologies or production systems, and how best to deploy them electronically. In fact, a taxonomy of knowledge types, with a mapping of acquisition methods to each, is needed.

Fortunately, this study was able to reuse items developed for the *GVU Survey*. But this was only a proof-of-concept. Any meaningful implementation of the *Schemer* System (or another like it) will require support for item development, preferably by incrementally building pools from items submitted by SMEs. As in test development, items will have to be classified by their author's COI, then carefully pretested and analyzed for their discriminability before being added to an item pool. There is also an opportunity for evaluating alternative protocols for presenting items to subjects based on their background and the knowledge domain being tested, and more flexible highly-interactive formats for presenting items to subjects electronically.

Knowledge derivation. Consensus Analysis provides a rigorous framework for deriving knowledge from information acquired from a group of SMEs. However, further refinements of the method are required to accommodate missing information and guessing, different difficulty levels of problems, and to enable appropriate classification of decision-makers into their respective communities-of-interest. These features are particularly important for supporting the idea of acquiring information from SME's incrementally and at their convenience. With regards to this last point, we envision the use of wireless communication devices, e.g., PDAs, as a useful means to acquire information from SMEs in less-intensive, asynchronous sessions.

Knowledge representation. For derived knowledge to be useful, it needs to be represented in a way supported by other tools, but without sacrificing information about the details of its structure and semantics. Hence, the expressiveness and adequacy of existing knowledge representation standards, e.g., KIF [34], KRSL [35], RDF Core [2] and DAML+OIL [3] need to be reviewed and evaluated.

Knowledge reuse. A minimal use of derived knowledge would be to publish it electronically. However, in the case of some knowledge, e.g., controlled vocabularies or ontologies, new services will be required to support rapid integration of this knowledge with other technology. Thus, there is need to further explore new technologies for making knowledge more accessible to end-users and software applications.

7. Summary

This paper described the *Schemer* prototype, a Web-based infrastructure to acquire information from domain experts and process this information with a quantitative technique known as *Consensus Analysis.* This approach yields metrics that determine (1) whether a particular problem domain has salience for a group of subject matter experts, (2) the level of competency for each of the subject matter experts, (3) the consensus view of this group weighted by the competency of its members, and (4) a classification of subject matter experts by their appropriate community-of-interest.

There is an opportunity to harness the same social-cultural processes that fostered the creation, growth and success of the current Web to evolve rich ontologies. These will be the focal point of the "emergent" Semantic Web and will be constructed dynamically based on

consensus processes. Distributed and semantically-rich information spaces, supported by the infrastructure needed to easily navigate them, promise to be the transforming technology of the 21st century. New knowledge derivation techniques, such as consensus analysis, embedded in tools that enable dynamic evolution of ontologies are a critical component of the semantic Web. We see the *Schemer* prototype as an important step in the long march towards realizing a semantic Web infrastructure.

Acknowledgements

We want to thank Mark Rosenstein, Jon Kettenring, Sid Dalal and anonymous reviewers for commenting on earlier drafts of this paper, and Kim Romney, Sue Weller and Steve Borgatti for many fruitful exchanges on Consensus Analysis and its formal foundations. We are also grateful to Tracy Mullen, Marek Fiuk and Chumki Basu for their assistance with implementing the *Schemer* prototype, and to other colleagues in Telcordia Technologies' *Information and Computer Sciences Research* and *Internet Architectures Research* labs for participating in the experiment described in this paper. Both Insightful *S-Plus*® version 5.0 and Analytic Technologies *AnthroPac*® version 4.1 were used to benchmark data analysis.

References

[1] Fensel, D., 2001. Understanding is based on Consensus. Panel on Semantics on the Web. *10th International WWW Conference, Hong Kong*, 2001.

[2] The RDF Core Working Group. http://www.w3.org/2001/sw/RDFCore/

[3] D. Broekstra et al., 2001. Enabling Knowledge Representation on the Web by extending the RDF Schema. *10th International WWW Conference, Hong Kong* 2001.

[4] T. H. Davenport and L. Prusak. 1998. Working Knowledge: How Organizations Manage What They Know. Boston: Harvard Business School Press.

[5] D. Gibson, J. Kleinberg, and P. Raghavan. 1998. Inferring web communities from link topology. *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia*.

[6] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. 1995. Recommending and evaluating choices in a virtual community of use. *Proceedings of the CHI-95 Conference, Denver, CO.*

[7] C. F. Cargill, 1989. Information Technology Standardization: Theory, Process, and Organizations. Bedford, MA: Digital Press.

[8] C. Cargill, 1994. Prologue and Introduction. *Standard View* **2(3)**: (1994) 129.

[9] A. Farquhar, R. Fikes, and J. Rice. 1996. The Ontolingua Server: A Tool for Collaborative Ontology Construction. Knowledge Systems Laboratory, KSL-96-26 (September).

[10] M. M. Turoff and S. R. Hiltz. 1996. Computer-based Delphi processes. In M. Adler and E. Ziglio (eds.), *Gazing into the Oracle: The Delphi Method and its Application to Social Policy and Public Health.* pp. 56-85. London: Jessica Kingsley Publishers.

[11] R. M. Cooke, 1991. *Experts in Uncertainty: Opinion and Subjective Probability in Science.* New York: Oxford University Press.

[12] H. A. Linstone and M. Turoff (eds.). 1975. *The Delphi Method: Technique and Applications.* Reading, MA: Addison-Wesley.

[13] A. K. Romney, S. C. Weller, and W. H. Batchelder. 1986. Culture as consensus: A theory of culture and informant accuracy. *American Anthropologist* **88(2):313-338**.

[14] R. M. Keesing, 1974. Theories of culture. *Annual Review of Anthropology* **3:73-97**.

[15] A. L. Kroeber, 1948. *Anthropology*. New York: Harcourt, Brace.

[16] D. W. Read and C. A. Behrens. 1989. Modeling folk knowledge as expert systems. *Anthropological Quarterly* **62(3):107-120**.

[17] D. W. Read and C. A. Behrens. 1991. Computer representation of cultural constructs: New research tools for the study of kinship systems. In M. S. Boone and J. J. Wood (eds.), *Computer Applications for Anthropologists*. Pp. 228-250. Belmont, CA: Wadsworth Publishing Co.

[18] FGDC. 1994. *Content Standards for Digital Geospatial Metadata (June 8).* Washington, D. C.: Federal Geographic Data Committee.

[19] B. J. Hillman, R. G. Swensson, S. J. Hessel, D. E. Gerson, and P. G. Herman. 1997. Improving diagnostic accuracy: A comparison of interactive and Delphi consultations. *Investigative Radiology* **12:112-115**.

[20] B. W. Boehm, P. Bose, E. Horowitz and M. J. Lee. 1994. Software requirements as negotiated win conditions. *Proceedings of Information CommunityRE (April)* Pp. 74-83.

[21] R. G. D'Andrade, 1981. The cultural part of cognition. *Cognitive Science* **5:179-195**.

[22] W. H. Batchelder and A. K. Romney. 1986. The statistical analysis of a general Condorcet model for dichotomous choice situations. In G. Grofman and G. Owen (eds.), *Information Pooling and Group Decision Making*. Pp. 103-112. Greenwich, CT: JAI Press.

[23] W. H. Batchelder and A. K. Romney. 1988. Test theory without an answer key. *Psychometrika* **53:71-92**.

[24] D. Caulkins and S. Hyatt. 1999. Using consensus analysis to measure cultural diversity in organizations and social movements. *Field Methods* **11(1): 5-26**.

[25] S. C. Weller and N. C. Mann. 1997. Assessing rater performance without a standard using consensus theory. *Medical Decision Making* **17:71-79**.

[26] A. L. Comrey, 1962. The minimum residual method of factor analysis. *Psychological Rep*orts **11:15-18**.

[27] S. S. Schiffman, M. L. Reynolds, and F. W. Young. 1981. *Introduction to Multidimensional Scaling: Theory, Methods and Applications.* New York: Academic Press.

[28] H. M. Blalock, Jr. 1972. *Social Statistics*. New York: McGraw-Hill.

[29] L. J. Hubert, 1987. *Assignment Methods in Combinatorial Data Analysis. Statistics, textbooks and monographs***, (73)** New York: Marcel Dekker, Inc.

[30] J. H. Boose, 1989. A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition* **1(1): 3-37**.

[31] H. R. Bernard, 1988. *Research Methods in Cultural Anthropology*. Newbury Park, CA: Sage.

[32] J. P. Spradley, 1979. *The Ethnographic Interview*. New York: Holt, Rinehart and Winston.

[33] O. Werner and G. M. Schoepfle. 1987. *Systematic Fieldwork* (2 vols). Newbury Park, CA. Sage.

[34] M. R. Genesereth and R. E. Fikes (eds.). 1992. *Knowledge Interchange Format, Version 3.0 Reference Manual*. Computer Science Department, Stanford University, Technical report Logic-92-1.

[35] J. Allen and N. Lehrer. 1992. *Knowledge Representation Specification Language (KRSL), Version 2.0.1 Reference Manual*. Draft of the DARPA/Rome Laboratory Planning and Scheduling Initiative. ISX Corporation.

Appendix A

Items derived from Georgia Tech's Graphics, Visualization and Usability Center's *10th World Wide Web User Survey* on "Online Privacy and Security." Answers in parentheses based on simple "majority view" obtained from survey of 1,482 respondents. (See *http://www.gvu.gatech.edu/gvu/user_surveys/survey-1998-10/*.)

In general, how concerned are most WWW users about security on the Internet (e.g., others reading their email, finding out what websites they visit, etc.)? Keep in mind that in this context "security" can mean privacy, confidentiality, and/or proof of identity for a WWW user or for someone else.

1. Older (50+ years old) WWW users tend to be more concerned than younger users. (F)

2. Experienced (> 4 years experience) WWW users tend to be less concerned than inexperienced users. (T)

In general, how concerned are most WWW users about security in relation to making purchases or banking over the Internet? Keep in mind that "security" can mean privacy, confidentiality, and/or proof of identity for a WWW user or for someone else.

3. Older (50+ years old) WWW users tend to be more concerned than younger users. (F)

4. Experienced (> 4 years experience) WWW users tend to be less concerned than inexperienced users. (T)

One thing that makes it difficult to study Internet security is people's and business' reluctance to report security problems for fear of causing more problems for themselves. In addition, it is not always clear where they should be reported. One idea is to have a "clearinghouse" where security problems can be studied and tracked. Please provide you opinions about how such an idea might be received.

5. Most WWW users would be willing to report a security break-in of their personal machine or network to a clearinghouse that maintained their anonymity? (T)

6. Most WWW users would be willing to report a security break-in of their business machine or network to a clearinghouse that maintained their anonymity? (T)

7. Less than 10% of WWW users have ever had their credit card number stolen (either online or offline)? (F)

8. More than 50% of WWW users are willing to use their credit card on the web? (T)

9. Less than 20% of WWW users have an unlisted phone number? (F)

10. Most WWW users are unwilling to put their name and address in a directory for public access on the Web (e.g. the online equivalent of a phone company's "White Pages")? (F)

11. Most WWW users are willing to conduct banking on the Web without a statement from the bank of the security procedures used? (F)

12. WWW users within the United States are more concerned than those in Europe or elsewhere about conducting business online outside of their own country without a statement of the security procedures used? (T)

13. In general, PRIVACY is more important than CONVENIENCE to most WWW users? (T)

14. WWW users will more likely participate in an "online auction" for something they are interested in purchasing? (F)

15. Most WWW users think using the Internet for shopping and banking would make their life easier? (T)

16. For most WWW users security features are the deciding factor in choosing whether or not to do business with an Internet-based company? (F)

17. Most WWW users believe that metrics to measure "how secure" a specific site is rated would not be of any help or value to them? (F)

When one views a Web page, they issue a request to a machine that returns the page to them. Which of the following information do most WWW users believe is technically possible to record/log about their page request?

18. Their email address (T)
19. Time of the request (T)
20. Their machine address (T)
21. The requested page (T)
22. An identifier that persists across visits to that site (T)
23. The type of browser they are using (T)
24. Their machine's operating system (T)
25. Their geographical location (F)
26. Their screen size (F)

What information would most WWW users agree ought to be collected for each Web page they request?

27. Their email address (F)
28. Time of the request (T)
29. Their machine address (F)
30. The requested page (T)
31. An identifier that persists across visits to that site (F)
32. The type of browser they are using (F)
33. Their machine's operating system (F)
34. Their geographical location (F)
35. Their screen size (F)

Most WWW users would give demographic information to a Web site ...

36. if a statement was provided regarding what information was being collected (T)
37. if a statement was provided regarding how the information was going to be used (T)
38. in exchange for access to the pages on the Web site (F)
39. in exchange for a small discount at the Web site's store or on their products (F)
40. in exchange for some value-added service (e.g., notification of events, etc.) (F)
41. if the data would only be used in aggregate form (i.e., not on an individual basis) (T)

What conditions would cause most WWW users to refrain from filling out online registration forms at sites?

42. Takes too much time (T)

43. Required to give their name (F)
44. Required to give an email address (F)
45. Required to give their mailing address (F)
46. Information is not provided on how the data is going to be used (T)
47. Accessing the site is not worth revealing the requested information (T)
48. The entity collecting the data is not trusted (T)

Recent attention has been given to mass electronic mailings (a.k.a. spammings) which often contain advertisements, political statements, get-rich-quick schemes, etc. Among most WWW users, which of the following policies would most likely find support?

49. The Government ought to pass a law making it illegal. (F)
50. Mass mailing agencies ought to have to pay an 'impact' fee. (F)
51. A blacklist of spammers should be built to allow message filtering. (F)
52. A registry ought to be created which contains a list of those not wishing to receive mass mailings. (F)

53. Most of the time upon receiving a mass mailing, WWW users will read the message, then either send a message back asking not to be included in future mailings, retaliate in some manner (e.g., mailing bombings, denial of service, etc.), or perform some other action. (F)

Most WWW users would support which of the following?

54. New laws to protect privacy on the Internet. (T)
55. The establishment of key escrow encryption (where a trusted party keeps a key that can read encrypted messages). (T)
56. Web sites need information about their users to market their site to advertisers. (T)
57. Content providers have the right to resell information about its users to other companies. (F)
58. A user ought to have complete control over which sites get what demographic information. (T)
59. Magazines to which a WWW user subscribes have the right to sell their name and address to companies they feel will interest that user. (F)
60. WWW users like receiving mass postal mailings that were specifically targeted to their demographics. (F)
61. WWW users like receiving mass electronic mailings. (F)
62. WWW users ought to be able to take on different aliases/roles at different times on the Internet. (T)
63. WWW users value being able to visit sites on the Internet in an anonymous manner. (T)
64. WWW users ought to be able to communicate over the Internet without people being able to read the content. (T)
65. WWW users would prefer Internet payment systems that are anonymous to those that are user identified. (T)
66. Third party advertising agencies should be able to compile usage behavior across different web sites for direct marketing purposes. (F)
67. There ought to be stricter laws to protect children's privacy than adult's privacy on the Internet. (T)

Ontology versioning on the Semantic Web

Michel Klein and Dieter Fensel
Vrije Universiteit Amsterdam
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
michel.klein/dieter@cs.vu.nl

Abstract Ontologies are often seen as basic building blocks for the Semantic Web, as they provide a reusable piece of knowledge about a specific domain. However, those pieces of knowledge are not static, but evolve over time. Domain changes, adaptations to different tasks, or changes in the conceptualization require modifications of the ontology. The evolution of ontologies causes operability problems, which will hamper their effective reuse. A versioning mechanism might help to reduce those problems, as it will make the relations between different revisions of an ontology explicit. This paper will discuss the problem of ontology versioning. Inspired by the work done in database schema versioning and program interface versioning, it will also propose building blocks for the most important aspects of a versioning mechanism, i.e., ontology identification and change specification.

1 Introduction

Ontologies are often seen as basic building blocks for the Semantic Web, as they provide a reusable piece of knowledge about a specific domain. However, those pieces of knowledge are often not static, but evolve over time. Domain changes, adaptations to different tasks, or changes in the conceptualization require modifications of the ontology. The evolution of ontologies causes operability problems, which will hamper the effective reuse.

Support to handle those changes is needed. This is especially important in a decentralized and uncontrolled environment like the web, where changes occur without attunement. Much more than in an controlled environment, this may have unexpected and unknown results. With the rise of the Semantic Web, those uncontrolled changes will have even more impact, because *computers* will use the data. There are no longer humans in the chain that — using a vast amount of background knowledge and implicit heuristics — can spot erroneous combinations due to unexpected changes.

The problem is even worse, because there are a lot of dependencies between data sources, applications and the ontologies. Changes to the latter will thus have far-reaching side effects. It is often not practically possible to synchronism the changes to an ontology with modifications to the applications and data sources that use them. Therefore, a versioning methodology is needed to handle revisions of ontologies and the impact on existing sources.

In this paper, we will explore the problem of ontology versioning and we will propose some elements of a versioning framework. We will first discuss the nature of ontology versioning in Section 2. Because compatibility is a key issue in versioning, Section 3 contains an

analysis of the compatibility between schema's and conforming data. To come up with concrete requirements for a versioning framework, we will look at current practices for handling ontology change in Section 4, in which we will also formulate those requirements. Section 5 will eventually present the proposed baseline for an ontology versioning framework on the web. It will mainly concentrate on identification and referring issues. Section 6 concludes the paper and sketches the directions for further research. Finally, the appendix shows how this ideas may be implemented in RDF Schema and / or DAML+OIL.

2 The problem: versioning of ontologies

Before we can investigate the solutions for ontology versioning, we first have to take a closer look at what it actually is. In a general sense, ontology versioning just means that there are multiple variants of an ontology around. In practice, those variants often originates from changes to an existing variant of the ontology and thus form a derivation tree. It is also possible that different "versions" of ontologies of the same domain are independently developed and do *not* have a derivation relation. In this case, however, we will not use the word "version", but we will see the variants as separate ontologies that describe the domain from a specific *viewpoint* or *perspective*. In our view, ontology versioning is closely related to changes in ontologies.

We define ontology versioning as *the ability to handle changes in ontologies by creating and managing different variants of it*. To achieve this ability, we need a methodology with methods to distinguish and recognize versions, and with procedures for updates and changes in ontologies. This also implies that we should keep track of the relationships between versions.

Focused on ontologies, we can say that a *versioning methodology* provides mechanism to disambiguate the interpretation of concepts for users of the ontology variants, and that it makes the compatibility of the variants explicit. The extend of the changes determines the compatibility between the versions. This implies that the semantic impact of changes should be examined. The central question that a versioning methodology answers is: how to reuse existing ontologies in new situations, without invalidating the current usage.

2.1 Causes of ontology changes

A versioning methodology for ontologies copes with changes in *ontologies*. To examine the causes of changes, we will have to look at the nature of ontologies. According to Gruber (1993), an ontologies is a *specification of a shared conceptualization of a domain*. Hence, changes in ontologies are caused by either:

1. changes in the domain;
2. changes in the shared conceptualization;
3. changes in the specification.

The first type of change is often occurring. This problem is very well known from the area of database schema versioning. Ventrone and Heiler (1991) sketches seven different situations in which changes in a domain (domain evolution) require changes to a database model. An

example of this type of change is the merge of two university departments: this is a change in the real world, which requires that an ontology that describes this domain is modified, too.

Changes in the shared conceptualization are also frequently happening. It is important to realize that a *shared* conceptualization of a domain is not a static specification that is produced once in the history, but has to be reached over time. Fensel (2001) describes ontologies as dynamic networks of meaning, in which consensus is achieved in a social process of exchanging information and meaning. This view attributes a dual role to ontologies in information exchange: they provide consensus that is both a *pre-requisite* for information exchange and a *result* of this exchange process.

An conceptualization can also change because of the usage perspective. Different tasks may imply different views on the domain and consequently a different conceptualization. When an ontology is adapted for a new task or a new domain, the modifications represent changes to the conceptualization. For example, consider an ontology about traffic connections in Amsterdam, with concepts like roads, cycle-tracks, canals, bridges and so on. When the ontology is adapted from a bicycle perspective to a water transport perspective, the conceptualization of a bridge changes from a remedy for crossing a canal to a time consuming obstacle[1].

Finally, a specification change is a kind of translation, i.e., a change in the way in which a conceptualization is formally recorded. Although ontology translation is an important and non-trivial issue in many practical applications, it is less interesting *from a versioning perspective*, for two reasons. First, an important goal of a translation is to retain the semantics, i.e., specification variants should be equivalent[2] and they thus only cause syntactic interoperability problems. Second, a translation is often created to use the ontology in an other context (i.e., an other application or system), which heavily reduces the importance of interoperability questions. Therefore, we will leave specification changes alone and concentrate on support for changes in the semantics of an ontology, caused by either domain changes or conceptualization changes.

2.2 Consequences of the change

Versioning support is necessary because changes to ontologies may cause incompatibilities, which means that the changed ontology can not simply be used instead of the unchanged version. There are several type of things that may depend on an ontology. Each of these dependencies may cause a different type of incompatibility.

- In the first place, there is the data that conforms to the ontology. In a semantic web, this can be web pages of which the content is annotated with terms from an ontology. When an ontology is changed, this data may get an different interpretation or may use unknown terms.

- Second, there are other ontologies that use the changed ontology. This may be ontologies that are built from the source ontology, or that import the ontology. Changes to the source ontology may affect the resulting ontology.

[1] Actually, for many people this meaning is also an element of the bicycle perspective.

[2] Although in practice a translation often implies a change in semantics, possibly caused by differences in the representation languages. See for a exploration of ontology language differences and mismatches (Corcho and Gómez-Pérez, 2000) and (Klein, 2001).

- Third, applications that use the ontology may also be hampered by changes to the ontology. In the ideal case, the conceptual knowledge that is necessary for an application should be merely specified in the ontology; however, in practice applications also use an internal model. This internal model may become incompatible with the ontology.

All things considered, we see that a versioning methodology is necessary to take care of the following relations:

- between succeeding revisions of one ontology;
- between the ontology and:
  - instance data;
  - related ontologies;
  - related applications.

In the rest of the paper, we will mainly concentrate on the relation between *data sources* and related ontologies. This has two reasons. First, this specific dependency is occurring very frequently at the Semantic Web and therefore forms an urgent problem. Second, because the other dependencies can be considered as more complex cases of the first dependency, we will have to start with the first case to come to the more complex situations in future work.

3 Analysis of compatibility

In Section 2, we discussed the relations between ontologies and depending data sources in general. In this section, we will take a closer look at the compatibility of changed ontologies and data sources.

We can imagine that — when the Semantic Web evolves — there will be various versions of many ontologies around. There will also be a lot of web pages and applications that use (or are intended to use) a specific version of the ontology. Consequently, there are a number of (possible incompatibility) ways of combining versions of ontologies with versions of data sources. Basically, there are three ways to relate ontology versions with data sources: (1) using the intended version of the ontology for a data source, (2) using a newer version of the ontology, and (3) using an older version of the ontology. As the first type of of combination is naturally compatible, we thus have to explore the incompatibility of ontologies and data sources in two directions, which are illustrated in Figure 1. The terms that we use are known from the database schema versioning literature Roddick (1995).

- **prospective use**: the use of data sources that conform to a previous version of the ontology via a newer version of the ontology (*i.e. view the data from a newer perspective*).

- **retrospective use**: the use of data sources that conform to a newer version of the ontology via a previous version of the ontology (*view the data from an older perspective*).

Based on these directions of use, we can now categorize the compatibility of ontology revisions into different types. In this categorization, we examine whether it is valid to use a revision of an ontology on the *set of all possible instances* of an other revision of the ontology. In other words, we assume that the data source that conforms to a specific version of the ontology uses the whole ontology, i.e., all concepts and relations. From a compatibility perspective, this is a worst case scenario. Table 1 shows the types of compatibility.
We can describe the types of revisions as follows:

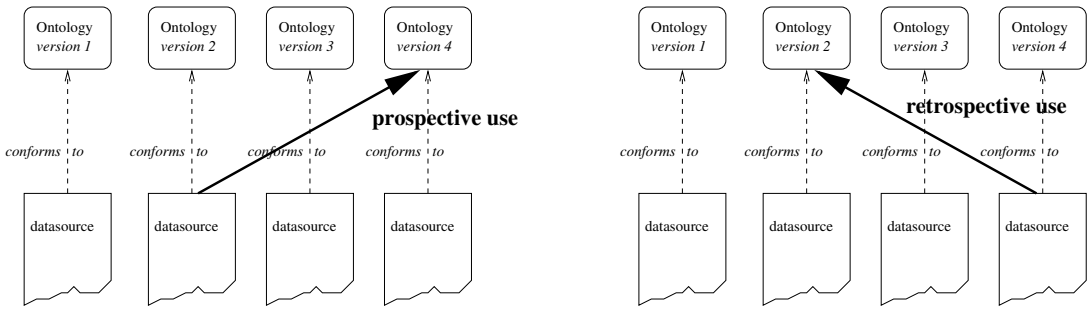


Figure 1: Two examples of prospective and retrospective use of ontologies.

		prospective use valid	
		yes	no
retrospective use valid	yes	full compatible	upward compatible
	no	backward compatible	incompatible

Table 1: Categorization of compatibility

- **full compatible revisions** (upward and backward): the semantics of the ontology is not changed, e.g. syntactic changes or updates of natural language descriptions; this type of change is compatible in both *prospective* use and *retrospective* use.

- **backward compatible revisions**: the semantics of the ontology are changed in such a way that the interpretation of data via the new ontology is the same as when using the previous version of the ontology, e.g. the addition of an independent class; this type of change is compatible in *prospective* use;

- **upward compatible revisions**: the semantics of the ontology is changed is such a way that an older version can be used to interpret newer data sources correctly, e.g. the removal of an independent class; this revision is compatible in *retrospective* use.

- **incompatible revisions**: the semantics of the ontology is changed in such a way that the interpretation of old data sources is invalid, e.g. changing the place in the hierarchy of a class; this type of change is incompatible in both *prospective* use and *retrospective* use.

Notice that both backward compatibility and upward compatibility are transitive: when the changes from $v1$ to $v2$ as well as the changes from $v2$ to $v3$ are backward compatible, then the changes from $v1$ to $v3$ are also backward compatible.

Consequently, if we know that all subsequent *revisions* to an ontology up to a certain version are backward compatible, it is also possible to name the resulting version of the *ontology itself* backward compatible. However, it is never allowed to call a version of an ontology *upward or full compatible*, because the semantics of future versions are not known beforehand! It is always possible that new versions of ontologies will introduce new things that cannot correctly be interpreted via older ontology versions. Thus, *backward compatibility* can be a characteristic of an ontology, but *upward or full compatibility* can not.

In the characterization above, we looked at compatibility from a theoretic perspective, considering data sources that consist of all possible instances of the ontology to which they

conform. As we already said, this is a worst case scenario. In practice, much more combinations of ontology versions and data sources yield a valid interpretation of the data than can be assumed from the schema in Table 1.

For example, although the table depicts that an incompatible revision can not be used for prospective interpretation (to interpret data sources that conforms to an older version), it is very well possible that a specific prospective use is valid, i.e., that the ontology can be used to correctly interpret some data that is committed to a previous version. This occurs when the *ontological commitment* of a data source, i.e., that part of the ontology that is actually "used" by a data source[3], is a subset of the complete "ontology" that is not affected by the revisions.

It is very much required that versioning methodologies and techniques for the Semantic Web exploit this "practical" compatibility where possible. That is, the techniques should not only determine whether a specific combination an ontology version and a data source version provides a valid interpretation *in general*, but — even when the combination is not generally valid — try to use as much knowledge as possible. We think that such techniques for "maximal exploitation" are essential for the development of the Semantic Web.

4 Current practices and requirements

To come up with concrete requirements for an ontology versioning mechanism on the web, we will now look at the current practices for managing changes in web-ontologies. We will describe a few typical scenarios for ontology change and explore the effects of those scenarios on two examples.

Currently, there is no agreed versioning methodology for ontologies on the web. However, in an decentralized and uncontrolled environment like the web, changes are certainly needed and do occur! When we look at the current practices, we can sketch several scenarios for ontology changes.

1. The ontology is silently changed; the previous version is replaced by the new version without any (formal) notification.

2. The ontology is visibly changed, but only the new version is accessible; the previous version is replaced by the new version.

3. The ontology is visibly changed, and both the new version and the previous version are accessible.

4. The ontology is visibly changed, both the new version and the previous version are accessible, and there is an explicit specification of the relation between concepts of the new version and the previous version.

4.1 Simple example

To come to concrete requirements for a versioning methodology, we will now look at the effects of these scenarios on the compatibility when an ontology changes. As an example we use an ontology of the education system in the Netherlands and web pages that are annotated with this ontology.

[3] This definition and the use of the term "ontology" is quite sloppy.

In the distant past, there was only one type of higher education, which was called an "University". A small part of an ontology that describes this looks as follows:[4]

class-def Education
class-def AcadamicEducation
  subclass-of Education
class-def Higher-Education-Institute
class-def University
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value AcadamicEducation

Many years later, a new type of higher-education was introduced, which provides professional education. This type was called "HBO". The above ontology has to be extended with the following three definitions. This addition is a monotonic extension to the ontology and the new version can be considered as being backward compatible with the first version.

class-def ProfessionalEducation
  subclass-of Education
disjoint AcadamicEducation ProfessionalEducation
class-def HBO
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value ProfessionalEducation

Let us suppose that there are a lot of web pages about education in the Netherlands around, which are annotated using the first version of the ontology. We will now try to interpret this information using the second version of the ontology (prospective use). We describe the effects of the change for each of the scenarios that are listed above.

Ex. 1, ad 1 When we have completely no clue that the ontology is changed, we encounter — in this backward compatible case — no problems at all. The terms that are used in the data source are the same as those in the ontology, and they also have the same meaning. A "University" on a web page is correctly interpreted.

Ex. 1, ad 2 When we know that the ontology is changed, but we don't know anything about the previous version of the ontology, nothing is sure anymore! Definitions could be changed and we can not derive that the term "University" on a web page (using ontology version 1) is the same as our definition.

Ex. 1, ad 3 In case the previous ontology can be accessed, we can compare and relate the ontologies, and see whether the changes interferes with the semantics of existing terms. In this case, we could have derived that the concept of "University" is not changed, and that the data sources can still correctly be interpreted.

[4]We use the "presentation syntax" of OIL (Fensel et al., 2000a) to represent the ontology; the interpretation is more or less straightforward. We could also have used DAML+OIL, but this would have required much more space.

Ex. 1, ad 4 If the relation between the concepts is explicitly specified, it would be clear that the the new version is backward compatible with the previous version, because the new version only adds a concepts. We could then safely conclude that the interpretation of previous data is still valid.

Notice that in the last three scenarios it is important to know which version of the ontology is used to annotate the data sources. This should me made explicit in some way.

4.2 More complicated example

In the year 2000, the Dutch government decided that both professional and academic institutes for higher-education are allowed to call themselves "University". This implies a new change to our ontology, resulting in a third version. This version is in general incompatible with the previous version. In the new version, the definition of "University" is changed to:

class-def University
  subclass-of Higher-Education-Institute
  slot-constraint *type-of-education*
    has-value (ProfessionalEducation **or** AcadamicEducation)

Let us again look at the consequences of this change with current versioning practices:

Ex. 2, ad 1 When we do not know that the ontology is changed, we use an other interpretation of a "University" than intended. Because in this case, a University-v1 is a subclass of University-v2, the interpretation of data is not incorrect, but also not complete. We cannot interpret that every "University" in the data sources actually provide academic education.

Ex. 2, ad 2 Same problem as with the change in the previous example.

Ex. 2, ad 3 If we have both version of the ontologies, we could compare them and see that only the definition of "University" is changed. We can see that both versions are subclasses of "Higher-Education-Institute", and interpret all instances of the old "University" as instance of "Higher-Education-Institute". This is again correct but incomplete. It ignores some knowledge that is available. Notice that, in this case, smart agents (agents capable of performing OIL classification) can derive that both Universities and HBOs are subclasses of new universities. Figure 2 shows[5] the classes in our example before and after classification.

Ex. 2, ad 4 In the case in which the relations between the concepts in the two ontologies are explicitly specified, it would tell us that "University" in the new ontology subsumes both "HBO" and "University" in the previous version.

It is also worth to notice that, in the last two scenarios, it is necessary to be able to distinguish between the different version of a concept. As the definition of "University" is changed, we need a separate identifier for each version of the definition. Otherwise, it is not possible to relate the previous and new definition to each other. In Figure 2, this is temporarily solved by appending "-v2" to the concept name.

[5] Modeled with the OILed tool, `http://img.cs.man.ac.uk/oil/`.

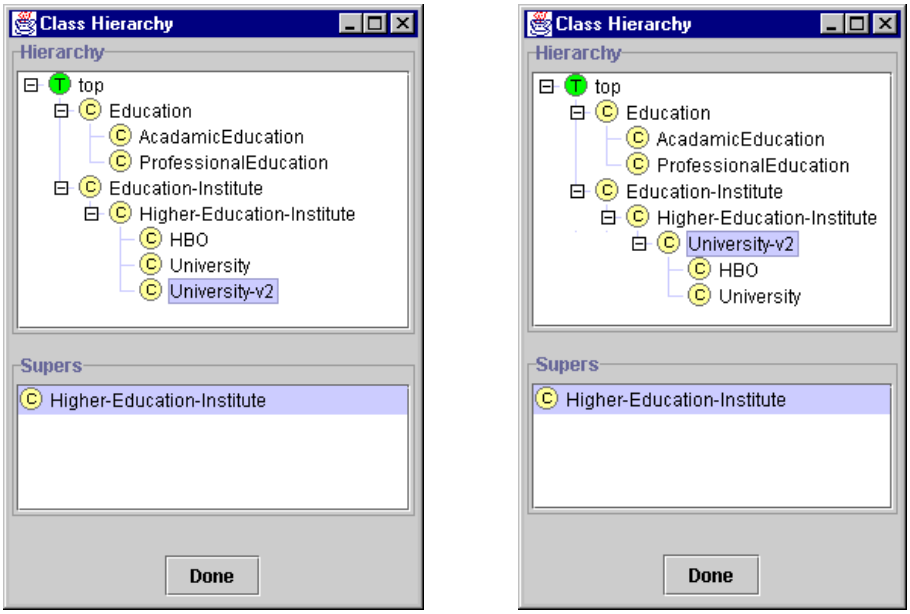


Figure 2: The hierarchy of the example ontology before and after classification with FaCT.

4.3 Observations

Based on the examples above, we can make a few observations. First, changing an ontology without any notification *may* result in a correct interpretation of the data. This is the case when the modification in the ontology does not affect the existing definitions, i.e., when the change is a monotonic extension. Heflin and Hendler (2000) show that the addition of concepts or relations are such extensions. When used on a data source, ontologies that are extended in this way yield the same perspective as when the original ontology is used. The interpretation is also valid when the revised concepts subsumes the original concepts. However, although the interpretation is correct in this case, it is only partial: not all the knowledge is exploited.

Because many changes in ontologies consist just of additions of concepts, it is understandable that the first scenario of ontology change is sometimes used. It is, however, not difficult to think of an change that — in this scenario — will result in an *invalid* interpretation, e.g., every change that restricts the extension of a class. This scenario should therefore be prevented.

Second, we see that it can be beneficial to have access to older versions of the ontology. This allows to compare the definitions and judge the validness of definitions used with other versions the data. In case of a cleanly modeled ontology, [6] it is even possible to have some automate support for this, e.g. by using the FaCT classifier[7].

As a third observation, we see knowledge about the relation concepts of different versions may yield in a partial but correct interpretation of the data. This relation can either be

[6]That is, the definitions of concepts should state whether they are necessary or necessary and sufficient; in Description Logic parlance: primitive or defined. This way of modeling is more naturally in OIL than in DAML+OIL, as the second requires that a defined concept is modeled as an equivalence to a couple of restrictions. It is therefore questionable whether in practice DAML+OIL ontologies can benefit much from the classification support. See also the discussion on this topic on the RDF-Logic mailing-list: `http://lists.w3.org/Archives/Public/www-rdf-logic/2001Mar/0000.html`.

[7]`http://www.cs.man.ac.uk/~horrocks/FaCT/`

manually specified, or can partly be derived, as described in the previous paragraph. For the specification of the relation between concepts of different versions, we need a identification mechanism for individual concepts of an ontology version. If we cannot refer to a specific version of a concept, this specification is not possible.

4.4 Requirements for versioning framework

Now we have explored the problems around ontology versioning, we can formulate requirements and wishes for a versioning methodology. We will first define the general goal for a versioning framework.

> A versioning methodology should provide mechanisms and techniques to manage changes to ontologies, while achieving maximal interoperability with existing data and applications. This means that it should retain as much information and knowledge as possible, without deriving incorrect information.

Notice that this is much stricter than just specifying whether it is valid to use a certain version of an ontology with a data source.

The general goal can be detailed in a number of more specific requirements. We impose the following requirements on a versioning framework, in increasing level of difficulty:

- for every use of a concept or a relation, a versioning framework should provide an unambiguous reference to the intended definition; (**identification**)
- a versioning framework should make the relation of one version of a concept or relation to other versions of that construct explicit; (**change specification**)
- a versioning framework should — as far as possible — automatically translate and relate the versions and data sources, to enable transparent access. (**transparent evolution**)

5 Building blocks for a versioning methodology

After we have stated the problems and requirements, we will now provide elements of a versioning methodology. We will concentrate our discussion on ontology identification and change specification.

5.1 Ontology identification on the web

Identity of ontologies The first question that has to be answered when we want to identify versions of an ontology on the web is: what is the identity of an ontology? This is not as trivial as it seems. In Section 3, we already anticipated this question by stating that syntactic changes and updates of natural language descriptions are fully compatible revisions. However, this is very debatable! If an ontology is seen as a specification of a conceptualization, then every modification to that specification can be considered a new conceptualization of the domain. In that case, the descriptions specify different concepts, which are *per definition* not equal.

Looking at this from another perspective, one might regard an ontology primarily as a conceptualization, which is represented as complete as possible in a specification. In this

case one could argue that an update to a natural language description of a concept is not a semantic change, but just a refined description of the same conceptualization.

In this philosophical debate, we take the following (practical) position. We assume that an ontology is represented in a file on the web. Every change that results in a different character representation of the ontology constitutes a revision. In case the logical definitions are not changed, it is the responsibility of the author of the revision to decide whether this revision is semantic change and thus forms an new conceptualization with its own identity, or just an change in the representation of the same conceptualization.

Identification on the web The second question is: how does this relate to web resources and there identity? This is brings us into the very slippery debate on the meaning of URIs, URNs and resources (see Champin et al., 2001, and the discussion on the RDF Interest mailing-list that followed its publication). The main questions in this discussion are: what is a resource and how should it be identified. However, we will circumvent these questions and approach the problem from another direction: are the "entities" in our domain (i.e., the entities in the domain of ontology versions, e.g. a conceptualization, a revision, a specification) resources and can we give them an identifier?

These questions are relatively easy to solve! According to the definition of Uniform Resource Identifiers (URI's) (defined in Berners-Lee et al., 1998), "a resource can be anything that has identity". In (Berners-Lee, 1996) is stated: a "resource" is a conceptual entity (a little like a Platonic ideal). Both definitions comprise our idea of an ontology. Hence, an ontology can harmlessly be regarded as a resource. An URI, which "is a compact string of characters for identifying an abstract or physical resource" (Berners-Lee et al., 1998) can be used to identify the resources. Notice that URI's provide a general identification mechanisms, as opposed to Uniform Resource Locators (URL's), which are bound to the *location* of a resource.

The important step in our proposed method is to separate the identity of ontologies completely from the identity of files on the web that specify the ontology. In other words, the class of ontology resources should be distinguished from the class of file resources. As we have seen above, a revision — which is normally specified in a new file — *may* constitute a new ontology, but this is no automatism. Every revision is a new file resource and gets a new file identifier, but does not automatically get a new ontology identifier.

Notice that we are at this point not compliant with the RDF Schema specification (Brickley and Guha, 2000), which states:[8]

> this specification recommends that a new namespace URI should be declared whenever an RDF schema is changed.

This is recommendation seems too strong, because it also advises to use new URI's when only small corrections are made that do not affect the meaning, i.e., when the conceptualization is not changed. This has already caused problems. For example, the Dublin Core working group changed the URI of their meta-data term definitions when they published a new version with more precisely stated definitions. This has caused a lot of annoyance in the library community, who had to work with several URI's for equivalent concepts. Eventually, the DC steering committee decided to use one URI for all the versions of their definitions.

[8] Although we might be *intentionally* compliant, because the RDFS specification argues that changing the *logical* structure might break depending models. This recommendation could thus be interpreted as only valid for logical changes.

Baseline of an identification method When we take into account all these considerations, we propose an identification method that is based on the following points:

- a distinction between three classes of resources:
  1. files;
  2. ontologies;
  3. lines of backward compatible ontologies.
- a change in a file results in a new file identifier;
- the use of a URL for the file identification;
- only a change in the conceptualization results in a new ontology identifier;
- a new type of URI for ontology identification with a two level numbering scheme:
  - minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology);
  - major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies);
- individual concepts or relations, whose identifier only differs in minor number, are assumed to be equivalent;
- ontologies are referred to by an ontology URI with the according major revision number and the *minimal extra commitment*, i.e., the lowest necessary minor revision number.

The ideas behind these points are the following. As already pointed out in the beginning of this section, the distinction between ontology identity and file identity has the advantage that file changes and location changes (e.g., copy of an ontology) can be isolated from ontological changes. By using a new type of URI, it is possible to encode all the information in it that is necessary for our usage, and it also prevents confusion with URL's that specify a location.

The distinction between individual ontologies on the one hand and lines of backward compatible ontologies on the other hand, provides a simple way to indicate a very general type of compatibility, likewise the "BACKWARD-COMPATIBLE-WITH" field in SHOE (Heflin and Hendler, 2000). The distinction we make is also in line with the idea of "levels of generality", which is discussed in (Berners-Lee, 1996). Applications can conclude directly — without formal analyses or deduction steps — that a version can be validly used on data sources with the same major number and a equal or lower minor number. To achieve a maximal backward compatibility, we also propose that not the minor number of the newest revision is specified in a data source, but the minimal addition to the base version that is used by this data source. For example, suppose an ontology with concepts A, B and C. Version 1.1 added a concept D and version 1.2 added concept E. Then a data source data only relies on concepts A, C and D, would specify its commitment only to version 1.1, although there is already a version 1.2 available. We adopted this idea from software-program library versioning, as described in (Brown and Runge, 2000).

An interesting point for discussion is whether it would be possible to specify the *real* ontological commitment, instead of only the necessary extra commitment. This could lead to even more detailed decisions on compatibility. In our example, this would mean that the data sources specifies that it relies on exactly A, C and D. This would require a different type of identification.

The point that states that individual concepts with a identifier that only differs in minor number are considered to be equivalent, is necessary to actually enable the backward compatibility. By default, all resources on the web with a different identifier are considered to different. This statement allows the creation of a stand-alone ontology revision, which has concepts that are equal to a previous version.

5.2 Change specification and transparent evolution

For change specification and transparent evolution, there are two important requirements. First, because of the suggested practice of referring to the minimal extra addition, the changes in a line of backward compatible ontologies should be easily recognizable and identifiable. Actually, the additions from one version to another together form a *class* of descriptions. Our suggestion is to make this explicit by adding a class `Additions<Major>.<Minor>`, e.g., `Additions1.2`, of which the new descriptions are an instance. This makes that class a unique identifier for the set of additions in a certain revision. The additions can be retrieved by asking for all instances of a specific "Addition" class.

Second, the relation to a previous version should explicitly be specified. One aspect of this specification is a pointer to the ontology from which it is derived. These pointers together form a lattice of versions that can be used to deduce the derivation relation from one version to an arbitrary other version of the *ontology*. A second aspect of this specification is the relation between *concepts and relations* in the previous and current version of the ontology. As concepts and relations that do not have the same major number in their identifier are assumed to be different, this specification should both specify equivalence relations as subsumption relations. However, although a lot of relations between revisions of concepts can be specified with "subclass-of" and "equivalence" relations, a more extensive (rule-)language is necessary to enable the efficient specification of the relations, e.g. a language with quantifiers.

We have seen that an explicit specification of the relation between concepts allows to retain as much information as possible. The relation between two revisions of an ontology can be specified in a separate translation ontology. Both the previous version of the ontology and the translation ontology should be linked from the new version, to allow the automatic collection of all the statements that specify the relation.

6 Conclusions and further work

In this paper, we have explored the problem of ontology versioning in a web based context. We discussed the nature of ontology changes and looked at the consequences. Ontology versioning shares some characteristics with database schema versioning and program library versioning, but also as some peculiarities which are specific for the web based context, mainly introduced by the decentralized nature of the web.

After examining the effects on compatibility of a few example scenarios for ontology versioning, we have sketched some elements for a versioning framework for ontologies. This elements are mainly focused on identification and referring. Our main goal in the design of a framework is to achieve "maximal use" of the available knowledge. This implies that is is not sufficient to find out whether a specific interpretation of a ontology on data is invalid, but that we try to derive as much valid information as possible.

The ideas that are presented in this paper will be implemented in the On-To-Knowledge

project (Fensel et al., 2000b), which builds an ontology-based tool environment to perform knowledge management, dealing with large numbers of heterogeneous, distributed, and semi-structured documents typically found in large company intranets and the World-Wide Web.

The work described in this paper is still ongoing. There are a lot of things that are not yet done. We think that the most important flaw is the lack of a detailed analysis of the effect of specific changes on the interpretation of data. This could be done in the same line as (Banerjee et al., 1987), where effects of schema changes on OO databases are analyses. This analyses should also cover the problem of data that becomes inconsistent. Further, the role of time is also not taken into account. Finally, a clear example is needed to illustrate the proposals.

Eventually, this work will result in a versioning framework that gives very detailed procedures to allow evolving ontologies, while achieving maximal compatibility.

References

Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record (Proc. Conf. on Management of Data)*, 16(3):311–322.

Berners-Lee, T. (1996). Generic resources. Design Issues.

Berners-Lee, T., Fielding, R., and Masinter, L. (1998). RFC 2396: Uniform Resource Identifiers (URI): Generic syntax. Status: DRAFT STANDARD.

Bray, T., Hollander, D., and Layman, A. (1999). Namespaces in xml. `http://www.w3.org/TR/REC-xml-names/`.

Brickley, D. and Guha, R. V. (2000). Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium.

Brown, D. J. and Runge, K. (2000). Library interface versioning in solaris and linux. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, Georgia.

Champin, P.-A., Euzenat, J., and Mille, A. (2001). Why urls are good uris, and why they are not. `http://www710.univ-lyon1.fr/~champin/urls.pdf`.

Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *Proceedings of the AAAI'97*, pages 369–376.

Corcho, O. and Gómez-Pérez, A. (2000). A roadmap to ontology specification languages. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pages 80–96, Juan-les-Pins, France. Springer-Verlag.

Fensel, D. (2001). Ontologies: Dynamic networks of formally represented meaning. Rejected for SWWS.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. (2000a). OIL in a nutshell. In Dieng, R. and Corby, O., editors, *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International*

Conference EKAW 2000, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pages 1–16, Juan-les-Pins, France. Springer-Verlag.

Fensel, D., van Harmelen, F., Klein, M., Akkermans, H., Broekstra, J., Fluit, C., van der Meer, J., Schnurr, H.-P., Studer, R., Hughes, J., Krohn, U., Davies, J., Engels, R., Bremdal, B., Ygge, F., Lau, T., Novotny, B., Reimer, U., and Horrocks, I. (2000b). On-to-knowledge: Ontology-based tools for knowledge management. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain.

Foo, N. (1995). Ontology revision. In Ellis, G., Levinson, R., Rich, W., and Sowa, J. F., editors, *Proceedings of the 3rd International Conference on Conceptual Structures (ICCS'95): Applications, Implementation and Theory*, volume 954 of *LNAI*, pages 16–31, Berlin, GER. Springer.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2).

Heflin, J. and Hendler, J. (2000). Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449. AAAI/MIT Press, Menlo Park, CA.

Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In Gomez-Perez, A., Gruninger, M., Stuckenschmidt, H., and Uschold, M., editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA.

Oliver, D. E., Shahar, Y., Musen, M. A., and Shortliffe, E. H. (1999). Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine*, 15(1):53–76.

Pinto, H. S., Gómez-Pérez, A., and Martins, J. P. (1999). Some issues on ontology integration. In *Proceedings of the Workshop on Ontologies and Problem Solving Methods during IJCAI-99*, Stockholm, Sweden.

Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393.

Roddick, J. F., Craske, N. G., and Richards, T. J. (1994). A taxonomy for schema versioning based on the relational and entity relationship models. *Lecture Notes in Computer Science*, 823:137–??

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236. Also published in/as: Bellcore, TM-STS-016302, Jun.1990.

Ventrone, V. and Heiler, S. (1991). Semantic heterogeneity as a result of domain evolution. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(4):16–20.

A Implementation in DAML+OIL

This appendix gives a rough sketch how the ideas about identification and change tracking might be used in RDF Schema or DAML+OIL. In both languages it is a common practice to use the URL (location) of the ontology as its identifier. This URL is then defined as a namespace, which makes the terms in the ontology identifiable. However, it is not required for the namespace mechanism that *URL's* are used. A namespace is defined as follows (Bray et al., 1999):

> [Definition:] An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names.
>
> ...
>
> [Definition:] URI references which identify namespaces are considered identical when they are exactly the same character-for-character. Note that URI references which are not identical in this sense may in fact be functionally equivalent.

This means that actually *URI's* function as identification mechanism.[9] It is therefore easy to adapt this mechanism for our ontology identification mechanism, which uses a separate URI for ontology identity. We should then first design a URI for ontologies.

According to the URI definition, a "generic URI" has the following format: `<scheme>://<authority><path>?<query>`. Champin et al. (2001) pointed out that the use of a URL as identification guarantees that the publisher has the authorization to use that specific part of the URL namespace. This advantage can be retained when the `<authority>` and `<path>` component in the new URI scheme also use the server name and path to the part of the namespace that is owned by the publisher.

Our suggestion would be to use `ontology` as a name for the scheme, constitute the `<authority>` and `<path>` from the server name and path of a URL, and use the major (and probably a minor number) as last two elements of the path. A typical identifier for an ontology would look as follows: `ontology://www.cs.vu.nl/~{}mcaklein/ontology/example/2/1/`, while an line of backward compatible ontologies is identified by `ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/`.

There is still one important open question. Using URL's as ontology identifiers have the advantage that localization of the file that specifies the ontology is trivial. With separate URI scheme, we cannot use the URI of the ontology to locate the file that specifies it. There are several solutions for this problem. One would be to have some kind of lookup system, like a DNS for host names and IP-numbers. A practical solution would be to provide a direct mapping from ontology URI's to file URL's. For example, one could agree that when "`ontology`" is replaced by "`http`" in a ontology URI, one acquire a URL of a directory with specifications

[9] Remember that URL's are a subset of URI's. The URI definition Berners-Lee et al. (1998) says:

> A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

of the ontology (conceptualization), and that appending `newest.rdfs` to the directory URL yields the URL of the most recent specification. Although we now show that there are several implementation possible, the versioning framework should specify exactly how people and applications should behave with this respect.

Our suggestion is to extend the meta-information in a DAML+OIL ontology also with the *location* of the ontology specification. Together with an identifier, a pointer to the previous version and the the "translation" ontology, a piece of an DAML+OIL ontology might look as follows:

```
<Ontology rdf:about="">
  <identifier>ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/</identifier>
  <derivation>
    <from rdf:resource="ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/" />
    <relation rdf:resource="ontology://www.cs.vu.nl/~mcaklein/ontology/example/2/1/delta/" />
  </derivation>
  <location>http://www.cs.vu.nl/~mcaklein/ontology/example/2/1/base.rdfs</location>
  <info>${Id}: base.rdfs,v 1.15 2001/05/22 10:26:46 mcaklein Exp $</info>
</Ontology>

...

<rdfs:Class rdf:ID="Additions2.1">
</rdfs:Class>

<rdfs:Class rdf:ID="HBO">
  <rdf:type rdf:resource="#Additions2.1"/>
</rdfs:Class>
```

Ontology Library Systems: The key to successful Ontology Re-use

Ying Ding & Dieter Fensel
Division of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, the Netherlands
www.cs.vu.nl/~ying,~dieter

Abstract

Increasingly, effort has been devoted to surveying ontology-related research studies from various aspects. However, no survey is available for the ontology library system. For this reason, we decided to examine existing library systems in this paper. First, we identified the main criteria (management, adaptation, and standardization) for evaluating the functionality of the library systems. Then, based on the further enriched criteria, we surveyed most existing ontology library systems. Finally, we summarized the comparison and proposed various important requirements for structuring ontology library systems. The ontology library systems surveyed include: WebOnto, Ontolingua, DAML Ontology Library System, SHOE, Ontology Server, IEEE Standard Upper Ontology, OntoServer and ONIONS.

1. Introduction

With the rapid development of the World Wide Web, the amount of available information online has increased exponentially. A lack of standardization and common vocabulary has continued to generate heterogeneity, which strongly hinders information exchange and communication. *Ontologies*, which capture the semantics of information from various sources and giving them a concise, uniform and declarative description, have gained significance due to the demands in academia and industry [1]. As the number of different ontologies is on the increase, the task of maintaining and re-organizing them in order to facilitate the re-use of knowledge is challenging. A breakthrough in ontology technology would require methodological aids and tools that enable effective and efficient development. A key aspect in achieving this is successful re-use of ontologies. Being developed for supporting knowledge sharing and reuse, it is the lack of proper support of ontology re-use that hampers a broader dissemination of the ontology. *Ontology library systems* are an important tool in grouping and re-organizing ontologies for further re-use, integration, maintenance, mapping and versioning.

An *Ontology library system* is a library system that offers various functions for managing, adapting and standardizing groups of ontologies. It should fulfill the needs for re-use of ontologies. In this sense, an ontology library system should be easily accessible and offer efficient support for re-using existing relevant ontologies and standardizing them based on upper-level ontologies and ontology representation languages. For this reason, an ontology library system will, at the very least, feature a functional infrastructure to store and maintain ontologies, an uncomplicated adapting environment for editing, searching and reasoning ontologies, and strong standardization support by providing upper-level ontologies and standard ontology representation languages.

Recently, increasing effort has been devoted to surveying ontology-related research studies from various aspects, including that of ontology representation languages [2], ontology development [3], and ontology learning approaches [4]. However, no survey has been made of ontology library systems. This prompted us to examine the existing library systems in this paper. We will identify the main criteria for evaluating their functionality. We will also

carefully evaluate existing proposals according to these requirements. In order to facilitate ontology re-use, a library system must, at the very least, support the following: (see Figure 1):

- ontology re-use by open *storage*, *identification* and *versioning*.
- ontology re-use by providing smooth *access* to existing ontologies and by providing advanced support in *adapting* ontologies to certain domain and task-specific circumstances (instead of requiring such ontologies to be developed from scratch).
- ontology re-use by fully employing the power of *standardization*. Providing access to *upper-layer ontologies* and standard *representation languages* is one of the keys to developing knowledge sharing and re-use to its full potential.

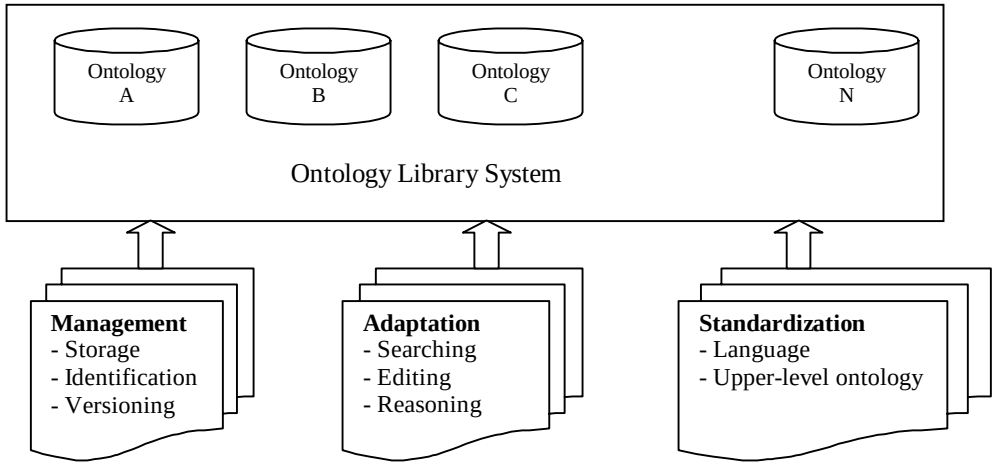


Figure 1. General overview of the structure of the survey

The aspects above can be further specified as the follows:

Management. The most important function of an ontology library system is that of facilitating the re-use of knowledge (ontologies). After all, the main purpose of ontologies is to enable knowledge sharing and re-use [5]. Important aspects of the re-use functionality of an ontology library system are open storage, identification, and versioning support.

- **Storage** (how to store the ontology): (a) Is the ontology easily accessible (via a client/server architecture, Peer-to-Peer, etc.) in supporting remote access and editing?; (b) Are ontologies classified according to some existing or homemade categories? (Classifying ontologies is an important step in reorganizing them such that users can easily search and identify relevant ontologies. It emphasizes the library system function of reorganizing ontologies); and (c) Are ontologies stored in modules? (The modularity structure of an ontology library system can facilitate the process of re-use, mapping and integration; it guarantees proficient ontology re-use).
- **Identification** (how to uniquely identify an ontology): Each ontology must have a unique identifier in the ontology library system.
- **Versioning** (how to maintain the changes of ontologies in an ontology library system): Versioning is very critical in ensuring the consistency among different versions of ontologies.

Adaptation. Ontology library systems should make facilitate the task of extending and updating ontologies. They should provide user-friendly environments for searching, editing and reasoning ontologies. Important aspects in an ontology library system include support in finding and modifying existing ontologies.

- **Searching** (how to search ontology from the ontology library system): Does a library system provide certain searching facilities, such as keyword-based searching or other advanced searching? Does it feature an adequate browsing function?

- **Editing** (how to add, delete and edit specific ontologies in the ontology library system. One of the most important features that an ontology library system should have is one that modifies stored ontologies or adds new ontologies): How does the system support the editing function? Does it support remote and cooperative editing?
- **Reasoning** (how to derive consequences from an ontology): How does the system support ontology evaluation and verification? Does it make it possible to derive any query-answering behavior?

Standardization. Ontology library systems should follow existing or available standards, such as standardized ontology representation languages and standardized taxonomies or structures of ontologies.

- **Language** (the kind of standard ontology language used in the ontology library system, for instance, RDFs[1], XMLs[2] or DAML+OIL[3]): Does the system only support one standard language or other different languages?
- **Upper-level ontologies** (Is the ontology library system 'grounded' in any existing upper-level ontologies, such as Upper Cyc Ontology, SENSUS, MikroKosmos, the PENNMAN Upper Model, and IEEE upper-layer ontology?): The upper-level ontology captures and models the basic concepts and knowledge that could be re-used in creating new ontologies and in organizing ontology libraries.

This survey report is structured as the follows. We will begin by examining current ontology library systems in light of the aspects outlined above. Next, we will provide a summary of our comparison of these systems. Finally, we will discuss various important requirements for structuring ontology library systems.

2. State-of-the-art Survey

This section surveys current important ontology library systems. These include: WebOnto[4] (Knowledge Media Institute, Open University, UK), Ontolingua [5] (Knowledge Systems Laboratory, Stanford University, USA), DAML Ontology library system[6] (DAML, DAPAR, USA), SHOE[7] (University of Maryland, USA), Ontology Server[8] (Vrije Universiteit, Brussels, Belgium), IEEE Standard Upper Ontology[9] (IEEE), OntoServer[10] (AIFB, University of Karlshruhe, Germany) and ONIONS[11] (Biomedical Technologies Institute (ITBM) of the Italian National Research Council (CNR), Italy). ONIONS is a methodology for ontology integration and was successfully implemented in several medical ontology library systems. Strictly speaking, it is not an ontology library system. However, since it defines various criteria for developing such a system, we will examine it briefly here. There are many more ontology library systems than we included in our comparison. We have only included approaches that are publicly available as those offer enough detailed information to enable us to evaluate their actual functionalities.

2. 1 WebOnto

WebOnto is an ontology library system developed by the Knowledge Media Institute of the Open University (UK) [6]. It is designed to support the collaborative creating, browsing and editing of ontologies. It provides a direct manipulation interface displaying ontological expressions and also an ontology discussion tool called Tadzebao, which could support both asynchronous and synchronous discussions on ontologies ([7], see Figure 2).

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/XML/
[3] http://www.ontoknowledge.org/oil/oilhome.shtml
[4] http://eldora.open.ac.uk:3000/webonto
[5] http://www-ksl-svc.stanford.edu:5915/
[6] http://www.daml.org/ontologies/
[7] http://www.cs.umd.edu/projects/plus/SHOE/
[8] http://www.starlab.vub.ac.be/research/dogma/OntologyServer.htm
[9] http://suo.ieee.org/refs.html
[10] http://ontoserver.aifb.uni-karlsruhe.de/
[11] http://saussure.irmkant.rm.cnr.it/onto/

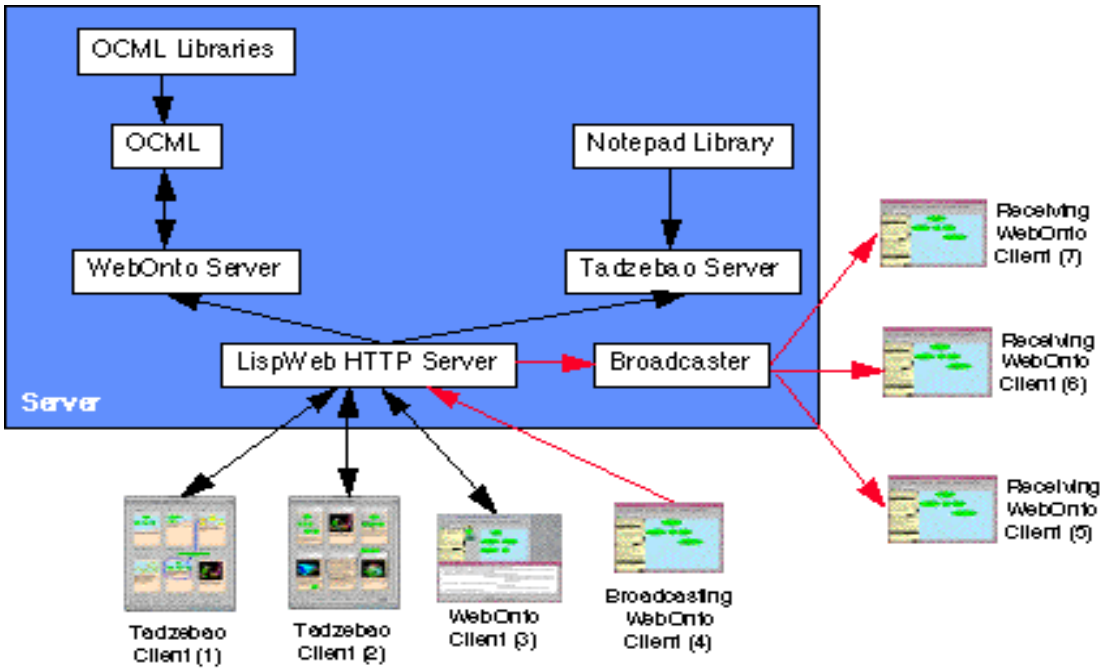


Figure 2. The architecture of the Tadzebao and WebOnto Server [6].

2.1.1 Management

Storage. WebOnto relies on a client/server-based *architecture*. The servers are responsible for storing and maintaining ontologies and user dialogues, the clients are the interfaces to access the stored ontologies. Ontologies stored in the WebOnto are not *classified* according to some existing categories. Ontologies are divided into small units (for instance, ontology is the tree-structural of classes, and the small unit is the class and its parents). They are then stored in a specific *Module* containing name, type, and the names of class parents. This system can draw graphical representations of ontologies based on the modularity storage.
Identification. Ontologies stored in the WebOnto library system are identified by their unique names. Even though an ontology is divided into small units, each unit contains the name, type, and the names of the class parents.
Versioning. WebOnto only mentioned that ontologies can be inherited from ancestor ontologies. No actual versioning support is provided.

2.1.2 Adaptation

Searching. Ontologies are graphically displayed. They can only be browsed by using browsing commands, such as viewing a new ontology or inspecting its structure. No direct query interface is provided.
Editing. TaDzeBao is designed for discussing and editing ontologies (see Figure 2). It supports both asynchronous and synchronous discussions and editing on ontologies. The Tadzebao server is responsible for maintaining the ontologies, and delivering ontologies to requesting Tadzebao clients. The client is responsible for presenting a consistent view of the selected ontologies.
Reasoning. Ontologies in WebOnto are represented in OCML, which supports rule-based reasoning.

2.1.3 Standardization

Language. Ontologies (classes, instances, functions, procedures, or rules) are represented in OCML only [6]. In other words, no standard representation languages for ontologies are supported.
Upper-level ontologies. WebOnto does not include a 'giant' standard upper-level ontology but has a more fine-grained structure ([8] and [9]). At the top there is the base ontology describing the meta-model of OCML (things such as relations, functions, procedures, classes, instances, slots, etc., similar to SHOE). Below are the imported (with a few modifications) 'simple time' ontology from the Ontolingua library and ontologies describing

organizations, technologies, events and basic common concepts. Figure 3 shows the typical ontology inclusion hierarchy in the WebOnto ontology library system.

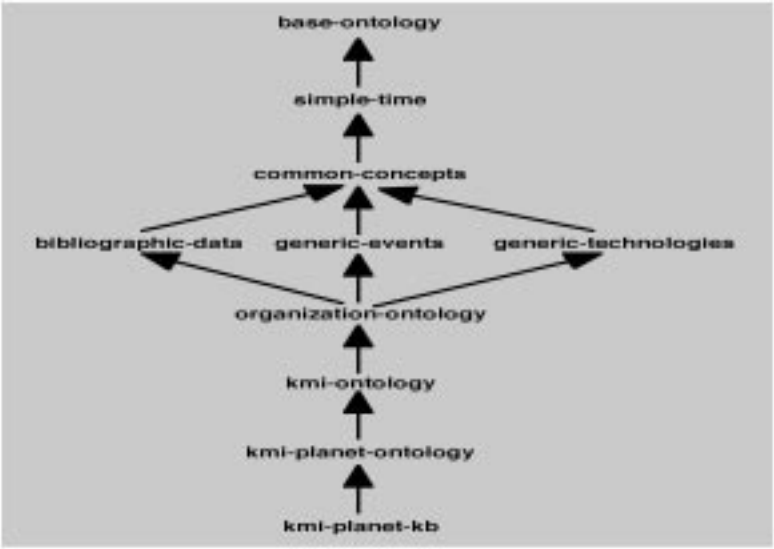


Figure 3. Structure of upper-level ontology in WebOnto

2.1.4 Summary

The WebOnto's ontology library system is client/server and graphically based. It stores an ontology as a module with a unique name for identification. It supports asynchronous and synchronous ontology editing. Ontology searching is limited to ontology navigating or browsing (but graphical-based). The ontology is represented by OCML, which can support rule-based reasoning. It does not have any ontology versioning function or strong support in respect to ontology standardization issues.

2.2 Ontolingua

Ontolingua was developed in the early nineties at the Knowledge Systems Laboratory of Stanford University (see Figure 4, [10]). It consists of a server and a representation language. The server provides a repository of ontologies (ontology library system) to assist users in generating new ontologies and amending the existing ontologies collaboratively. The ontology stored at the server can be converted into different formats [11].

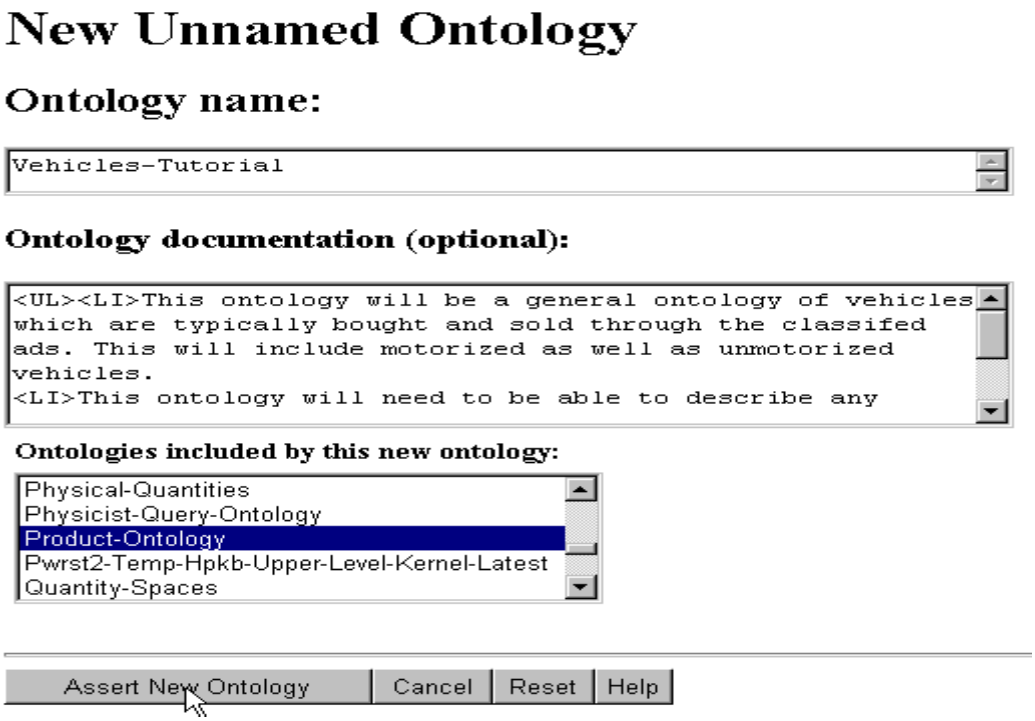


Figure 4. Screenshot of part of Ontolingua system (how to create an ontology)

2.2.1 Management

Storage. Ontolingua is client/server-based. It provides a distributed server *architecture* for ontology construction, use and re-use. Access to the contents of ontologies is provided via a network API and access to information derived from the contents by a general-purpose reasoner. The ontology server works like a database server and can enable distributed ontology repositories for editing, browsing, etc. The ontology server of Ontolingua supports a suite of other services, including configuration management for ontologies, support for ontologies that have components resident on remote servers, and support for an *Ontology-URL* that enables ontologies to be linked to the World Wide Web [12]. Ontologies stored in Ontolingua are not *classified* according to some existing categories. Ontology re-use in Ontolingua is supported by a *modular* structured library based on the following functions: inclusion, polymorphic, refinement, and restriction. Ontologies in this ontology library system are organized based on the lattice theory. Each ontology defines a set of formal terms. Ontologies can include (import from) other ontologies. Terms contained in an ontology are in the namespace of the ontologies that include it. In the lattice, an ontology includes the ontologies under which it is indented (see Figure 5). It applies the minimization and amortization principles enabling ontology writers to re-use existing ontologies in a flexible, powerful way [10].

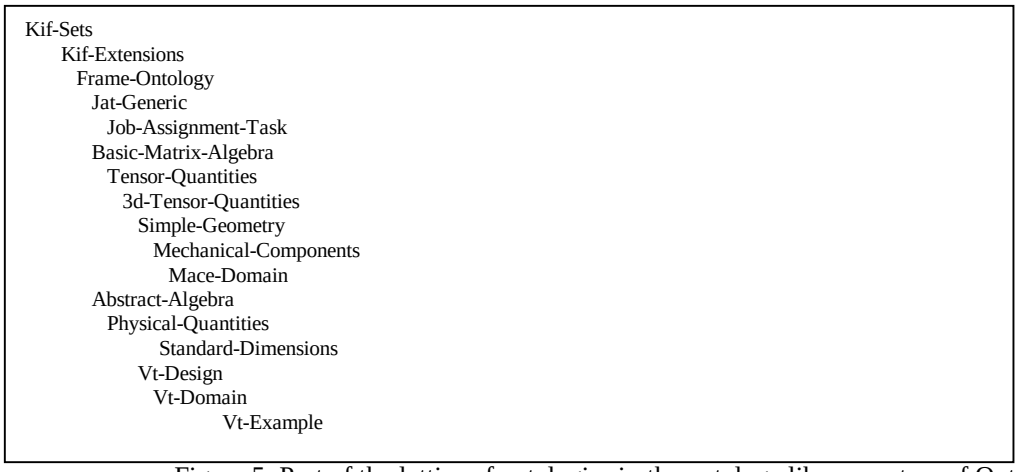


Figure 5. Part of the lattice of ontologies in the ontology library system of Ontolingua

The naming policy for Ontolingua is a good example of how the re-use of knowledge can be facilitated. The ontology in which a symbol is defined is called the symbol's home ontology. For example, if the symbol *S* is defined in ontology *A* as well as in ontology *B*, then from the perspective of ontology *A*, the input text "*S*" is interpreted as "the symbol named *S* defined in ontology *A*". From the perspective of ontology *B*, however, the input text "*S*" is interpreted as "the symbol named *S* defined in ontology *B*". The Ontolingua server input/output system includes a symbol-renaming feature that allows users to assign a name to a symbol, which is local to the perspective of a given ontology. This feature enables ontology developers to refer to symbols from other ontologies using names that are appropriate to a given ontology. It also enables them to determine how to resolve naming conflicts among symbols from multiple ontologies.

Identification. Each ontology has a name that uniquely distinguishes itself from any other ontology.

Versioning. The Ontolingua server does not feature any versioning functions.

2.2.2 Adaptation

Searching. Ontolingua features graphical ontology *browsing* and supports swift jumps from one term in the ontology to others term using hyperlinks. Ontolingua's class/subclass browsers can display an entire hierarchy in compact fashion, offering users a swift overview of an ontology. One particularly difficult task for ontology library systems is that of supporting efficient query answering from ontologies represented in a highly expressive language. Ontolingua develops an **idiom-based retrieval feature** that returns instances of a sentence containing diagrammatic variables from a given ontology. The retrieval feature employs a general purpose reasoner (i.e., theorem prover) and classifier that can be run as a background process to infer and cache sentences that match idioms used by the API and by translators. The general purpose reasoner developed for the Ontolingua representation language can provide basic reasoning support for ontology services including classification, deriving and catching instances of sentence diagrams to support idiom-based ontology access, ontology testing, and client-side execution [12]. Ontolingua also provides several tools that allow users to search for terms within ontologies in the library. A user may choose to use wild cards in searching the entire library for terms whose name matches the specified pattern. Context-sensitive searching is also available when the user needs to fill in the name of a term, by for instance, adding a value to a slot. Constraints are used to limit searches in context-sensitive searching. Finally, Ontilingua provides a *Reference ontology* that serves as an index of the ontology repository for class-based retrieval. Users can browse the reference ontology looking for classes of interest in the repository.

Editing. Ontolingua features four basic types of pages for the simple interface: the table of contents for the ontology library system, ontology summary pages, frame pages (for classes, relations or instances), and the class browser. Remote distributed groups can use their web browsers to build, and maintain ontologies stored at the server. However, this work cannot be carried out at the same time. Ontolingua supports vocabulary translation, which enables ontology builders to specify translation rules declaratively between the vocabulary used in a source ontology and the vocabulary used in a target ontology [12]. Ontolingua allows users to undo or redo any number of modifications made to the ontology since it was last saved.

Reasoning. Ontolingua enables developers to use an ontology to describe familiar situations and to query those situations to determine whether those situations have expected properties. It also includes a feature for specifying a test suite for an ontology, in which each test consists of a situation specification, a set of queries about the situation, and the answers expected to the queries.

2.2.3 Standardization

Language. The ontologies are stored primarily in KIF. KIF is a monotonic first order logic with a simple syntax and some minor extensions to support reasoning about relations. This language provides explicit support for building ontological modules that can be assembled, extended, and refined in a new ontology [1]. KIF is widely used among researchers in the United States .

Upper-level ontology. The public version of CYC upper-level ontology, called HPKB-UPPER-LEVEL with extended material drawn from Pangloss, WordNet, and Penma, is available on the Ontolingua server [13]. It contains approximately 3000 concepts, English definitions, and a few basic relationships between them. This upper-level ontology aims to maximize re-usability, enabling a greater degree of interoperation among knowledge-based systems by trying to account for all features associated with one event.

2.2.4 Summary

Ontolingua's ontology library system is client/server-based. It offers several options for re-using ontology: modular structure storage, lattice of ontology, naming policy, and a reference ontology (upper-level taxonomy). It supports collaborative ontology editing. Users can access the ontology library system via the Web. It also includes some relatively advanced searching features (wild-card and context intensive searching). In addition, Ontolingua supports ontological language translating, ontology testing and ontology integrating. HPKB-UPPER-LEVEL on the Ontolingua server maximizes re-usability and enables a greater degree of interoperation among knowledge-based systems.

2.3 DAML Ontology library system

The DAML ontology library system is part of the DARPA Agent Markup Language (DAML) Program, which officially started in August 2000. The goal of the DAML effort is to develop a language and tools to facilitate the concept of the Semantic Web. The ontology library system contains a catalogue of ontologies developed using DAML (Figure 6)[12]. This catalogue of DAML ontologies is available in XML, HTML, and DAML formats. People can submit new ontologies via the public DAML ontology library system.

```
<ontology uri="http://www.davincinetbook.com:8080/daml/rdf/personal-info.daml" id="2">
   <description>DAML ontology for homework 1</description>
   <poc name="Mark Neighbors" organization="Booz-Allen &amp;amp; Hamilton" email="neighbors_mark@bah.com"/>
   <submitter name="Mark Neighbors" organization="Booz-Allen &amp;amp; Hamilton" email="neighbors_mark@bah.com"
        date="2000-10-31"/>
   <keyword>personal information</keyword>
   <dmoz>http://dmoz.org/dmoz5</dmoz>
   <dmoz>http://dmoz.org/dmoz6</dmoz>
   <funder>DARPA DAML Program</funder>
   <class>AnnotatedBulletList</class>
     <class>Bullet</class>
   <class>BulletList</class>
   <class>Company</class> ……
…….
      <property>bulletList</property>
   <property>companies</property>
   <property>currentEmployerIDs</property>
   <property>currentProjectIDs</property>
   <property>description</property>
…….
<namespace>http://156.80.108.115/2000/10/daml-ont.daml</namespace>
   <namespace>http://www.w3.org/1999/02/22-rdf-syntax-ns</namespace>
   <namespace>http://www.w3.org/2000/01/rdf-schema</namespace>
  </ontology>
```

Figure 6. Example of an ontology in the DAML ontology library

[12] http://www.cs.man.ac.uk/~Ehorrocks/DAML-OIL/

2.3.1 Management

Storage. The DAML ontology library system is client/server-based. The structure of the stored ontology in this library system includes: ontology uri: ontology id; description; keyword; poc (point of contract): name, organization, email; submitter: name, organization, email; dmoz (open directory category); funder; classes (class names); properties (properties names); and namespaces (for example, Figure 6). Ontologies are *classified* according to Open Directory Category (www.dmoz.org), which includes arts, business, computers, games, health, home, kids and teens, news, recreation, reference, regional, science, shopping, society, sports, and world. This library also provides a summary of submitted ontologies, sorted by URI, Submission Date, Keyword, Open Directory Category, Class, Property, Funding Source, and Submitting Organization. The DAML ontology library system is still at its early age. It only provides a simple environment for people to submit and browse ontologies in the library. No *module* storage is considered at this moment.
Identification. Ontologies are identified by the URIs and identifiers.
Versioning. No versioning functions are provided.

2.3.2 Adaptation

Searching. This ontology library system offers no specific searching features. It contains only a catalogue of ontologies in three formats: XML, HTML and DAML. The HTML version can help users search by generated indexing of ontologies on URI, submission date, keyword, open directory category, class, property, namespace used, funding source, and submitting organization. The other two formats support simple browsing only.
Editing. No specific editing functions are available.
Reasoning. At present, this ontology library system does not support any reasoning functions[13].

2.3.3 Standardization

Language: The DAML ontology library is very preliminary at this moment. It aims to push and popularize the standard ontology language. It is, therefore, not surprising that the library system supports language standards for the semantic web, i.e. RDF, RDF Schema and DAML-ONT (now is DAML+OIL).
Upper-level ontology. At present, there is no upper-level ontology for the DAML ontology library system.

2.3.4 Summary

The DAML ontology library system is just in its beginning stages. Naturally, the objective is to offer various functions for ontology library system management. So far, however, we have not been able to find any publicly available literature with detailed information on the technology. Even at this early stage, this system is very strong in its support for web-based ontology languages.

2.4 SHOE (University of Maryland, USA)

SHOE (Simple HTML Ontology Extensions) was developed by the University of Maryland (USA) ([14], see Figure 7). SHOE is also the first web-semantics language developed as a markup, and has been used for various applications, including for food safety for the US Food and Drug Administration and a military logistics planning system.

2.4.1 Management

Storage. SHOE's ontology library system contains lists of ontologies. These ontologies are indexed alphabetically. They are also *classified* based on the ontology dependency with clear tree structure. The upper-level ontology (Base

[13] Developed by University of Manchester (UK), the FaCT (Fast Classification of Terminologies, http://www.cs.man.ac.uk/~horrocks/FaCT/) reasoner can be integrated into the DAML ontology library system and will give it some reasoning supports.

Ontology), for instance, forms the root of the tree. The generic ontologies (e.g. Dubline Core Ontology, General Ontology, Measurement Ontology) form the first branch of the tree. And the specific ontologies (e.g. Beer Ontology, Commerce Ontology, Personal Ontology) make up the leaves of the tree. Except for the upper-level ontology, each ontology is stored in the standard format, including ontology ID, version, description, contact, revision date, extended ontologies, renames, categories, relationships, constants, inferences, definitions, notes and change history.

Identification. The unique name becomes of the identifier of ontology.

Versioning. SHOE's versioning scheme is very essential in handling different types of revisions. It maintains each version of ontology as a separate web page and each instance must state the version to which it adheres. Data sources can, therefore, upgrade to the new ontology. To enter a revision in SHOE, the ontology designer copies the original ontology file, assigns it a new version number, then adds or removes elements accordingly. If the revision merely adds ontology elements, then it can be used to form perspectives that semantically subsume the original perspective. Therefore, it can specify that it is compatible with previous versions using the optional BACKWARD-COMPATBLE-WITH field in the <ONTOLOGY> tag. Agents and query systems that discover this ontology can also use it instead of any of the ontologies with which it is backwardly compatible to form an alternate perspective for any data source.

SHOE is currently the ONLY project focusing on the problem of maintaining consistency as the ontology evolves. It separates instances from ontologies so that ontologies can provide different perspectives on the same data. It identifies different types of ontology revisions that could significantly affect the reasoning with existing data sources. For instance, revisions that add categories or relations have no effect, revisions that modify rules can change the answers to queries, and revisions that remove categories or relations may eliminate certain answers [14].

Figure 7. Screenshot of SHOE ontology library system

2.4.2 Adaptation

SHOE features no *Searching* and *Editing* environment. Users have to edit their ontology somewhere else and submit it to SHOE. The alphabetical indexing of ontologies enables users to browse and search SHOE ontologies.
Reasoning. Reasoning supports are provided to handle revision problems. For instance, a revision that adds or removes rules can provide an alternative perspective (p') for a legacy data source. This makes it possible to reason the subsumption relations of the alternative perspective (p') with the original perspective (p) ([14] and [15]).

2.4.3 Standardization

Language: Ontology is written in SHOE. SHOE is an HTML-based knowledge representation language. It is a superset of HTML, which adds the tags to semantic data. There are two categories for SHOE tags: tags for constructing ontologies and tags for annotating web documents. There is also an XML-based version of SHOE [15].
Upper-level ontology. Base ontology is the upper-level ontology for SHOE. It becomes the parent ontology for all SHOE ontologies on the web. All other SHOE ontologies extend the base ontology directly or indirectly. Base ontology declares the global data types (string, number, date and truth), ISA hierarchy (entity, SHOEEntity), and relationships (description and name). There is a one-to-one correspondence between a version of SHOE and a version of the base ontology. Thus, the version of the Base ontology reflects the version of SHOE.

2.4.4 Summary

The SHOE ontology library system contains various ontologies (written in SHOE) with direct or indirect extensions of the upper-level ontology (Base ontology). SHOE flags itself with its versioning functions to solve inconsistencies caused by ontology evolution. SHOE itself is an extended HTML language with adding tags to represent ontologies and semantic data.

2.5 Ontology Server (Vrije Universiteit Brussels, Belgium)

Ontology Server, which was developed by the Vrije Universiteit in Brussels, links ontology engineering to database semantics. It deploys database techniques to manage and understand ontologies. The database management system (DBMS), equipped with various syntactical constructs, enables database diagrams to present objects, sub-type taxonomies, integrity constraints, derivation rules, etc. (see Figure 8).

2.5.1 Management

Storage. The ontology model consists of 5 basic elements: context, terms, concepts, roles and lexons. The ontology contains a set of contexts, which form the ontology itself. The ontology has a name (mandatory and unique in the ontology server), a contributor, an owner, a status ("under development", "finished") and documentation (an arbitrary string in which the contributor or owner can specify relevant information). The context is a grouping entity to group terms and lexons in the ontology. Every context within an ontology has its own unique name. A concept is an entity representing some "thing" and is identified by a unique ID. A term is an entity representing a lexical representation of a concept. Lexon is a grouping element with a triple structure containing a starting term, a role and a second term. Lexon always appears in a context and describes certain relations that are valid within that context. In this case, the lexons can be considered relations between concepts. The Database Management System is used to implement storage.
Identification. The unique name is the identifier for each ontology in this ontology server.
Versioning. The first prototype currently under construction does not take account of the version control. However, this will become a crucial issue in the next step.

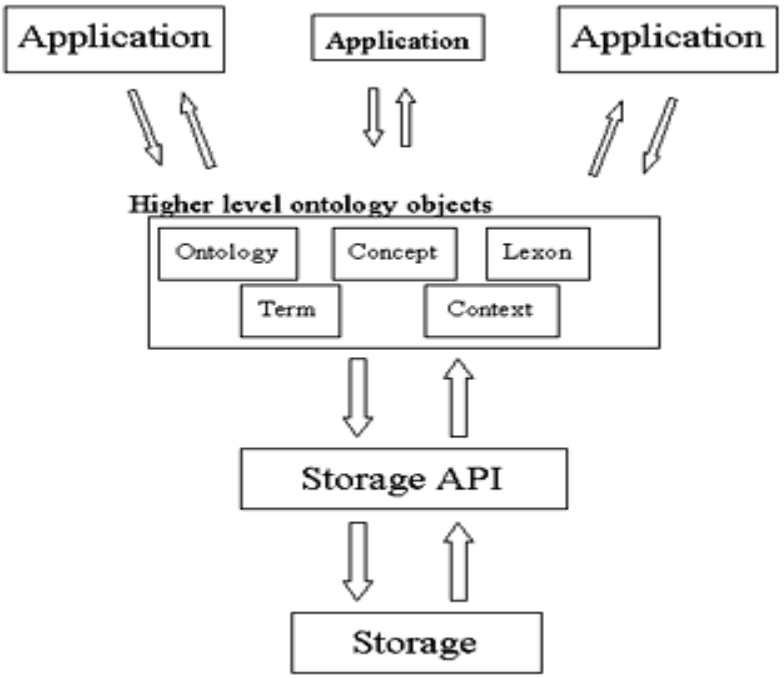


Figure 8. Ontology server architecture

2.5.2 Adaptation

Searching & **Editing.** Database API provides unified access to the basic structures of the ontology server. As the data in the ontology server, ontologies are managed by the DBMS (Database Management System). The API itself is specified as four different java interfaces. One of these is an interface to establish and close the connection to the database. The second interface features all of the basic functions required to add information to the ontology server (specific methods for adding ontology, context, terms, concept, lexons, users and versions are included). The third, the retrieval interface, features all of the basic functions to retrieve information from the ontology server. (The specific methods for this can be divided in two groups: (a) retrieving methods for detailed information about ontologies, contexts, terms, concepts, lexons, users and versions; and (b) retrieving methods for grouped information, such as those that retrieve all ontologies from the ontology server, all contexts from an ontology, all terms from a context, all lexons from a context and all users from the ontology server). And finally, the modification interface features all of the basic functions needed to modify information already present in the ontology server (it includes specific methods for modifying ontologies, contexts, terms, concepts, lexons, users and versions). In the future, an ontology manager will be developed to provide support for storing ontologies expressed in XML. WordNet will be added to the ontology server using the ontology manager. The ontology browser (currently under development) will assist users in accessing and browsing ontologies, contexts, terms, concepts, lexons, users and versions.

Reasoning. According to the documents available, no reasoning functions are specified.

2.5.3 Standardization

Language. The ontology object is expressed in XML.
Upper-level ontology. No upper-level ontology is adopted in this Ontology Server.

2.5.4 Summary

Ontology Server manages ontologies by using DBMS (Database Management System). It separates semantics from ontologies; thus, each ontology model contains 5 basic elements (context, terms, concepts, roles and lexons).

Ontology can be accessed and searched through database API, including via the modification interface, the retrieval interface, and in the future, ontology manager, and ontology browser.

2.6 Others

This section discusses systems that are either not very standard in ontology library systems or are still in the very preliminary stages of development.

2.6.1 IEEE Standard Upper Ontology (IEEE)

The IEEE Standard Upper Ontology (SUO) Working Group has invested tremendous effort, working with a large number of ontologists, to create a standard top-level ontology to enable various applications, such as data interoperability, information search and retrieval, automated inferencing, and natural language processing. Their ontology library system is very simple and is accessible in its preliminary form on their website. It contains a group of classified ontologies, such as, ontologies in SUO-KIF, formal ontologies and linguistic ontologies/lexicons. Only the very basic hyperlinks of the ontologies are provided to help users jump to the home pages hosted by the ontologies (see Figure 9). There are no clear management, adaptation and standardization functions, such as those discussed in Section 1.

One special effort worth mentioning here is the first-ever merger of certain SUO sources into a single and coherent ontology, an ontology accessible via the website. This merger was achieved by combining David Whitten's structural ontology, John Sowa's upper ontology, Allen's temporal logic, Russell and Norvig's upper ontology, Casati and Varzi's formal theory of holes, Barry Smith's formal theory of boundaries, Borgo, Guarino, and Masolo's formal theory of physical objects, the Core Plan Representation, and the Agents and Numbers ontologies from the Ontolingua server.

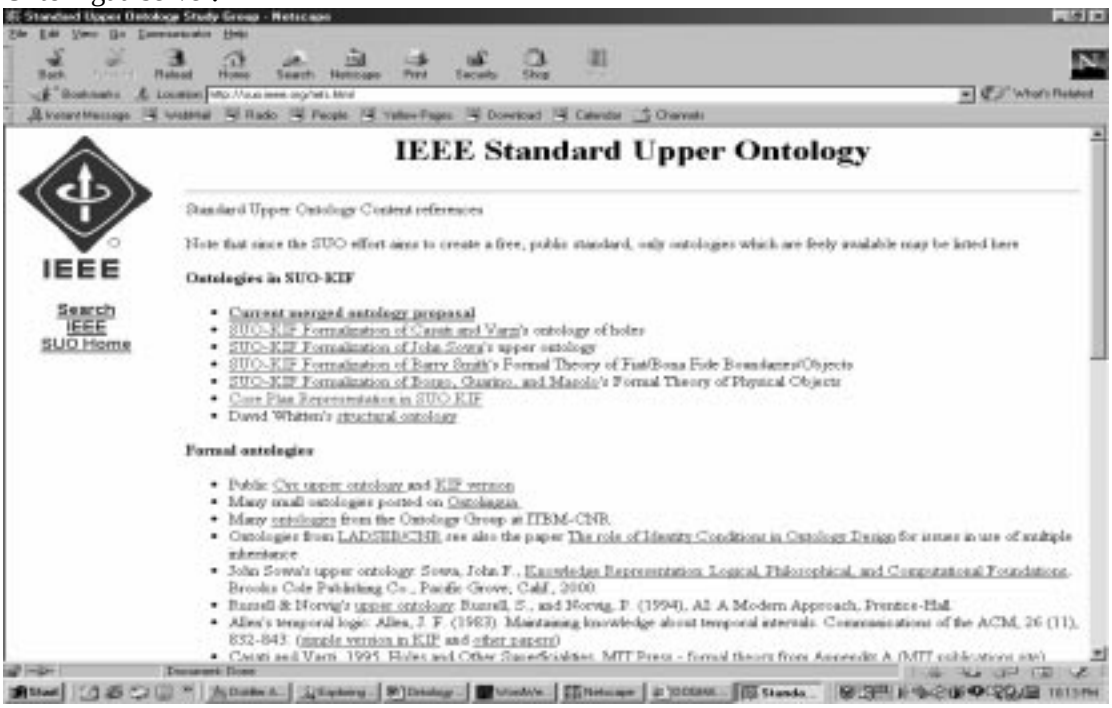


Figure 9. Screenshot of IEEE SUO

2.6.2 OntoServer (AIFB)

OntoServer, which is currently under construction, is an ontology server to support building, maintaining and using ontologies. It has a client/server-based architecture, which integrates various types of software or tools to form tool-based support for an ontology environment (see Figure 10).

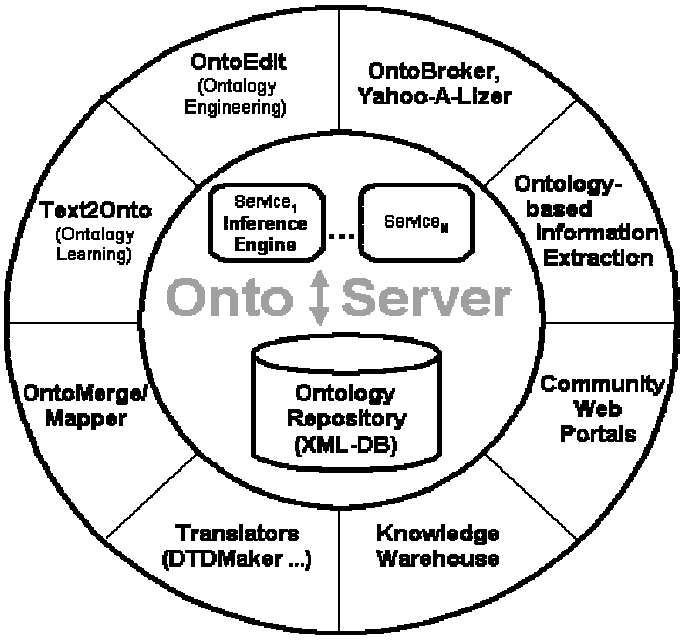


Figure 10. OntoServer infrastructure

OntoServer will integrate tools from ontology engineering (such as OntoEdit, Text-To-Onto, Ontology Merger/Mapper, Ontology Visualization and Translator) and ontology applications (such as OntoBroker, OntoWrapper, Ontology Visualization (Hyperbolic View), DTD-Maker, Intranet Management System and Semantic Community Web Portals). OntoServer is still in its very early stages; no detailed information is available from its homepage. Consequently, we can only sketch the infrastructure.

2.6.3 ONIONS

ONIONS *(ONtological Integration Of Naive Sources)* is a methodology for ontology integration (ontology mediation, alignment and unification). It was developed in the early 1990s to account for the problem of conceptual heterogeneity. ONIONS creates a *stratified* design of an ontology library system. It contains richly documented and formalized generic ontologies and a cognitively transparent top level. Moreover, intermediate *modules* contain the most general concepts of a domain, based on the generic ontologies and the top level. Domain ontologies are designed based on intermediate ontology ([16], see Figure 10).

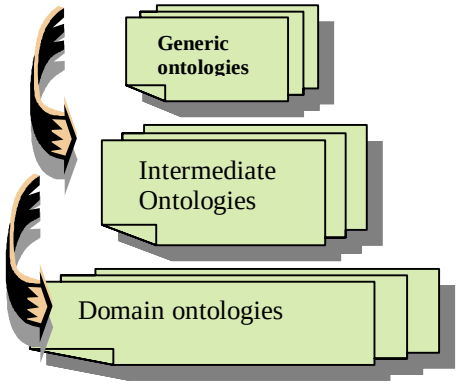


Figure 10. The stratified design of an ontology library system (ONIONS)

ONIONS is committed to developing a large-scale ontology library system for medical terminology. This methodology employs a design based on logic description for the modules in the library and makes extended use of generic theories, thus creating a *stratification* of the modules. The current implementation of the methodology employs LOOM, a knowledge representation system that supports classification services based on the description logic. Ontologies are *classified* based on description logic. The ontology library system covers all local definitions and the paradigms used in building multi-local, integrated definitions. Further classification in this library is based on steps pertaining to the diffusion, use, classification, and validation of the models.

ONIONS is mainly an ontology integration methodology, which is implemented by many projects. It creates a *stratified* ontology library system including generic ontology, intermediate ontology and domain ontology.

3. Summary of the Survey

Research on ontology library systems is still a very new field. The following summary is based on our survey of the ontology library systems described above (see also Table 1).

3.1 Management

Storage. The ontology library systems in this survey fall into one of two categories: (a) those with a client/server-based *architecture* aimed at enabling remote accessing and collaborative editing (WebOnto, Ontolingua, DAML Ontology Library); and (b) those that feature web-accessible architecture (SHOE, IEEE SUO). Ontology Server features a database-structured architecture. Most ontologies in this survey of ontology library systems are classified or indexed. They are stored in a modular structured library (or lattice of ontologies). WebOnto, Ontolingua and ONIONS all highlight the importance of a modular structure in an ontology library system as that structure facilitates the task of reorganizing ontology library systems and re-using and managing ontologies.

Identification. The standard way to identify an ontology is by its Unique name or Identifier.

Versioning. Only SHOE supports versioning for handling the dynamic changes of ontologies. Versioning is an important aspect of the ontology library system. Although many of the systems surveyed do not currently have this function, they clearly show that it is needed for future improvements.

*3.2 **Adaptation***

Searching. Most of these ontology library systems can be accessed through the Internet or World Wide Web. They offer simple browsing only. Ontolingua is the only one that offers some functional searching features, such as keyword searching (wide-card searching), simple query answering, context sensitive searching, etc. As it is embedded in the database management system, Ontology Server could also provide SQL-based searching.

Editing. Most ontology library systems only provide simple editing functions. WebOnto and Ontolingua support collaborative ontology editing (asynchronous and synchronous).

Reasoning. Very simple reasoning functions are provided by WebOnto (rule-based reasoning), Ontolingua (ontology testing) and SHOE (ontology revision).

Table 1. The summary of the ontology library system survey

		WebOnto	Ontolingua	DAML library	SHOE	Ontology Server	Others (IEEE SUO, OntoServer, ONIONS)
Management	**Storage**	- client/server-based - no classification - modularity storage	- client/server-based - no classification - modular structured library (lattice of ontologies, naming policy)	- client/server-based - classification of ontology - no modularity storage	- web accessible - classification of ontology - tree structure of ontology dependency	- database access - no classification - modularity storage	- web access (IEEE SUO), client/server-based (OntoServer), - classification of ontology (IEEE SUO, ONIONS) - stratified design (ONIONS)
	Identification	- unique name - unique unit name	- unique name	- unique URI and Identifier	- unique name	- unique name	
	Versioning	- indirect: inherited from ancestor ontology	No versioning	No versioning	- versioning support for ontology revision	- no versioning	- no versioning
Adaptation	**Searching**	- graphical display - simple browsing	- simple browsing - idiom-based retrieval facility for simple query answering - wild-card searching - context sensitive searching - reference ontology as the index	- simple browsing	- simple browsing	- database API - DBMS - add, modify, retrieve - ontology manager - ontology browser	- simple browsing (IEEE SUO,
	Editing	TaDzeBao: - asynchronous and synchronous discussions and editing on ontologies,	- simple interfaces - collaborative ontology construction - vocabulary translation - undo/redo - hyperlinked environment	No specific editing functions	- no editing	- add, modify, retrieve	- no editing
	Reasoning	- rule-based reasoning	- use situation to determine the expected properties. - ontology testing	- no reasoning	- limited reasoning support for ontology revision	- no reasoning	- no reasoning
Standardization	**Language**	OCML	KIF - ontology language translation	RDF, RDFs, DAML+OIL	SHOE	XML	
	Upper-level Ontology	- no standard upper-level ontology - a more fine-grained structure: based ontology, simple-time, common concepts	- public version of CYC upper-level ontology (HPKB-UPPER-LEVEL)	No standard upper-level ontology	- Base Ontology	- no standard upper-level ontology	- IEEE SUO (upper-level ontology integration)

3.3 ***Standardization***

Language. These ontology library systems use different languages to store their ontologies. In this case, the important function for the future ontology library system should support inter-language translating (like Ontolingua) or some standard language should be accepted or proposed within the ontology community (such as DAML+OIL).
Upper-level Ontology. Ontolingua has a public version of CYC upper-level ontology called HPKB-UPPER-LEVEL with some modification drawn from Pangloss, WordNet, and Penma. WebOnto and SHOE doesn't have the standard upper-level ontology but has its own fine-grained structure (e.g., Base Ontology). IEEE SUO tries to set up a public standard upper-level ontology.

4. Conclusions: Ontology library system requirement

Now that we have surveyed the ontology library systems above, we will summarize important requirements for structuring an ontology library system to enhance ontology management, adaptation and standardization:

4.1 ***Management***

Storage. A client/server-based *architecture* is critical to an ontology library system's capacity to support collaborative ontology editing. An ontology library system should also be web accessible.
It is necessary to *classify* ontology in an ontology library system in order to facilitate searching, managing and re-using ontology. Some of the ontology classification mechanisms available are based on distinguishable features of ontologies. Examples include the following:

- the *subject* of ontologies (The DAML ontology library system classifies ontologies according to the Open Directory Category (www.dmoz.org));
- the *structure* of the ontology (The Ontolingua ontology library system has an inclusion lattice showing the inclusion relations between different ontologies);
- inter and intra ontology *features* ([17] indexed ontologies based on the intra and inter ontology features. Examples include *general, design process, taxonomy, axioms, inference mechanism, application, contributions, etc.*);
- the *lattice* structure ([18] built a lattice of ontologies showing the relevance of ontologies);
- the *dimensions* of the ontology ([19] indexed ontologies using dimensions (task/method dependency and domain dependency) to partition the library into a core library and a peripheral library);
- *stratified upper-level* ontology (ONIONS used generic, intermediate and domain layer to index ontologies),
- the *relations* of ontology ([17] indexed ontology based on defined relations, such as the subset/superset relation, extension relation, restriction, and mapping relation),
- the *components* of ontology ([17] also mentioned the indexing of ontology based on the component of ontologies, such as domain partitioning (partition domain in logical units), alternative domain views (polymorphic refinement), abstraction (abstract and detailed ontologies), primary ontologies versus secondary ontologies, terminological, information and knowledge modeling ontologies).

Modular organization in the ontology library system organizes units into modules. This serves to maximize cohesion within modules and minimize interaction between modules [20]. Most of the ontology library systems that aim to facilitate ontology re-use, ontology mapping and integration have adopted this structure. ONIONS also highlights the *stratified* design of an ontology library system. Different *naming policies* assist the ontology library system to achieve the modular organization or stratified storage of ontologies [21]. The disjointed partitioning of classes can facilitate modularity, assembling, integrating and consisting checking of ontologies. If, for instance, a certain class, such as 'people,' were disjointed from another class, say 'countries', then consistency checks could be carried out much sooner and faster. Thus, the partition modification has proven to be extremely valuable for editing purposes. Linking class names with their own contexts or using name space for differentiating them can serve to prevent violation within individual ontologies. As ontologies continue to grow, so too does the importance of systematic and consistent naming and organizational rules.

Identification. Unique ontology URL, Identifier and name are used as the identifier for ontologies in the ontology library systems.

Versioning. A version control mechanism is very important to an ontology library system. Unfortunately, most existing ontology library systems cannot support it, except for SHOE.

4.2 ***Adaptation***

Searching & Editing. An ontology library system should feature a visualized browsing environment, using hyperlinks or cross-references to closely related information. It should support collaborative editing and offer advanced searching features by adopting various existing information retrieval techniques, database searching features, or AI heuristic techniques. Ontology library systems could also monitor user profiles based on access patterns in order to personalize the view of ontologies [22].

Reasoning. A simple reasoning function should be included in order to facilitate ontology creation, ontology mapping and integration.

4.3 ***Standardization***

Language. Syntactically, an ontology representation language should be standardized or inter- or intra- ontology language translation should be supported. Semantically, an ontology library system should feature the *common vocabulary* (or faceted taxonomy). At any rate, it should eliminate the implicitness and misunderstanding of terms in different ontologies (due to synonyms, homonyms, etc.) for most generic classes. Preferably, an ontology library system should also support compatibility with or mapping between multiple controlled vocabularies from different domains. This would not only serve to guarantee flexibility in expressing an ontology semantically, but also to liquidate implicitness. The structures of these common vocabularies or multiple controlled vocabularies must be faceted, or modulated so as to facilitate the re-use, mapping and integration of ontologies [23]. These vocabularies can help in simple synonym matching, sibling analysis, and disjoint partition checking.

Upper-level Ontology. Standard upper-level ontology is important for better organization of ontology library systems (Ontolingua, IEEE SUO).

*4.4 **Others***

Ontology scalability. Ontology library systems should also consider increasing the scale of ontologies.

Maintaining facility. Ontology library systems should also provide some maintenance features, such as consistency checking, diagnostic testing, support for changes, and adaptation of ontologies for different applications.

Explicit documentation. Each ontology in an ontology library system should be extensively documented. The documentation should include such information as how the ontology was constructed, how to make extensions and what the ontology's naming policy, organizational principles and functions are. Such explicit documents about the ontologies themselves will pave the way for efficient ontology management and re-use.

Acknowledgement:

This research effort was supported by OnToKnowledge, a project funded by the European Union (www.ontoknowledge.org). We would like to thank Enrico Motta, Asun Gomez-Perez, Alexander Maedche, York Sure for providing useful information. We would also like to thank Michel Klein and Borys Omeliango for their helpful comments on drafts of this paper.

References

[1] D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, 2001.

[2] O. Corcho, & A. Gomez-Perez, A road map on ontology specification languages. In *Workshop on Applications of Ontologies and Problem solving methods, 14th European Conference on Artificial Intelligence (ECAI'00)*, Berlin, Germany, August 20-25, 2000.

[3] D.Jones, T. Bench-Capon & P. Visser, Methodology for ontology development. In *Proceedings of IT&KNOWS Conference of the 15th IFIP World Computer Congress* (Chapman-Hall), pp, 20-35. 1998.

[4] A. Maedche, & S. Staab, Ontology Learning for the Semantic Web. To appear in: IEEE Intelligent Systems. 16(2), March/April 2001.

[5] P.R.S. Visser, R.W. van Kralingen, & T.J.M. Bench-Capon, A method for the development of legal knowledge systems. *Proceedings of the Sixth International Conference on Artificial Intelligence and Law (ICAIL'97)*, Melbourne, Australia. 1997.

[6] J. Domingue, Tadzebao and WebOnto: Discussing, Browsing, and Editing on the Web. In B. Gaines and M. Musen (editors), *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Worksho*p, April 18th-23th, Banff, Canada, 1998. Available online at http://kmi.open.ac.uk/people/domingue/banff98-paper/domingue.html

[7] E. Motta, *Reusable Components for Knowledge Models*. PhD Thesis. Knowledge Media Institute. The Open University, UK, 1998.

[8] E Motta., S.Buckingham-Shum and J. Domingue, Ontology-Driven Document Enrichment: Principles, Tools and Applications. *International Journal of Human-Computer Studies* **52**, (2000) 1071-1109.

[9] J. B. Domingue and E. Motta, Planet-Onto: From News Publishing to Integrated Knowledge Management Support. *IEEE Intelligent Systems* **15**, (2000) 26-32.

[10] A.Farquhar, R. Fikes, & J. Rice, The Ontolingua server: Tools for collaborative ontology construction. *International Journal of Human Computer Studies* **46**, (1997) 707-728.

[11] A.J.Duineveld, R. Stoter, M.R. Weiden, B.Kenepa, V.R. Benjamins, WonderTools? A comparative study of ontological engineering tools. In *Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada, October, 1999.

[12] R. Fikes,& A. Farquhar, Large-scale repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems* **14**, (1999).

[13] S.Aitken, Extending the HPKB-Upper-Level Ontology experiences and observations. In *Proceedings of the Workshop on Applications of Ontologies and Problem Solving Methods(ECAI'98)*, Brighton, England, August 1998.

[14] J. Heflin & J. Hendler, Dynmaic ontologies on the Web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).* AAAI/MIT Press, Menlo Park, CA, 2000. pp. 443-449.

[15] J. Heflin, J. Hendler, and S. Luke, *SHOE: A Knowledge Representation Language for Internet Applications*. Technical Report CS-TR-4078 (UMIACS TR-99-71), Dept. of Computer Science, University of Maryland at College Park. 1999.

[16] D.M. Pisanelli, A. Gangemi, & G. Steve, An Ontological Analysis of the UMLS Metathesaurus, *Journal of American Medical Informatics Association* **5** (1998) 810-814.
[17] P.R.S. Visser, and T.J.M. Bench-Capon, A Comparison of Four Ontologies for the Design of Legal Knowledge Systems. *Artificial Intelligence and Law* **6** (1998) 27-57.
[18] F. N. Noy, & C.D. Hafner, The state of the art in ontology design: A survey and comparative review. *AI Magazine* **4** (1997) 53-74.
[19] G. van Heijst, A.T. Schreiber, and B. J. Wielinga, Using explicit ontologies in KBS development. *Int. Journal of Human-Computer Studies* **45** (1997) 183-292.
[20] G. van Heijst, et al. A Case Study in Ontology Library Construction. *Artificial Intelligence in Medicine* **7** (1995) 227-255.
[21] D.L McGuinness,. R. Fikes, J. Rice, & S. Wilder, An environment for merging and testing large ontologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000).* Breckenridge, Colorado, April 12-15, 2000.
[22] J. Domingue, & E. Motta, A knowledge-based news server supporting ontology-driven story enrichment and knowledge retrieval. In D. Fensel and R. Studer (editors), *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW '99),* LNAI 1621, Springer-Verlag, 1999.
[23] D.L. McGuinness, Conceptual modelling for distributed ontology environment. In *Proceedings of the Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS2000), Darmstadt, Germany, August 14-18, 2000.*

UML and the Semantic Web

Stephen Cranefield
Department of Information Science, University of Otago
PO Box 56, Dunedin, New Zealand
E-mail: scranefield@infoscience.otago.ac.nz

Abstract. This paper discusses technology to support the use of UML for representing ontologies and domain knowledge in the Semantic Web. Two mappings have been defined and implemented using XSLT to produce Java classes and an RDF schema from an ontology represented as a UML class diagram and encoded using XMI. A Java application can encode domain knowledge as an object diagram realised as a network of instances of the generated classes. Support is provided for marshalling and unmarshalling this object-oriented knowledge to and from an RDF/XML serialisation. The paper also proposes an extension to RDF allowing the identification of property–resource pairs in a model for which 'closed world' reasoning cannot be used due to incomplete knowledge.

1 Introduction

The vision of the Semantic Web is to let computer software relieve us of much of the burden of locating resources on the Web that are relevant to our needs and extracting, integrating and indexing the information contained within. To enable this, resources on the Web need to be encoded in, or annotated with, structured machine-readable descriptions of their contents that are expressed using terms or structures that have been explicitly defined in a domain *ontology*.

Currently, there is a lot of research effort underway to develop ontology representation languages compatible with World Wide Web standards, particularly in the Ontology Inference Layer (OIL [1]) and DARPA Agent Markup Language (DAML [2]) projects. Derived from frame-based representation languages from the artificial intelligence knowledge representation community, OIL and DAML schema build on top of RDF Schema by adding modelling constructs from description logic [3]. This style of language has a well understood semantic basis but lacks both a wide user community outside AI research laboratories and a standard graphical presentation—an important consideration for aiding the human comprehension of ontologies.

This paper discusses Semantic Web technology based on an alternative paradigm that also supports the modelling of concepts in a domain (an ontology) and the expression of information in terms of those concepts. This is the paradigm of object-oriented modelling from the software engineering community. In particular, there is an expressive and standardised modelling language, the Unified Modeling Language (UML [4]), which has graphical and XML-based formats, a huge user community, a high level of commercial tool support and an associated highly expressive constraint language. Although developed to support analysis

and design in software engineering, UML is beginning to be used for other modelling problems, one notable example being its adoption by the Meta Data Coalition[1] [5] for representing metadata schemas for enterprise data.

The proposed application of UML to the Semantic Web is based on the following three claims:

- UML class diagrams provide a static modelling capability that is well suited for representing ontologies [6].
- UML object diagrams can be interpreted as declarative representations of knowledge [7].
- If a Semantic Web application is being constructing using object-oriented technology, it may be advantageous to use the same paradigm for modelling ontologies and knowledge.

However, there is one significant current shortcoming of UML: it lacks a formal definition. The semantics of UML are defined by a metamodel, some additional constraints expressed in a semi-formal language (the Object Constraint Language, OCL), and descriptions of the various elements of the language in English. The development of formal semantics for UML is an active area of research as evidenced by a number of recent workshops [8, 9, 10] and the formation of an open-membership international research group to facilitate work in this area [11]. In particular, as UML is a very large language with some redundancy, research is underway to identify a core of UML modelling constructs from which the other language elements can be derived [12, 13]. A formal definition of this core will then indirectly provide semantics for the complete language.

The present author believes that future developments in this area will provide at least a subset of UML with the formal underpinnings required for the unambiguous interpretation of ontologies. For the present, the use of the more basic features of class and object diagrams for representing ontologies and knowledge seems no more prone to misinterpretation than the use of the Resource Description Framework—a language which underlies the Semantic Web but also lacks official formal semantics (although some have been proposed [14]). It is also worth noting that many of the difficulties in providing precise semantics for UML lie with its features for dynamic modelling of systems, rather than the static models used in the present work.

Further discussion of the benefits and limitations of UML for ontology modelling is beyond the scope of this paper which focuses on technology to support the use of object-oriented modelling for the Semantic Web. An overview of this technology is given in Section 2. Section 3 presents an example ontology in UML and gives a brief description of the features of UML used in this example (for a good introduction to a much larger subset of the language see Muller [15]). Section 4 describes the techniques used to generate an RDF schema and a set of Java classes (complete with RDF-based object marshalling support) from a UML ontology. Section 5 proposes an extension to RDF that allows the inclusion in an RDF model of information on how complete that model is for particular property–resource pairs. Some preliminary thoughts on the possibility of performing reasoning with ontologies and knowledge expressed in UML are presented in Section 6. Finally, Section 7 gives an overview of related work.

[1] the MDC has since merged with the Object Management Group to work jointly on the OMG's Common Warehouse Metamodel for data warehousing, business intelligence, knowledge management and portal technology metadata.

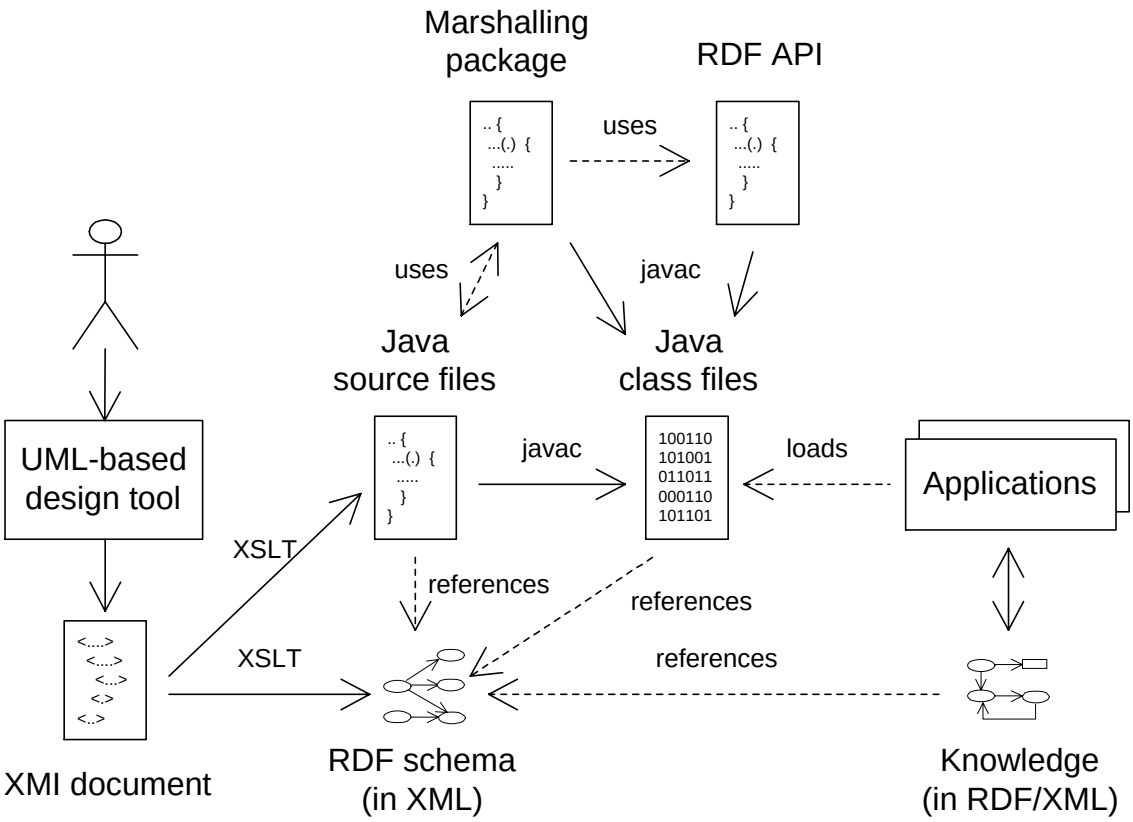


Figure 1: Overview of the implemented technology for object-oriented knowledge representation

2 Required technology for UML and the Semantic Web

To enable the use of UML representations of ontologies and knowledge, standard formats are needed to allow both ontologies and knowledge about domain objects to be published on the Web and transmitted between agents. In addition, there is a need for code libraries to help application writers parse and internalise this serialised information.

This technology already exists for UML class diagrams. The XML Model Interchange language (XMI) defines a standard way to serialise UML models. There are also a number of Java class libraries existing or under development to provide a convenient interface for applications to access this information. However, there is currently no similar technology available to help Java applications construct, serialise and read object-oriented encodings of knowledge that are conceptualised as UML object diagrams. XMI documents can encode the structure of object diagrams, but this is necessarily done in a domain-independent way using separate but cross-referenced XML elements for each object, link, link end and attribute–value binding. What is required is a way to generate from an ontology a domain-specific encoding format for knowledge about objects in that domain, and an application programmer interface (API) to allow convenient creation, import and export of that knowledge.

Figure 1 shows an approach to object-oriented ontological and object-level knowledge representation that has been implemented and is described in this paper. First, a domain expert designs an ontology graphically using a CASE tool supporting the Unified Modeling Language, and then saves it using the standard XML-based format XMI (XML Model Interchange). A pair of XSLT (Extensible Stylesheet Language Transformations) stylesheets then

take the XMI representation of the ontology as input and produce (i) a set of Java classes and interfaces corresponding to those in the ontology, and (ii) a representation of the ontology using RDF (Resource Description Framework) using the modelling concepts defined in RDF Schema. The generated Java classes allow an application to represent knowledge about objects in the domain as in-memory data structures. The generated schema in RDF defines domain-specific concepts that an application can reference when serialising this knowledge using RDF (in its XML encoding). The marshalling and unmarshalling of object networks to and from RDF/XML documents is performed by a pair of Java classes: `MarshalHelper` and `UnmarshalHelper`. These delegate to the generated Java classes decisions about the names and types of fields to be serialised and unserialised, but are then called back to perform the translation to and from RDF, making use of an existing Java RDF application programmer's interface [16].

Note that the generated RDF schema does not contain all the information from the original UML model. If an application needs access to full ontological information, it can use the original XMI document with the help of one of the currently available or forthcoming Java APIs supporting the processing of UML models. The purpose of the RDF schema is to define RDF resources corresponding to all the classes, interfaces, attributes and associations in the ontology in order to support RDF serialisation of instance information. For the sake of human readers, the schema records additional information such as subclass relationships and the domains and ranges of properties corresponding to attributes and associations. However, this information is not required for processing RDF-encoded instance information because each generated Java class contains specialised methods `marshalFields` and `unmarshalFields` containing hard-coded knowledge about the names and types of the class's fields. This is a design decision intended to avoid potentially expensive analysis of the schema during marshalling and unmarshalling. This it should be possible to use this serialisation mechanism in situations where optimised serialisation is important, such as in agent messaging systems.

3 An example domain

This section presents an example ontology modelled as a UML class diagram and some knowledge encoded as an object diagram. The ontology defines a subset of the concepts included in the CIA World Factbook and is adapted from an OIL representation of a similar subset [17].

3.1 An ontology in UML

Figure 2 presents the CIA World Factbook ontology represented as a UML class diagram. The version shown here is not a direct translation from OIL: there is an additional class `AdministrativeDivision`, UML association classes are used where appropriate, and instead of defining the classes `City` and `Country` as specialised types of `Region` (GeographicalLocation in the OIL original), the ontology represents these as optional roles that a region may have.

The boxes in the diagram represent classes, and contain their names and (where applicable) their attributes. The lines between classes depict association relationships between

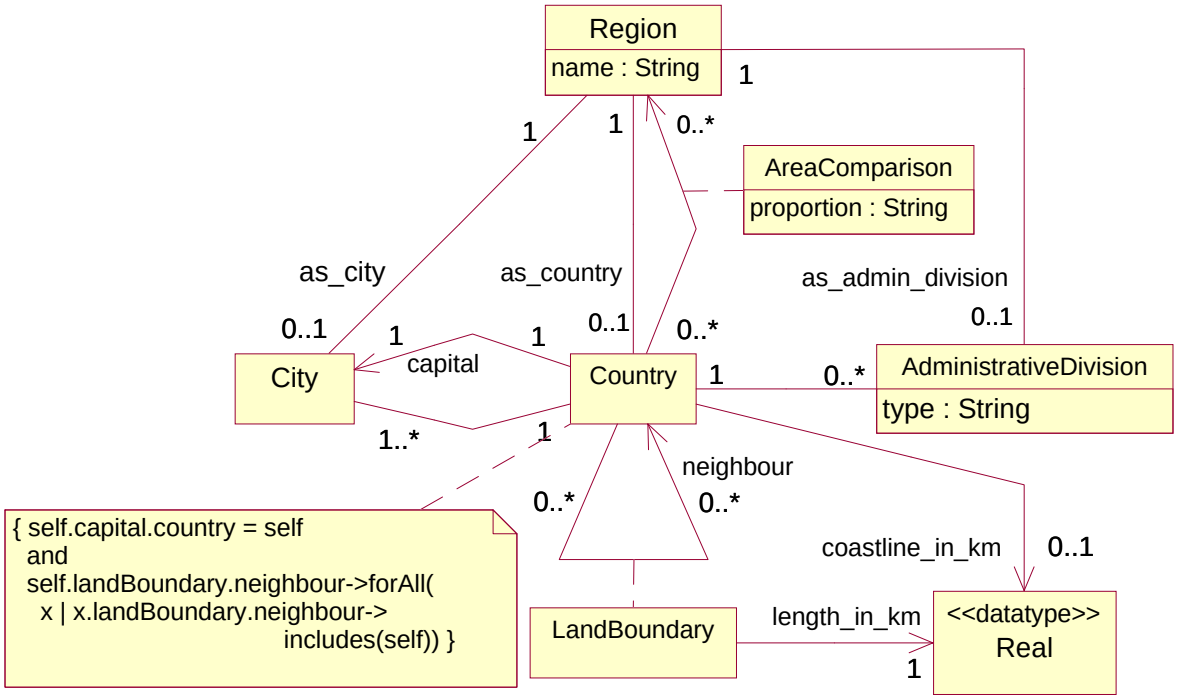


Figure 2: A UML ontology for a subset of the CIA World Factbook

classes. A class `A` has an association with another class `B` if an object of class `A` needs to maintain a reference to an object of class `B`. An association may be bidirectional, or (if a single arrowhead is present) unidirectional. A 'multiplicity' expression at the end of an association specifies how many objects of that class may take part in that relationship with a single object of the class at the other end of the association. This may be expressed as a range of values, with '`*`' indicating no upper limit. Association ends may be optionally named. In the absence of a name, the name of the adjacent class, with the initial letter in lower case, is used as a default name. Associations can be explicitly represented as classes by attaching a class box to an association (see `LandBoundary` and `AreaComparison` in the figure). This is necessary when additional attributes or further associations are required to clarify the relationship between two classes.

The dog-eared rectangle in the lower left corner of the figure contains a constraint in the Object Constraint Language (OCL). This language provides a way to constrain the possible instances of a model in ways that cannot be expressed using UML's structural modelling elements alone. The constraint shown here states that i) a country's capital is a city in that country, and ii) if a country c has another as a neighbour, then that neighbouring country has c as a neighbour. Finally, the keyword "datatype" appearing in guillemets above the class `Real` indicates that this is a pre-existing built-in datatype. OCL defines a minimal set of primitive datatypes and it is currently assumed that the ontology designer has used these primitive types.

Note that UML includes notation for class generalisation/specialisation relationships, although this is not required for the example presented in this paper.

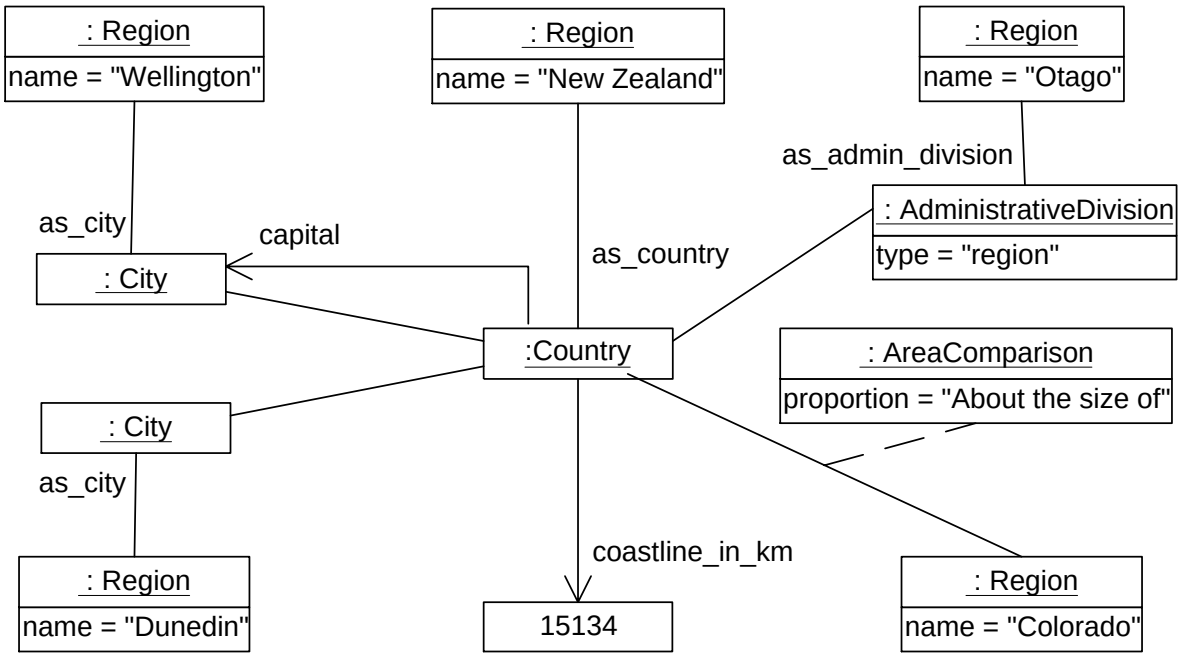


Figure 3: Information about New Zealand as a UML object diagram

3.2 Knowledge as an object diagram

Figure 3 presents some knowledge about New Zealand from the CIA World Factbook, expressed as an object diagram. For brevity, this diagram omits most of New Zealand's cities and administrative divisions, and provides no information about the region named Colorado, to which New Zealand is compared in terms of area.

In object diagrams, rectangles denote objects, specifying their class (after an optional name and a colon) and the object's attribute values. The lines between objects show 'links': instances of associations between classes.

4 From UML to RDF and Java

The previous section presented some knowledge expressed as a UML object diagram. This is an abstract representation of that knowledge. To enable this style of knowledge representation to be used for Semantic Web applications it is necessary to define an API to allow creation of the knowledge in this form and a serialisation format to allow Web publication and transmission of the knowledge. These can be generated automatically from an XML encoding of the Word Factbook using the XSLT stylesheets that have been developed. One stylesheet produces an RDF schema corresponding to the ontology and the other produces a corresponding set of Java classes and interfaces.

XSLT is a language for transforming XML documents into other documents. An XSLT stylesheet is comprised of a set of templates that match nodes in the input document (represented internally as a tree) and transform them (possibly via the application of other templates) to produce an output tree. The output tree can then be output as text or as an HTML or XML document.

The main issue common to both mappings is the problem of translating from UML

classes, which may have different types of features such as attributes, associations and association classes, to a model where classes only have fields or (in RDF) properties. It was also necessary to generate default names for fields where association ends are not named in the UML model. The OCL conventions for writing navigation paths through object structures were used to resolve these issues. Also, attributes and association ends with a multiplicity upper limit greater than one are represented as set-valued fields (bags in RDF Schema) or, in the case of association ends with a UML "ordered" constraint, list-valued fields (sequences in RDF Schema). Further details about the mappings have been discussed elsewhere [18] and are beyond the scope of this paper.

4.1 The generated RDF schema

The Resource Description Framework (RDF) is a simple resource–property–value model designed for expressing metadata about resources on the Web. RDF has a graphical syntax as well as an XML-based serialisation syntax. For readability, examples in this paper are presented in the graphical syntax, although in practice they are generated in the XML format.

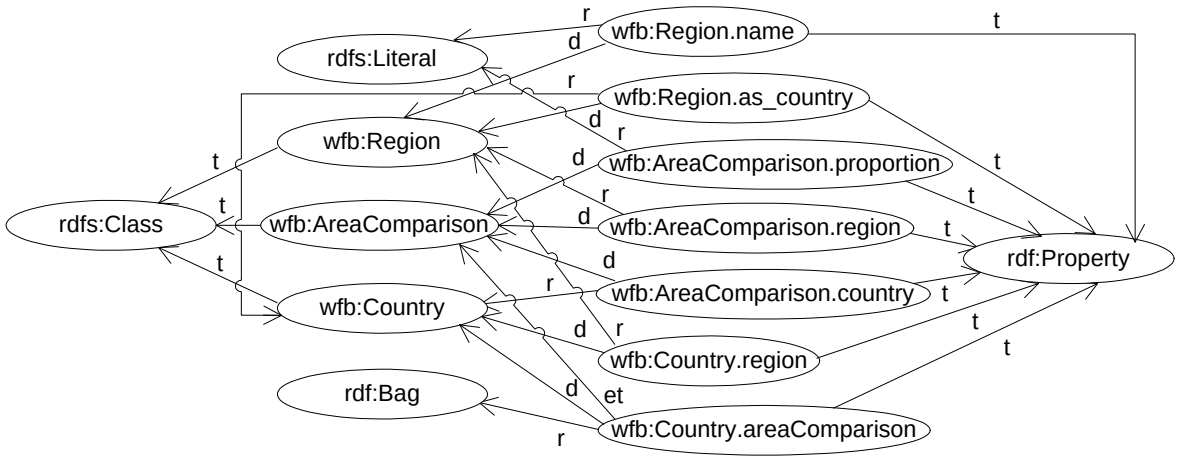
RDF Schema is a set of predefined resources (entities with uniform resource identifiers) and relationships between them that define a simple meta-model including concepts of classes, properties, subclass and subproperty relationships, a primitive type `Literal`, bag and sequence types, and domain and range constraints on properties. Domain schemas (i.e. ontologies) can then be expressed as sets of RDF triples using the (meta)classes and properties defined in RDF Schema. Schemas defined using RDF Schema are commonly called RDF schemas (small 's').

The main issue in generating an RDF schema that corresponds to an object-oriented model is that RDF properties are first-class objects and are not defined within the context of a particular class. This can lead to conflicting range declarations if the same property (e.g. `head`) is used to represent a field in two different classes (e.g. `Brew` and `Department`). The solution chosen was to prefix each property name representing a field with the name of the class. This has the disadvantage that in the presence of inheritance a class's fields may be represented by properties with different prefixes: some specifying the class itself and some naming a parent class. This might be confusing for a human reader but is not a problem for the current purpose: to specify a machine-readable format for object-oriented knowledge interchange.

Figure 4 presents a subset of the generated RDF schema corresponding to the UML model presented in Figure 2. Only the classes `Country` and `Region` and the relationships between them are included here.

In the standard RDF graphical notation used in the figure, an ellipse represents a resource with its *qualified name* shown inside as a namespace prefix followed by a local name. A namespace prefix abbreviates a Uniform Resource Identifier (URI) associated with a particular namespace, and the URI for the resource can be constructed by appending the local name to the namespace URI. A property is represented by an arc, with the qualified name for the property written beside the arc (in this case the arcs are given labels with the corresponding URIs shown in the table).

Figure 4 includes one property that is not part of RDF Schema. There is no mechanism in RDF Schema to parameterise a collection type (such as `rdf:Bag`) by the class of elements it may contain. Therefore, the non-standard property `rdfsx:collectionElementType` was introduced to represent this information (this is abbreviated in the figure by the arc label

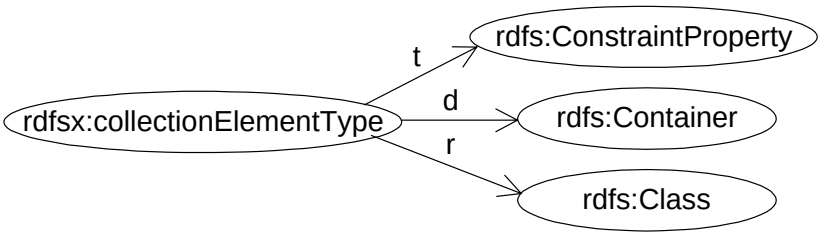


Property labels	Name space abbreviations
t = rdf:type	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
d = rdfs:domain	rdfs = http://www.w3.org/2000/01/rdf-schema#
r = rdfs:range	rdfsx = http://nzdis.otago.ac.nz/0_1/rdf-schema-x#
et = rdfsx:collectionElementType	wfb = *any new namespace chosen for this schema*

Figure 4: Part of the World Factbook schema in RDF

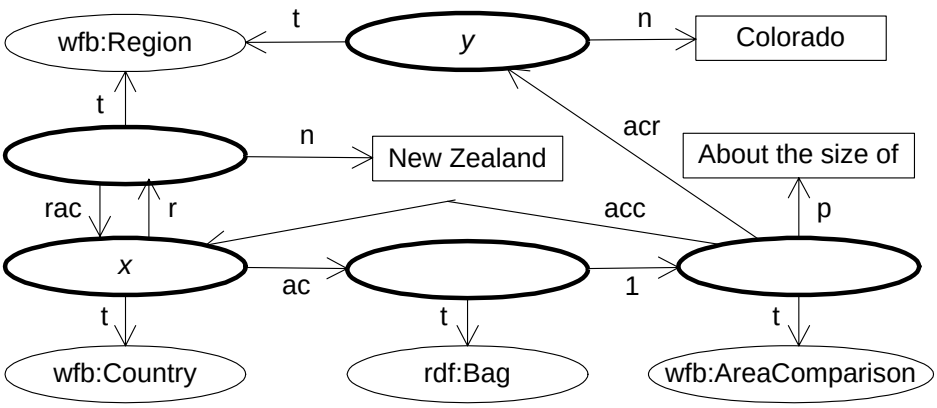
`et`). The definition of this property is shown in Figure 5. The object serialisation mechanism described in this paper does not require this information but it is useful to people reading the schema.

The schema in Figure 4 completely defines the encoding of instance data in RDF. Figure 6 shows how an object diagram is encoded in RDF with reference to the schema. This corresponds to the central `Region` and `Country` objects from Figure 3 together with the `AreaComparison` link to the Colorado `Region` object. The five resources outlined in bold are the ones being defined. There are two resources of type `wfb:Region`, one of type `wfb:Country`, one of type `rdf:Bag` (representing the set of the country's area comparisons) and one element in the bag, an instance of the area comparison association class (which is represented in RDF as the type `wfb:AreaComparison`). Depending on the needs of the



Property labels	Name space abbreviations
t = rdf:type	rdfs = http://www.w3.org/2000/01/rdf-schema#
d = rdfs:domain	rdfsx = http://nzdis.otago.ac.nz/0_1/rdf-schema-x#
r = rdfs:range	

Figure 5: An extension to RDF Schema



Property labels	Name space abbreviations
t = rdf:type	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
1 = rdf:_1	wfb = *namespace chosen for the World Factbook schema*
n = wfb:Region.name	
rac = wfb:Region.as_country	
r = wfb:Country.region	
ac = wfb:Country.areaComparison	
p = wfb:AreaComparison.proportion	
acr = wfb:AreaComparison.region	
acc = wfb:AreaComparison.country	

Figure 6: Information about New Zealand encoded in RDF

application, defined objects might be assigned URIs or represented as anonymous resources. In this case, they are anonymous—the labels x and y are included to allow reference to these resources later in this paper. The rectangles represent RDF literals.

Note that while this graphical notation for RDF looks complicated, its encoding in XML only requires five XML elements to represent the information, as shown in the appendix.

4.2 The generated Java classes and marshalling framework

The generated RDF schema described in the previous section defines a domain-specific serialisation format for object-oriented representation of knowledge about the domain. To facilitate the processing of knowledge communicated in this form, a set of Java classes can also be generated from the ontology using XSLT. These allow Java applications to instantiate instances of the domain concepts. In addition, the generated classes, along with some additional utility classes, allow these in-memory structures to be marshalled and unmarshalled to and from the RDF serialisation format defined by the generated RDF schema. The aim of the marshalling code is to allow a Java application to maintain an internal representation of object-oriented knowledge and to easily read and write parts of this knowledge to and from a format suitable for transmission or publication on the Web.

Figure 7 presents a class diagram outlining the structure of the generated Java classes and the marshalling framework. The class `MarshalHelper` is part of a support package used by the generated classes. A static method `marshalObjects` provides the entry point for an application to marshal a network of objects. A similar class `UnmarshalHelper` is

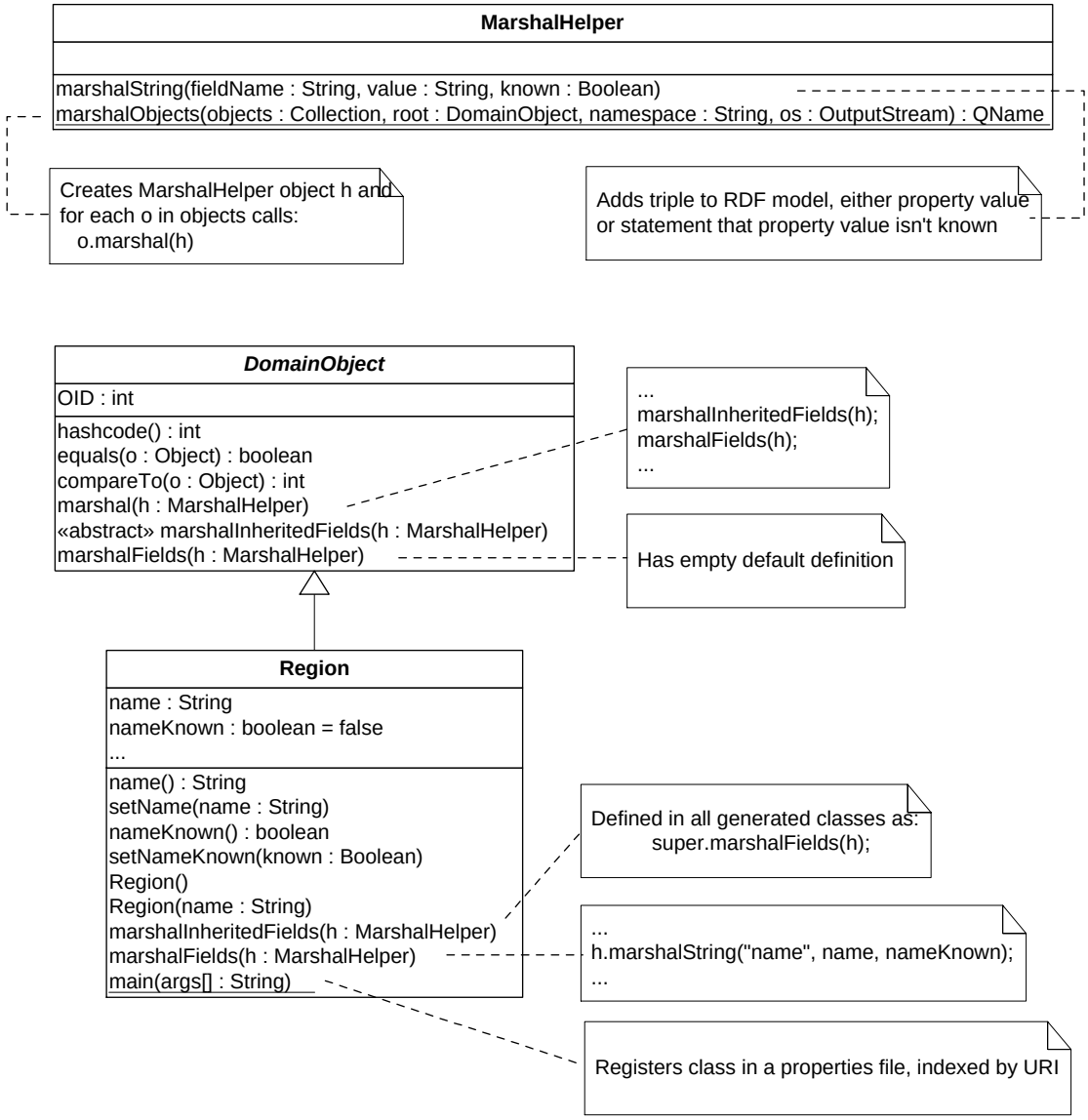


Figure 7: The structure of the generated classes and the marshalling methods

also provided, but is not discussed here. The class `DomainObject` is an abstract base class that all generated classes specialise (the specialisation relationship is represented by a closed arrow pointing to the more general class). The class `Region` is shown as an example of a generated class.

This diagram does not show all the fields and methods. In particular, the class `Region` also contains fields and methods related to the association ends `as_city`, `as_country` and `as_admin_division` from the ontology shown in Figure 2. There are some fields and methods depicted that are related to whether or not a field value is “known”. This is discussed in Section 5.

There is a significant difference between knowledge represented propositionally and knowledge represented in the form of an object diagram. Propositions are self-contained statements of knowledge whereas object diagrams are networks of objects. When serialising knowledge, an application may only wish to include some of the information it knows about a domain.

```
// Build object diagram
Region rNZ = new Region("New Zealand");
Country cNZ = new Country();
AreaComparison ac = new AreaComparison();
ac.setCountry(cNZ);
ac.setRegion(rNZ);
ac.setProportion("About the size of");
Set comparisons = new HashSet();
comparisons.add(ac);
Region rColorado = new Region("Colorado");
rNZ.setAs_country(cNZ);
cNZ.setRegion(rNZ);
cNZ.setAreaComparisonSet(comparisons);
// Now marshal it
Set toMarshal = new HashSet();
toMarshal.add(rNZ); toMarshal.add(cNZ);
toMarshal.add(ac); toMarshal.add(rColorado);
try {
  QName rootQName =
    MarshalHelper.marshalObjects(
      toMarshal, rNZ, "http://nzdis.otago.ac.nz/nzdata1#",
      new FileOutputStream("nz.xml"));
}
catch (MarshallingException e) { ... }
```

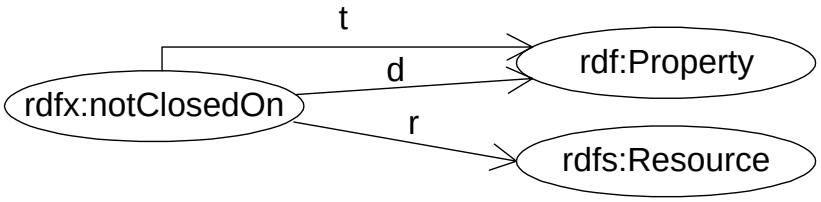
Figure 8: Sample Java code to create and marshal an object diagram

For example, Figure 6 compares New Zealand's area to that of Colorado, but doesn't provide the information that Colorado is an administrative division of the United States. To allow this selectivity, the `marshalObjects` method takes a collection of objects as an argument. Links to any objects outside this collection will not be serialised. To allow a particular entry point into the knowledge structure to be identified, a root object is specified and the method returns the qualified name of the RDF resource in the serialised model that represents that object. A namespace for the serialised information is also provided.

Figure 8 shows the Java code that would produce the RDF serialisation in Figure 6.

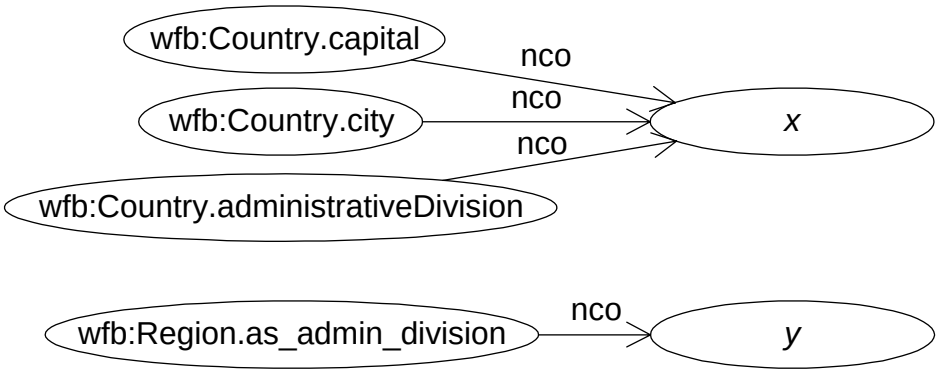
5 Modelling incomplete knowledge

Because object diagrams are inter-linked networks of objects rather than sets of discrete facts, and because classes may have attributes or associations that are optional (i.e. have a multiplicity lower bound of zero), it is important to be able to distinguish between a statement that there are no values for a given property and the omission or lack or knowledge about a given property. In other words, the recipient of object-oriented information needs a way of knowing for which objects and which properties a closed world assumption can safely be made. This is achieved by including extra boolean fields in the generated Java classes that record for each regular field whether or not its value is 'known' or, for set- or list-valued fields, 'closed'—meaning that the contents of the set or list provide complete knowledge of that field. Setting the value of a single-valued field sets its 'known' field to true and all fields also have a method



Property labels	Name space abbreviations
t = rdf:type	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
d = rdfs:domain	rdfs = http://www.w3.org/2000/01/rdf-schema#
r = rdfs:range	rdfx = http://nzdis.otago.ac.nz/0_1/rdf-x#

Figure 9: Schema for the notClosedOn property



Property labels	Name space abbreviations
nco = rdfx:notClosedOn	rdfx = http://nzdis.otago.ac.nz/0_1/rdf-x#
	wfb = *namespace chosen for the World Factbook schema*

Figure 10: Meta-knowledge about incomplete information

allowing the programmer to explicitly specify the status of the field.

When unmarshalling an object diagram from the RDF encoding it is assumed that complete information about all properties is included unless otherwise specified (although the opposite could equally well be implemented as the default assumption). Incomplete information is indicated using a non-standard RDF property `notClosedFor` that associates a property with a resource, meaning that complete information is not provided for that property applied to that resource. Figure 9 shows the declaration of the `notClosedOn` property.

Figure 10 presents an example of this property applied to the encoding of knowledge about New Zealand that was shown in Figure 6. When combined with the RDF-encoded information in Figure 6, this specifies that the RDF model does not contain complete (or possibly any) information about the capital, cities and administrative divisions of the country represented by the resource labelled x. Also, there is possibly missing information about the administrative division property of the region represented by the resource labelled y.

In order to have this meta-level information added to the RDF serialisation, the Java code shown in Figure 8 must have the following lines added before the call to `marshalObjects`:

```
cNZ.setCapitalKnown(false);
cNZ.setCitySetClosed(false);
cNZ.setAdministrativeDivisionSetKnown(false);
rNZ.setAs_admin_divisionKnown(false);
```

Similar mechanisms for handling incomplete knowledge have been used in knowledge representation systems LOOM [19] (which includes `:closed-world` and `:open-world` relation properties) and CLASSIC [20] (which allows roles to be declared to be 'closed'). This notion has also been formalised in description logic by the use of epistemic operators that modify roles, and in AI planning by the use of "local closed world" formulae [21]. The `notClosedOn` property used in this work should also include a reference to the current RDF model, but this is not currently possible as RDF does not provide a way to declare that a set of statements collectively constitute a model with a given URI.

6 Reasoning with OCL

The Semantic Web, as envisioned by Tim Berners-Lee [22], includes a logical layer which allows "the deduction of one type of document from a document of another type; the checking of a document against a set of rules of self-consistency; and the resolution of a query by conversion from terms unknown into terms known". One of the biggest challenges for the Semantic Web community is to find interoperable ways of incorporating inference rules into ontologies. There is much research to be done in this area. For example, although the XML DTD for OIL 1.0 defines a `rule-base` element, its content is unconstrained text and no semantic connection is made between this rule base and the rest of the language. The RDF schemas defining later versions of OIL and DAML do not currently contain any way of representing rules, although it is a goal of the DAML project to produce an enhanced language, DAML-L, with support for rules.

It is therefore an important question to evaluate how well UML fares in this regard. In fact, UML includes a powerful mechanism for expressing inference rules: the Object Constraint Language. OCL has been claimed to be "essentially a variant of [first order predicate logic] tuned for writing constraints on object structures" [23]. This claim is true from a syntactic viewpoint: OCL is sufficiently expressive to represent any first-order inference rules that an ontology designer may wish to specify (although this expressiveness also means that reasoning about unconstrained OCL expressions is likely to be undecidable in general). From a semantic viewpoint, the above claim cannot be verified as OCL currently lacks a formal specification. However, this shortcoming has been recognised in the UML 2.0 OCL RFP [24] and at least one proposal for formal semantics for OCL has already been made [25].

The object-oriented syntax of OCL is also unlike any commonly used logical language, and attempting to write rules in OCL can be frustrating for the inexperienced. A constraint can often be expressed in several different ways and the resulting expression can look quite unlike its counterpart in first-order logic. Consider the constraint in Figure 2. The second conjunct specifies that the neighbourhood relationship between countries is reflexive. The form of this constraint might be immediately recognised as a standard pattern by an OCL expert but it is not obvious to the uninitiated.

To enable tractable reasoning about ontologies in UML, and to avoid the awkward syntax of OCL, it would be useful to define a macro language on top of OCL comprising predicates such as `reflexive(`*`path-expression`*`)` which are defined in terms of OCL. The set

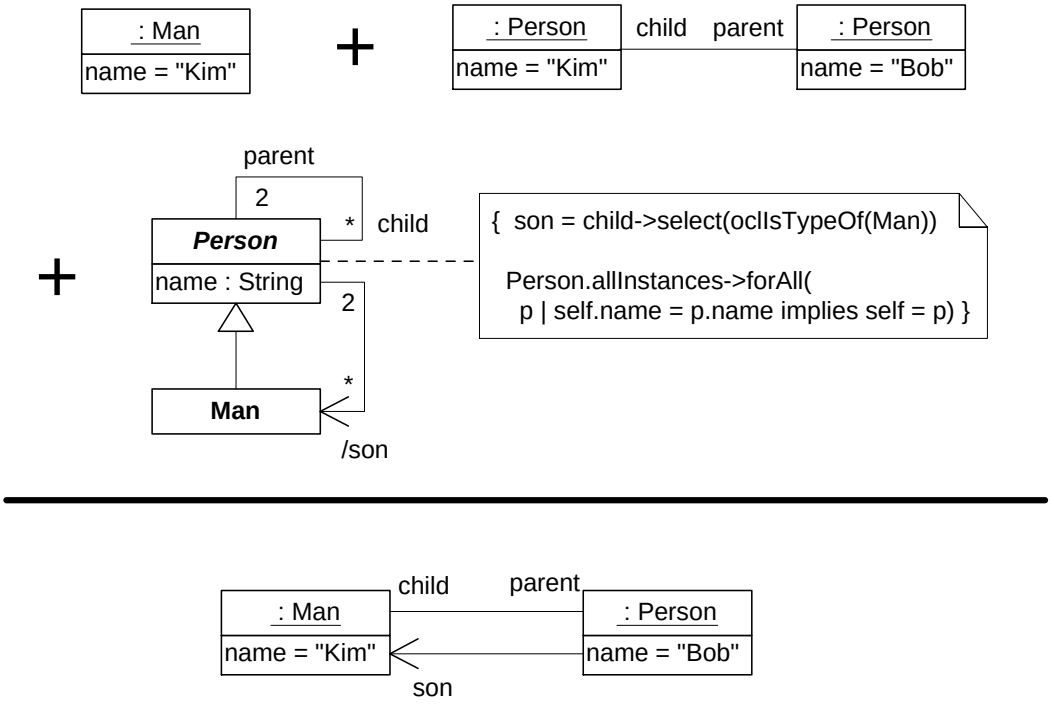


Figure 11: An example of inference over knowledge in UML

of macros could be chosen to ensure that reasoning over these expressions is tractable. This would also help to allow the translation of rules between UML-based and other representations of an ontology. This is a subject for future research.

Recent research has also shown how inference rules in UML can be expressed as graph transformations on the UML metamodel [26, 13]. To give a taste of what inference with UML might look like, Figure 11 shows how new knowledge in the form of an object diagram can be generated by combining existing knowledge and information about the ontology. In this example, one agent has communicated to another that there is an object of class `Man` with "Kim" as the value of its `name` attribute. The other agent knows that there is a `Person` object with name Kim and that this object is the child of a `Person` object with name Bob. The ontology for this domain states that `Man` is a specialisation of `Person`, and includes two OCL constraints: one defining the derived role (indicated by '/') `son` (a son is a child that is a man), and the other stating (rather unrealistically) that the `name` attribute uniquely identifies objects of class `Person`. Over several steps of inference the agent can conclude that the two objects with name Kim are the same and therefore Kim is a male child, i.e. a son. Implementing this style of deduction in UML is a subject for future research.

7 Related Work

A number of other projects have investigated the serialisation of instances of ontological models.

Skogan [27] defined a mechanism for generating an XML document type definition (DTD) from a UML class diagram and implemented this as a script for the UML-based modelling

tool Rational Rose. This is being used for the interchange of geographical information. The mapping is only defined on a subset of UML and many useful features of class diagrams, including generalisation relationships, are not supported.

Work has also been done on producing DTDs [28] and XML schemas [17] from models expressed in ontology modelling languages (Frame Logic and OIL respectively). The latter work reported that the XML Schema notion of type inheritance does not correspond well to inheritance in object-oriented models, which was a factor in the choice of RDF as a serialisation format in the research described here.

Since its initial proposal, OIL has been redesigned as an extension of RDFS [29]. This means that an ontology in OIL is also an RDF schema and therefore knowledge about resources in a domain modelled by an OIL ontology can easily be expressed using RDF.

The UML-based Ontology Toolset (UBOT) project at Lockheed Martin is working on tools to map between UML and DAML representations of ontologies [30]. This project has a different focus from the work described in this paper. Rather than using the existing features of UML to describe ontologies, the language is being extended with a set of UML 'stereotypes' (specialisations of UML modelling constructs) that correspond to classes and properties in the RDF schema for DAML. A proposal has also been made for an extension to the UML metamodel that would allow global properties in DAML ontologies to be modelled as aggregations of UML association ends (which are local to classes) [31].

The Web site http://www.interdataworking.com provides a number of 'gateways' that can be used to convert between different data formats. One particular 'gateway stack' can be used to produce an RDF schema from an XMI document, although no information is given about the mapping and how much of UML is supported. The resulting schema is defined using a mixture of properties and (meta)classes from RDF Schema (such as `rdfs:subClassOf`) and from Sergey Melnik's RDF representation of the UML metamodel [32]. The schema defines properties and classes that can be referenced when encoding object information in RDF, and could itself be used as an alternative to an XMI encoding for publishing and serialising an ontology modelled using UML. However, as XMI is an Object Management Group standard for model interchange, it is being supported by an increasing number of tools and APIs and there seem to be few advantages in using a different format for encoding UML models. If it is required to annotate an ontology with additional information that is not part of the XMI format (one of Melnik's desiderata) this could be achieved using external annotations and XLink [33].

Xpetal [34] is a tool that converts models in the 'petal' output format of the UML-based modelling tool Rational Rose to an RDF representation. No details are provided about the mapping from UML to RDFS and which UML features are supported.

8 Conclusion

This paper has described technology that facilitates the application of object-oriented modelling, and the Unified Modeling Language in particular, to the Semantic Web. From an ontology specified in UML, a corresponding RDF schema and a set of Java classes can be automatically generated to facilitate the use of object diagrams as internal knowledge representation structures and the import and export of these as RDF documents. A mechanism was also introduced for indicating when an object diagram has missing or incomplete knowledge.

Important areas for future work are the identification of tractable subsets of OCL for

encoding inference rules and the definition of mappings between object-oriented representations of ontologies and knowledge and more traditional description logic-based formalisms. This would allow applications to choose the style of modelling most suitable for their needs while retaining interoperability with other subsets of the Semantic Web.

The software described in this paper is publicly available at http://nzdis.otago.ac.nz/projects under the name "uml-data-binding".

Acknowledgements

This work was done while visiting the Network Computing Group at the Institute for Information Technology, National Research Council of Canada, Ottawa, Canada. Thanks are due to Larry Korba and the NRC for hosting me and to the University of Otago for approving my research and study leave.

References

[1] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng and O. Corby, editors, *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer, 2000.

[2] DAML project home page. http://www.daml.org, 2000.

[3] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications, 1996.

[4] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[5] Meta Data Coalition home page. http://www.mdcinfo.com/, 2000.

[6] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-23/cranefield-ijcai99-iii.pdf.

[7] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, *Proceedings of the 2nd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2) at the 5th European Meeting on Cybernetics and Systems Research (EMCSR 2000)*, pages 573–578, Vienna, 2000. Austrian Society for Cybernetic Studies. Published under the title "Cybernetics and Systems 2000". An earlier version is available at http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2000-07.pdf.gz.

[8] Ana Moreira, L. F. Andrade, A. R. Deshpande, and S. Kent. Formalizing UML. Why? How? In *Addendum to the Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'98)*. ACM/SIGPLAN, 1998.

[9] S. Kent, A. Evans, and B. Rumpe. UML semantics FAQ. In A. Moreira and S. Demeyer, editors, *Object-Oriented Technology: ECOOP'99 Workshop Reader*, volume 1743 of *Lecture Notes in Computer Science*. Springer, 1999. http://www4.informatik.tu-muenchen.de/papers/KER99.html.

[10] J.-M. Bruel, J. Lilius, A. Moreira, and R.B. France. Defining precise semantics for UML. In J. Malenfant, S. Moisan, and A. Moreira, editors, *Object-Oriented Technology: ECOOP 2000 Workshop Reader*, volume 1964 of *Lecture Notes in Computer Science*. Springer, 2000.

[11] pUML. Precise UML Group Web site. http://www.puml.org, 2001.

[12] A. S. Evans and S. Kent. Meta-modelling semantics of UML: the pUML approach. In B. Rumpe and R. B. France, editors, *Proceedings of the 2nd International Conference on the Unified Modeling Language*, volume 1723 of *Lecture Notes in Computer Science*. Springer, 1999. http://www.cs.york.ac.uk/puml/papers/pumluml99.pdf.

[13] M. Gogolla. Graph transformations on the UML metamodel. In *Proceedings of the ICALP Workshop on Graph Transformations and Visual Modeling Techniques (GVMT'2000)*, pages 359–371. Carleton Scientific, 2000. ftp://ftp.informatik.uni-bremen.de/%2Flocal/db/papers/Gogolla_2000_GraGra.ps.

[14] W. Conen and R. Klapsing. A logical interpretation of rdf. *Linköping Electronic Articles in Computer and Information Science*, 5(13), 2000. http://www.ep.liu.se/ea/cis/2000/013/.

[15] P.-A. Muller. *Instant UML*. Wrox Press, 1997.

[16] S. Melnik. RDF API homepage. http://www-db-stanford.edu/~melnik/rdf/api.html, 2000.

[17] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and schema-languages: translating OIL-specifications in XML-Schema. In *Proceedings of the Workshop on Applications of Ontologies and Problem solving Methods, 14th European Conference on Artificial Intelligence (ECAI 2000)*, 2000. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/7.pdf.

[18] S. Cranefield. Networked knowledge representation and exchange using UML and RDF. *Journal of Digital Information*, 1(8), 2001. http://jodi.ecs.soton.ac.uk/.

[19] D. Brill. *LOOM Reference Manual version 1.4*. USC-ISI, 4353 Park Terrace Drive, Westlake Village, CA 91361, 1991.

[20] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and A. Borgida. "Reducing" CLASSIC to practice: Knowledge representation theory meets reality. *Artificial Intelligence*, 114(1–2):203–237, 1999.

[21] O. Etzioni, K. Golden, and D. Weld. Tractable closed world reasoning with updates. In L. Doyle, E. Sandewall, and P. Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 178–189. Morgan Kaufmann, San Francisco, California, 1994.

[22] T. Berners-Lee. Semantic Web road map. http://www.w3.org/DesignIssues/Semantic.html, 1998.

[23] T. Clark, A. Evans, S. Kent, S. Brodsky, and S. Cook. A feasibility study in rearchitecting UML as a family of languages using a precise OO meta-modeling approach. http://www.cs.york.ac.uk/puml/mml/mmf.pdf, 2000.

[24] UML 2.0 OCL request for proposal. OMG Document ad/2000-09-03, Object Management Group, 2000. http://www.omg.org/techprocess/meetings/schedule/UML_2.0_OCL_RFP.html.

[25] M. V. Cengarle and A. Knapp. A formal semantics for OCL 1.4. In *Proceedings of the 4th International Conference on the Unified Modeling Language*, Lecture Notes in Computer Science. Springer, 2001. (To appear).

[26] A. S. Evans. Reasoning with UML class diagrams. In *Proceedings of the Workshop on Industrial Strength Formal Methods (WIFT'98)*. IEEE Press, 1998. http://www.cs.york.ac.uk/puml/papers/evanswift.pdf.

[27] D. Skogan. UML as a schema language for XML based data interchange. In *Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99)*, 1999. http://www.ifi.uio.no/~davids/papers/Uml2Xml.pdf.

[28] M. Erdmann and R. Studer. Ontologies as conceptual models for XML documents. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*. Knowledge Science Institute, University of Calgary, 1999. http://sern.ucalgary.ca/KSI/KAW/KAW99/papers.html.

[29] J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding formal semantics to the Web: building on top of RDF schema. In *Proceedings of the Workshop on the Semantic Web: Models, Architectures and Management, Fourth European Conference on Research and Advanced Technology for Digital Libraries (ECDL'2000)*, 2000. http://www.ics.forth.gr/proj/isst/SemWeb/proceedings/session2-2/paper.pdf.

[30] UBOT project home page. http://ubot.lockheedmartin.com/, 2001.

[31] K. Baclawski, M. K. Kokar, P. A. Kogut, L. Hart, J. Smith, W. S. Holmes III, J. Letkowski, and M. L. Aronson. Extending UML to support ontology engineering for the Semantic Web. In *Proceedings of the 4th International Conference on the Unified Modeling Language*, Lecture Notes in Computer Science. Springer, 2001. (To appear. Draft version available at http://www.coe.neu.edu/~jsmith/semweb.pdf).

[32] S. Melnik. UML in RDF. http://www-db.stanford.edu/~melnik/rdf/uml/, 2000.

[33] XLink project home page. http://www.w3.org/XML/Linking, 2001.

[34] XPetal sourceforge Web page. http://sourceforge.net/projects/xmodel, 2001.

Appendix

The following is an XML serialisation of the RDF model representing information about New Zealand that was shown in Figure 6.

```
<rdf:RDF
  xmlns:wfb="http://nzdis.otago.ac.nz/0_1/world-fact-book#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<wfb:Region rdf:ID="region1">
  <wfb:Region.name>New Zealand</wfb:Region.name>
  <wfb:Region.as_country rdf:resource="#country1"/>
</wfb:Region>

<wfb:Country rdf:ID="country1">
  <wfb:Country.region rdf:resource="#region1"/>
  <wfb:Country.areaComparison rdf:resource="#bag1"/>
</wfb:Country>

<rdf:Bag rdf:ID="bag1">
  <rdf:li rdf:resource="#comparison1"/>
</rdf:Bag>

<wfb:AreaComparison rdf:ID="comparison1">
  <wfb:AreaComparison.country rdf:resource="#country1"/>
  <wfb:AreaComparison.region rdf:resource="#region2"/>
  <wfb:AreaComparison.proportion>About the size of
  </wfb:AreaComparison.proportion>
</wfb:AreaComparison>

<wfb:Region rdf:ID="region2">
  <wfb:Region.name>Colorado</wfb:Region.name>
</wfb:Region>

</rdf:RDF>
```

Metamodeling Architecture of Web Ontology Languages

Jeff Z. Pan and Ian Horrocks

Information Management Group
Department of Computer Science
University of Manchester
Oxford Road Manchester M13 9PL, UK
{pan,horrocks}@cs.man.ac.uk

Abstract. Recent research has shown that RDF Schema, as a schema layer Semantic Web language, has a non-standard metamodeling architecture. As a result, it is difficult to understand and lacks clear semantics. This paper proposes a fixed layer metamodeling architecture for RDF Schema (RDFS(FA)) and demonstrates how the problems of RDF Schema can be solved under RDFS(FA). Based on this metamodeling architecture, a clear model-theoretic semantics of RDFS(FA) is given. Interestingly, RDFS(FA) also benefits DAML+OIL by offering a firm semantic basis and by solving the "layer mistake" problem.

1 Introduction

The Semantic Web, with its vision stated by Berners-lee [1], aims at developing *languages* for expressing information in a machine understandable form. The recent explosion of interest in the World Wide Web has also fuelled interest in ontologies. It has been predicted (Broekstra et al. [3]) that ontologies will play a pivotal role in the Semantic Web since ontologies can provide shared domain models, which are understandable to both human being and machines.

Ontology (Uschold and Gruninger [20]) is, in general, a representation of a shared conceptualization of a specific domain. It provides a shared and common understanding of a *domain* that can be communicated between people and heterogeneous and distributed application systems. An ontology necessarily entails or embodies some sort of world view with respect to a given domain. The world view is usually conceived as a hierarchical description of important concepts (is-a hierarchy), a set of crucial properties, and their inter-relationships.

Berners-lee [1] outlined the architecture of Semantic Web. We would like to call it a *functional architecture* because the expressive primitives are incrementally introduced from languages in the lowest layer (i.e. matadata layer) to those in the higher layer (e.g. logical layer), so that the languages in each layer can satisfy the requirements of different kinds (or levels) of applications:

1. In the *metadata layer*, a simple and general model of semantic assertions of the Web is introduced. The simple model contains just the concepts of *resource* and *property*, which are used to express the meta information and will be needed by languages in the upper layers. The Resource Description Framework (RDF) (Lassila and R.Swick [13]) is believed to be the general model in metadata layer.

2. In the *schema layer*, simple Web *ontology* languages are introduced, which will define a hierarchical description of concepts (is-a hierarchy) and properties. These languages use the general model in metadata layer to define the basic metamodeling architecture of Web ontology languages. RDF Schema (RDFS) (Brickley and Guha [2]) is a candidate schema layer language.

3. In the *logical layer*, more powerful Web *ontology* languages are introduced. These languages are based on the basic metamodeling architecture defined in schema layer, and defines a much richer set of modelling primitives that can e.g. be mapped to very expressive Description Logics (Horrocks et al. [11], Horrocks [10]) to supply reasoning services for the Semantic Web. OIL (Horocks et al. [9]) and DAML+OIL (van Harmelen et al. [22]) are well known logical layer languages.

This paper will focus on the *metamodeling architecture* other than the functional architecture. We should point out that "metamodeling" and the "metadata layer" in the functional architecture are not the same. Metadata means data about data. *Metamodeling* concerns the definition of the modelling primitives (vocabulary) used in a modelling language. Many software engineering modelling languages, including UML, are based on metamodels. Among the Semantic Web languages, the schema layer languages are responsible to build the metamodeling architecture.

In this paper, we argue that RDFS, as a schema layer language, has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the model have dual roles in the RDFS specification. Therefore, it makes the RDFS specification itself quite difficult to understand by the modellers. The even worse thing is that since the logical layer languages (e.g. OIL, DAML+OIL) are all based on the metamodeling architecture defined by schema layer languages (RDFS), these languages therefore have the similar problems, e.g. the "layer mistake" discussed in Section 2.3.

We propose a fixed layer metamodeling architecture for RDFS (we call it *RDFS(FA)*) which is similar to the metamodeling architecture of UML. We analyse the problems of the non-fixed metamodeling architecture of RDFS and demonstrate how these problems can be solved under RDFS(FA). Furthermore, We give a clear semantics to RDFS(FA).

The rest of the article is organized as follows. In Section 2 we explain the data model of RDF, RDFS and DAML+OIL, the languages belonging to the matadata level, schema level and logical level of the Semantic Web Architecture respectively. We will focus on the metamodeling architecture of RDFS and locate what the problems are and where they come from. In Section 3 we discuss the advantages and disadvantages of fixed and non-fixed layer metamodeling architecture and then briefly explain the metamodeling architecture of UML. In Section 4 we propose RDFS(FA), and give a clear semantics to RDFS(FA). We also demonstrate how the "layer mistake" problem with DAML+OIL is solved in RDFS(FA). Section 5 briefly discuss the advantages of RDFS(FA) and our attitudes on how to make use of UML in the Web ontology languages.

2 Current Data Models of Semantic Web Languages

2.1 RDF Data Model

As a Semantic Web language in the *metadata layer* of the functional architecture, RDF is a foundation for processing metadata. It provides interoperability between applications that exchange machine-understandable information on the Web. The foundation of RDF is a model for representing named properties and property values. The RDF data model provides an abstract, conceptual framework for defining and using metadata. The basic data model consists of three object types:

Resources: All things being described by RDF expressions are called *resources*. A resource may be an entire Web page, a part of a Web page, a whole collection of pages (Web site); or an object that is not directly accessible via the Web, e.g. a printed book. Resources are always named by URIs.

Properties: A *property* is a specific aspect, characteristic, attribute, or relation used to describe a resource.

Statements: A specific resource together with a named property plus the value of that property for that resource is an RDF *statement*.

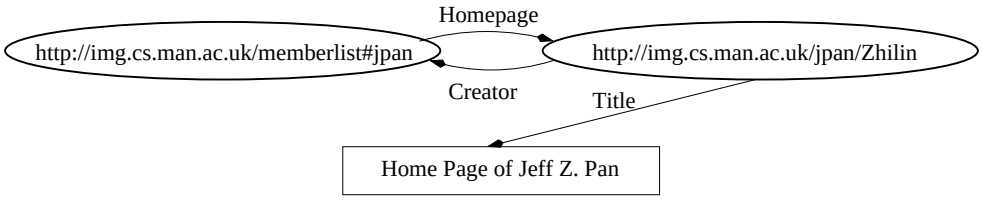


Figure 1: An Example of RDF in a Directed Labeled Graph

In a nutshell, the RDF data model is an object-property-value mechanism. The metadata information is introduced by a set of statements in RDF. There are several ways to express RDF statements. First, we can use the binary predicate form Property(object,value), e.g. Title('`http://img.cs.man.ac.uk/jpan/Zhilin`', "Home Page of Jeff Z. Pan"). Secondly, we can diagram an RDF statement pictorially using directed labeled graphs: '[object]-Property->[value]' (see Figure 1). Thirdly, RDF uses an Extensible Markup Language (XML) encoding as its interchange syntax:

```
<rdf:Description rdf:ID="http://img.cs.man.ac.uk/jpan/Zhilin">
  <Title>Home Page of Jeff Z. Pan</Title>
</rdf:Description>
```

The RDF data model is so called "property-centric". We can use the "about" attribute to add more properties to the existing resource. Generally speaking, with the object-property-value mechanism, RDF can be used to express:

- *attributes* of resources: in this case, the 'value' is a literal (e.g the "Title" property above);
- *relationships* between any two resources: in this case, the 'value' is a resource, and the involved properties represent different roles of the two resources with this relationship; in the following example, there exists a "creator-homepage" relationship between "http://img.cs.man.ac.uk/jpan/Zhilin" and "http://img.cs.man.ac.uk/memberlist#jpan" (see also Figure 1):

  ```
  <rdf:Description rdf:ID="http://img.cs.man.ac.uk/memberlist#jpan">
    <Homepage rdf:resource="http://img.cs.man.ac.uk/jpan/Zhilin"/>
  </rdf:Description>
  <rdf:Description about="http://img.cs.man.ac.uk/jpan/Zhilin">
    <Creator rdf:resource="http://img.cs.man.ac.uk/memberlist#jpan"/>
  </rdf:Description>
  ```

- *weak type* of resources: the 'type' is *weak* because RDF itself has no standard way to define a Class, so the type here is regarded only as a special attribute; for example,

  ```
  <rdf:Description about="http://img.cs.man.ac.uk/memberlist#jpan">
    <rdf:type rdf:resource="#Person"/>
  </rdf:Description>
  ```

- *statement about statement*: RDF can be used for making statements about other RDF statements, which are referred to as *higher-order statements*. This feature of RDF has yet to be clearly defined and is beyond the scope of this paper.

2.2 RDF Schema Data Model

As we have seen, on the one hand, RDF data model is enough for defining and using metadata. On the other hand, the modelling primitives offered by RDF are very basic. Although you can define "Class" and "subClassOf" as resources in RDF (no one can stop you doing that), RDF provides no standard mechanisms for declaring classes and (global) properties, nor does it provide any mechanisms for

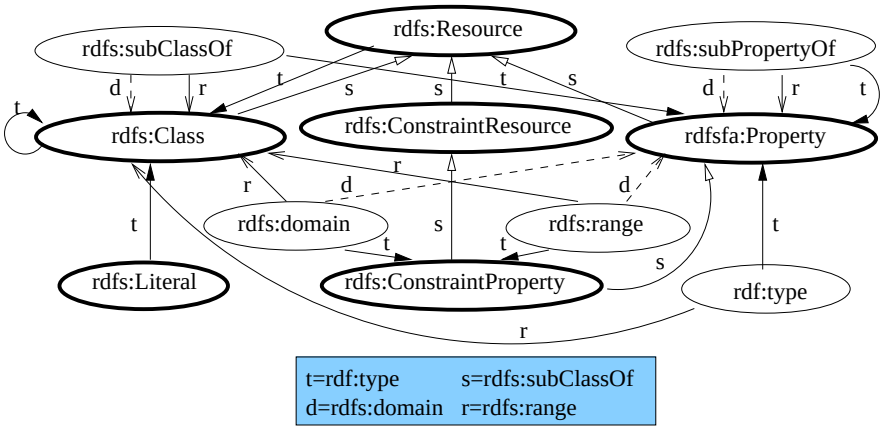


Figure 2: Directed Labeled Graph of RDF Schema

defining the relationships between properties or between classes. That is the role of RDFS–a Semantic Web language in the schema layer.

RDFS is expressed by using RDF data model. It extends RDF by giving an externally specified semantics to specific resources. In RDFS, rdfs:Class is used to define concepts, i.e. every class must be an instance of rdfs:Class. Resources that are described by RDF expressions are viewed to be instances of the class rdfs:Resource. The class rdf:Property is the class of all properties used to characterize instances rdfs:Resource. The rdfs:ConstraintResource defines the class of all constraints. The rdfs:ConstraintProperty is a subset of rdfs:ConstraintResource and rdf:Property, all of its instances are properties used to specify constraints, e.g. rdfs:domain and rdfs:range. For example, the following RDFS expressions

```
<rdfs:Class rdf:ID="Animal">
  <rdfs:comment>This class of animals is illustrative of a number of
  ontological idioms.</rdfs:comment>
</rdfs:Class>
<rdfs:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
<rdf:Description rdf:ID="John">
  <rdf:type rdf:resource="#Person"/>
  <rdfs:comment>John is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description rdf:ID="Mary">
  <rdf:type rdf:resource="#Person"/>
  <rdfs:comment>Mary is a person.</rdfs:comment>
</rdf:Description>
```

define the classes “Animal” and “Person”, with the latter being the subclass of the former, and two individuals “John” and “Mary”, which are instances of the class “Person”. Individual “John” can also be defined in this way,

```
<Person rdf:ID="John">
  <rdfs:comment>John is a person.</rdfs:comment>
</Person>
```

which is an implicit way to define rdf:type property. Note that here “Person” is a class.

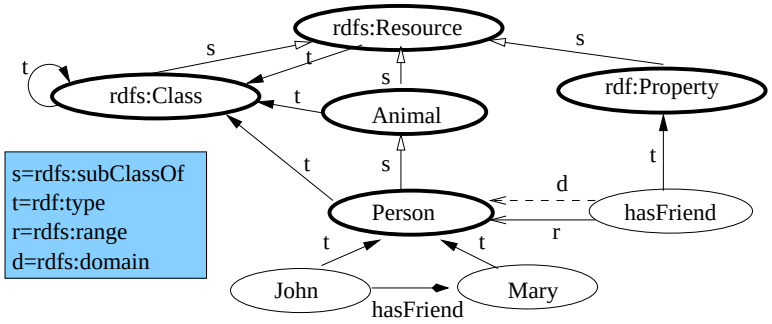


Figure 3: A “Person–hasFriend” Example of RDF Schema

Figure 2 pictures the subclass-of and instance-of hierarchy of RDFS: rdfs:Resource, rdfs:Class, rdf:Property, rdfs:ConstraintResource and rdfs:ConstraintProperty are all instances of rdfs:Class, while rdfs:Class, rdf:Property and rdfs:ConstraintResource are subclass of rdfs:Resource. It is confusing that rdfs:Class is a sub-class of rdfs:Resource, while rdfs:Resource itself is an instance of rdfs:Class at the same time. It is also strange that rdfs:Class is an instance of itself.

In RDFS, all properties are instances of rdf:Property. The rdf:type property models instance-of relationships between resources and classes. The rdfs:subClassOf property models the subsumption hierarchy between classes, and is transitive. The rdfs:subPropertyOf property models the subsumption hierarchy between properties, and is also transitive. The rdfs:domain and rdfs:range properties are used to restrict domain and range properties. For example, the following RDFS expressions

```
<rdf:Property rdf:ID="hasFriend">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
<rdf:Description about="#John">
  <hasFriend rdf:resource="#Mary"/>
</rdf:Description>
```

define a property “hasFriend” between two “Person”s and ⟨‘John’, ‘Mary’⟩ is an instance of “hasFriend” (see Figure 3).

In RDFS, properties are regarded as sets of binary relationships between instances of classes, e.g. a property “hasFriend” is a set of binary tuples between two instances of the class “Person”. One exception is the rdf:type, since it is just the *instance-of* relationship. In this sense, rdf:type is regarded as a special predefined property.

Figure 2 also shows the range and domain constraints in RDFS–rdfs:domain and rdfs:range can be used to specify the two classes that a certain property can associate with. So the rdfs:domain of rdfs:domain and rdfs:range is the class rdf:Property, the rdfs:range of rdfs:domain and rdfs:range is the class rdfs:Class. The rdfs:domain and rdfs:range of rdfs:subClassOf is rdfs:Class. The rdfs:domain and rdfs:range of rdfs:subPropertyOf is rdf:Property. The rdfs:range of rdf:type is the class rdfs:Class. The rdf:type property is regarded as a set of binary links between *instances* and classes (as mentioned above), while the value of the rdfs:domain property should be a *class*, therefore rdf:type does not have the rdfs:domain property (cf. Brickley and Guha [2]).

As we have seen, RDFS use some primitive modelling primitives to define other modelling primitives (e.g. rdf:type, rdfs:domain, rdfs:range, rdf:type and rdfs:subClassOf). At the same time, these primitives can be used to define ontologies as well, which makes it rather unique when compared to conventional model and metamodeling approaches, and makes the RDFS specification very difficult to read and to formalize (Nejdl et al. [16], Broekstra et al. [3]). For example, in Figure 3, it is confusing that although rdfs:Class is the rdf:type of “Animal”, both “Animal” and rdfs:Class are rdfs:subClassOf

rdfs:Resource, where rdfs:Class is a modelling primitive and “Animal” is an user-defined ontology class.

2.3 DAML+OIL Data Model

DAML+OIL is an expressive Web ontology language in the logical layer. It builds on earlier W3C standards such as RDF and RDFS, and extends these languages with much richer modelling primitives. DAML+OIL inherits many aspects from OIL, and provides modelling primitives commonly found in frame-based languages. It has a clean and well defined semantics based on description logics.

A complete description of the data model of DAML+OIL is beyond the scope of this paper. However, we will illustrate how DAML+OIL extends RDFS by introducing some new subclasses of rdfs:Class and rdf:Property. One of the most important classes that DAML+OIL introduces is daml:Datatype. DAML+OIL divides the universe into two disjoint parts, the object domain and the datatype domain. The object domain consist of objects that are members of classes described in DAML+OIL. The datatype domain consists of the values that belong to XML Schema datatypes. Both daml:Class (object class) and daml:Datatype are rdfs:subClassOf rdfs:Class. Accordingly, properties in DAML+OIL should be either object properties, which relate objects to objects and are instances of daml:ObjectProperty; or datatype property, which relate objects to datatype values and are instances of daml:Datatype- Property. Both daml:ObjectProperty and daml:DatatypeProperty are rdfs:subClassOf rdf:Pro- perty. For example, we can define a datatype property called “birthday”:

```
<daml:DatatypeProperty rdf:ID="birthday">
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProp-
  erty"/>
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</daml:DatatypeProperty>
```

Besides being an instance of daml:datatypeProperty, the “birthday” property is also an instance of daml:UniqueProperty, which means that “birthday” can only have one (unique) value for each instance of the “Animal” class. In fact, daml:UniqueProperty is so useful that some people even want to use it to refine DAML+OIL predefined properties, e.g. daml:maxCardinality:

```
<rdf:Property rdf:about="#maxCardinality">
  <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProp-
  erty"/>
</rdf:Property>
```

This statement seems obviously right, however, it is wrong because the semantics of daml:Uniq- ue-Property requires that only the ontology properties can be regarded as its instances (cf. van Harmelen et al. [21]). This is the so called “layer mistake”. The reason that people can easily make the above “layer mistake” lies in the fact that the schema layer language RDFS doesn’t *distinguish* the modelling information in the ontology level and that in the language level. Another example is what we had mentioned before in Figure 3, it is not appropriate that both rdfs:Class and “Animal” are rdfs:subClassOf rdfs:Resource.

It is the existence of the dual roles of some RDFS modelling elements, e.g. rdfs:subClassOf, that makes RDFS have unclear semantics. This partially explains why Brickley and Guha [2] didn’t define the semantics of RDFS. We should stress that DAML+OIL is built on top of the *syntax* of RDFS, but not the *semantics* of RDFS. On the contrary, RDFS relies on DAML+OIL to give semantics to its modelling primitives. In other words, DAML+OIL not only defines the semantics of its newly introduced modelling primitives, e.g. daml:UniqueProperty, daml:maxCardinality etc., but also the modelling primitives of RDFS, e.g. rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range etc (van Harmelen et al. [see 21]). This breaks the dependency between logical layer languages and schema

layer languages and indicates that RDFS is not yet a fully qualified schema layer Semantic Web language.

3 Fixed or Non-fixed Metamodeling Architecture?

3.1 The Advantages and Disadvantages of Non-fixed Metamodeling Architecture

The dual roles of some RDFS modelling elements indicate that something might be wrong with the metamodeling architecture of RDFS. The RDFS has a non-fixed metamodeling architecture, which means that it can have possibly infinite layers of classes. The advantage is that it makes itself compact. However, it has at least the following three disadvantages or problems:

1. The class rdfs:Class is an instance of itself. Usually, a class is regarded as a set, and an instance of the class is a member of the set. A Class of classes can be interpreted as a set of sets, which means its members are sets. In RDFS, all classes (including rdfs:Class) are instances of rdfs:Class, which is suspicious by close to Russell paradox. The paradox arises when considering the set of all sets that are not members of themselves. Such a set appears to be a member of itself if and only if it is not a member of itself, hence the paradox.

2. The class rdfs:Resource is a superclass and instance of rdfs:Class at the same time, which means that the super set (rdfs:Resource) is a member of the subset (rdfs:Class).

3. The properties rdfs:subClassOf, rdf:type, rdfs:range and rdfs:domain are used to define both the other RDFS modelling primitives and the ontology, which makes their semantics unclear and makes it very difficult to formalize RDFS. E.g. it is not clear that the semantic of rdfs:subClassOf is a set of binary relationships between two sets of objects or a set of binary relationships between two sets of sets of objects, or else.

As a result, RDFS has no clear semantics, it even rely on DAML+OIL to give itself semantics, which makes RDFS a not so satisfactory schema layer semantic Web language.

3.2 The Advantages and Disadvantages of Fixed Metamodeling Architecture

We can demonstrate the advantages of fixed metamodeling architecture by showing how the problems of RDF Schema mentioned in Section 3.1 are solved under the fixed metamodeling architecture.

The reason that problem 1 exists is that RDFS uses a single primitive rdfs:Class to implicitly represent possibly infinite layers of classes. *But do we really need infinite layers of classes?* In practice, rdfs:Class usually acts as a modelling primitive in the ontology language and is used to define ontology classes (e.g. “Person”). One reasonable solution is to explicitly specify a certain number of layers of *class* primitives, with one being an instance of another, and the *class* primitives in the top layer having no type at all, which means that it is not an instance of anything. It isn’t because it can’t have a type, but because it doesn’t have to have a type, from the pragmatic point of view. This is the main difference between the fixed and non-fixed metamodeling architecture.

But how many class primitives do we really need? Problem 2 indicates that we need at least *two* class primitives in different metamodeling layers–one as the type of rdfs:Resource, the other as a subclass of rdfs:Resource. In fact, in the four-layer metamodeling architecture of UML, there exist two class primitives in different metamodeling layers, which are *Class* in metamodel layer and *MetaClass* in meta-metamodel layer (see Section 3.3). In practice, it has not been found useful to have more than two class primitives in the metamodeling architecture (technology Inc. [19, pg. 298]). Therefore, it is reasonable to explicitly define two class primitives in different metamodeling layers of RDF Schema,

one is MClass in Metalanguage Layer and the other is LClass in Ontology Language Layer[1] (see Section 4.1). This makes RDFS have a similar metamodeling architecture to that of the well known UML, so that it is easy for the modellers to understand.

Problem 3 is mainly about predefined properties. It can be solved by specifying which level of class we intend to refer to when we use these predefined properties. (see Section 4.1).

From the discussion above, we believe that although the schema layer language won't be as compact as it is, there will be several advantages if it has a fixed metamodeling architecture:

1. We don't have to worry about Russell's Paradox.
2. It has clear formalized semantics.
3. DAML+OIL and other logical layer Semantic Web languages can be built on top of both the syntax and semantics of the RDFS with fixed metamodeling architecture.
4. It is similar to the metamodeling architecture of UML, easy to understand and use.

3.3 UML Metamodeling Architecture

The Unified Modelling Language (OMG [17]) is a general-purpose visual modelling language that is designed to specify, visualise, construct and document the artifacts of a software system. It is a standard object-oriented design language that has gained virtually global acceptance. UML has a four-layer metamodeling architecture.

1) The *Meta-metamodel Layer* forms the foundation for the metamodeling architecture. The primary responsibility of this layer is to define the language for specifying a metamodel. A meta-metamodel can define multiple metamodels, and there can be multiple meta-metamodels associated with each metamodel. Examples of meta-objects in the metamodeling layer are: MetaClass, MetaAttribute.

2) A *Metamodel* is an instance of a Meta-metamodel. The primary responsibility of the Metamodel layer is to define a language for specifying models. Examples of meta-objects in the metamodeling layer are: Class, Attribute.

3) A *Model* is an instance of a Metamodel. The primary responsibility of the Model Layer is to define a language that describes an information domain. Examples in Model layer are class "Person" and property "hasFriend".

4) *User Objects* are an instance of a Model. The primary responsibility of the User Objects Layer is to describe a specific information domain. Examples in User Objects Layer are "John", "Mary" and $\langle$'John', 'Mary'$\rangle$.

The four-layer metamodel architecture is a proven methodology for defining the structure of complex models that need to be reliably stored, shared, manipulated and exchanged (Kobryn [12]). In the next section, we will use the metamodeling methods of UML to build a fixed layer metamodeling architecture for RDFS.

4 Web Ontology Language Data Model with Fixed Metamodeling Architecture

We will now illustrate what the data model of an RDF-based Web ontology language will look like under the fixed metamodeling architecture.

[1] In this sense, there are three kinds of classes: meta classes in the Metalanguage Layer, language classes in the Language Layer and ontology classes, which are instance of LClass, in Ontology Layer.

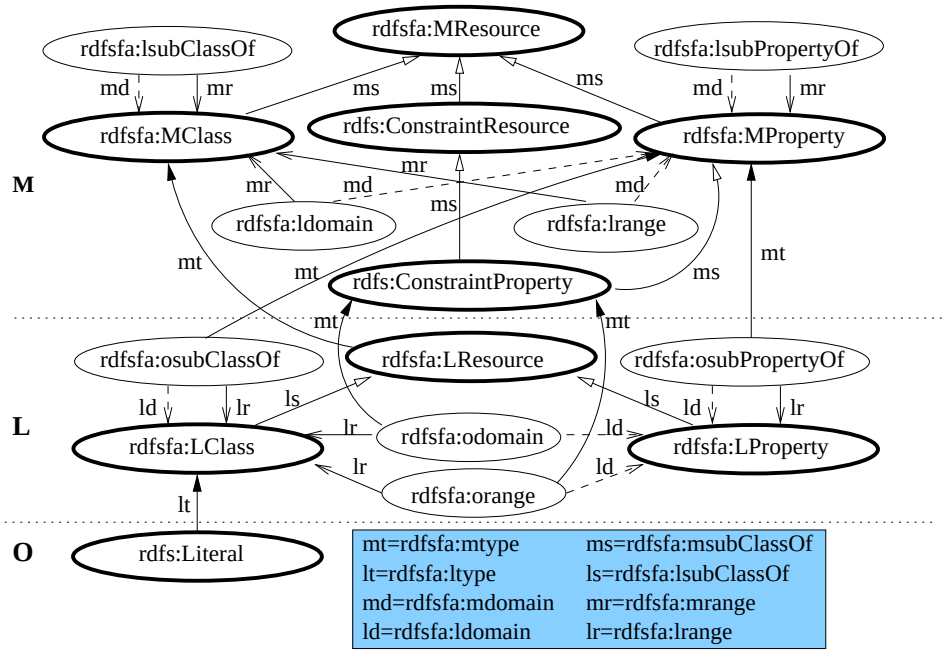


Figure 4: Directed Labeled Graph of RDFS(FA)

4.1 RDF Schema Data Model with Fixed Metamodeling Architecture

Firstly, we will map the original RDFS into RDFS under the Fixed metamodeling Architecture (or RDFS(FA) for short). One principle during this mapping is that we try to minimise the changes we make to RDFS.

As we discussed in Section 3.2, we believe it is reasonable to define a four-layer metamodeling architecture for RDFS(FA). These four metamodeling layers are:

1. The *Metalanguage Layer* (M Layer, corresponding to the Meta-metamodel Layer in UML) forms the foundation for the metamodeling architecture. Its primary responsibility is to define the language layer. All the modelling primitives in this layer have no types (see Section 3.2). Examples of modelling primitives in this layer are rdfsfa:MClass and rdfsfa:MProperty.

2. The *Language Layer* (L Layer, corresponding to the Metamodel Layer in UML), or *Ontology Language Layer*, is an instance of the Metalanguage Layer. Its primary responsibility is to define a language for specifying ontologies. Examples of modelling primitives in this layer are rdfsfa:LClass, rdfsfa:LProperty. Both of them are instances of rdfsfa:MClass.

3. The *Ontology Layer* (O Layer, corresponding to the Model Layer in UML) is an instance of Language Layer. Its primary responsibility is to define a language that describes a specific domain, i.e. an ontology. Examples of modelling primitives in this layer are “Person” and “Car”, which are instances of rdfsfa:LClass, and “hasFriend”, which is an instance of rdfsfa:LProperty.

4. The *Instance Layer* (I Layer, corresponding to the User Objects Layer in UML) is an instance of Ontology Layer. Its primary responsibility is to describe a specific domain, in terms of the ontology defined in the Ontology Layer. Examples in this layer are “Mary”, “John” and hasFriend$\langle$ ‘John’,‘Mary’$\rangle$.

RDFS(FA) is illustrated in Figure 4. We map the modelling primitives of RDFS to the primitives in corresponding metamodeling layers of RDFS(FA), so that *no* modelling primitives will have dual

roles in the metamodeling architecture of RDFS(FA).

First, we map rdfs:Class and its instance primitives in RDFS to the metamodeling architecture of RDFS(FA) as follows:

1. rdfs:Class is mapped to rdfsfa:MClass in Metalanguage Layer and rdfsfa:LClass in Language Layer, so that rdfsfa:LClass is an instance of rdfsfa:MClass.

```
<rdf:Description rdf:ID="MClass">
  <rdfs:comment>The concept of class in the Metalanguage Layer.
  </rdfs:comment>
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LClass">
  <rdfs:comment>The concept of class in the Language Layer.</rdfs:comment>
  <rdfsfa:lsubClassOf rdf:resource="#LResource"/>
</rdfsfa:MClass>
```

2. rdfs:Resource is mapped to rdfsfa:MResource in the Metalanguage Layer and rdfsfa:LResource in Language Layer, so that rdfsfa:MResource is the super class of all the modelling primitives in the Metalanguage Layer, while rdfsfa:LResource is an instance of rdfsfa:MClass and the superclass of rdfsfa:LClass.

```
<rdf:Description rdf:ID="MResource">
  <rdfs:comment>The most general resource in the Metalanguage Layer.
  </rdfs:comment>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LResource">
  <rdfs:comment>The most general resource in the Language Layer.
  </rdfs:comment>
</rdfsfa:MClass>
```

3. The rdfs:Property is mapped to rdfsfa:MProperty in the Metalanguage Layer and rdfsfa:LProperty in the Language Layer.

```
<rdf:Description rdf:ID="MProperty">
  <rdfs:comment>The concept of property in the Metalanguage Layer.
  </rdfs:comment>
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
<rdfsfa:MClass rdf:ID="LProperty">
  <rdfs:comment>The concept of property in the Language Layer.
  </rdfs:comment>
  <rdfsfa:lsubClassOf rdf:resource="#LResource"/>
</rdfsfa:MClass>
```

4. The rdfs:ConstraintResource is in the Metalanguage Layer, where it is rdfsfa:msubClassOf rdfsfa: MResource.

```
<rdf:Description rdf:ID="ConstraintResource">
  <rdfsfa:msubClassOf rdf:resource="#MResource"/>
</rdf:Description>
```

5. The rdfs:ConstraintProperty is in the Metalanguage Layer, where it is rdfsfa:msubClassOf rdfsfa: MProperty and rdfs:ConstraintResource.

```
<rdf:Description rdf:ID="ConstraintProperty">
  <rdfsfa:msubClassOf rdf:resource="#MProperty"/>
  <rdfsfa:msubClassOf rdf:resource="#ConstraintResource"/>
</rdf:Description>
```

As shown in Figure 4, modelling primitives are divided into three groups in the Metalanguage Layer, Language Layer and Ontology Layer. rdfsfa:LClass is not an instance of itself, but an instance of rdfsfa:MClass. rdfsfa:LResource is an instance of rdfsfa:MClass and a super class of rdfsfa:LClass. In general, there are three kinds of "classes" in the metamodeling architecture of RDFS(FA)[2]: *meta classes* in the Metalanguage Layer (e.g. rdfsfa:MClass, rdfsfa:MProperty), *language classes* in the Language Layer (instances of rdfsfa:MClass, e.g. rdfsfa:LClass, rdfsfa:LProperty) and *ontology class* in the Ontology Layer (instance of rdfsfa:LClass, e.g. "Person", "Car").

In order to solve problem 3 mentioned in Section 3.1, we need to be able to specify which kind of class (out of the three kinds of "classes" mentioned above) we want to refer to. In RDFS(FA), we add the layer prefix (e.g. m- for Metalanguage Layer, l- for Language Layer etc.) on the properties when we use the predefined property primitives. Based on the above principle, we can map the property primitives in RDFS to the metamodeling architecture of RDFS(FA) as follows:

1. rdfs:domain is a set of binary relationships between instances of rdf:Property and rdfs:Class. As classes and properties occur in three different layers of RDFS(FA), rdfs:domain is mapped to three different properties in RDFS(FA): rdfsfa:odomain, rdfsfa:ldomain and rdfsfa:mdomain. As shown in Figure 4, the rdfsfa:ldomain is defined in the Metalanguage Layer and used in the Language Layer, while rdfsfa:odomain is defined in the Language Layer and used in the Ontology Layer (see Figure 5).

   ```
   <rdfs:ConstraintProperty rdf:ID="odomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     ontology property describes instances of a particular ontology class.
     </rdfs:comment>
   </rdfs:ConstraintProperty>
   <rdf:Description rdf:ID="ldomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     language property describes instances of a particular language class.
     </rdfs:comment>
   </rdf:Description>
   <rdf:Description rdf:ID="mdomain">
     <rdfs:comment>This is how we specify that all instances of a particular
     meta property describes instances of a particular meta class.
     </rdfs:comment>
   </rdf:Description>
   ```

2. Similarly, rdfs:range is mapped to rdfsfa:orange, rdfsfa:lrange and rdfsfa:mrange.

   ```
   <rdfs:ConstraintProperty rdf:ID="orange">
     <rdfs:comment>This is how we specify that all instances of a particular
     ontology property have values that are instances of a particular ontolo-
     gy class.</rdfs:comment>
   </rdfs:ConstraintProperty>
   <rdf:Description rdf:ID="lrange">
     <rdfs:comment>This is how we specify the values of an instance of a
     particular language property have values that are instances of a
     particular language class. </rdfs:comment>
   ```

[2] Accordingly, there are three kinds of "properties" as well.

```
</rdf:Description>
<rdfsfa:MProperty rdf:ID="mrange">
  <rdfs:comment>This is how we specify the values of an instance of a
  particular meta property should be instances of a particular meta class.
  </rdfs:comment>
</rdf:Description>
```

3. rdf:type is a set of binary relationship between resource and rdfs:Class. As RDFS(FA) has meta classes, language classes and ontology classes, rdf:type is mapped to rdfsfa:otype, rdfsfa:ltype and rdfsfa:mtype. E.g. in Figure 4, rdfsfa:MClass is the rdfsfa:mtype of rdfsfa:LResource and rdfsfa:LClass.

```
<rdfsfa:MProperty rdf:ID="otype">
  <rdfs:comment>Indicates membership of an instance of rdfsfa:LClass
  </rdfs:comment>
  <rdfsfa:lrange rdf:resource="#LClass"/>
</rdfsfa:MProperty>
<rdf:Description rdf:ID="ltype">
  <rdfs:comment>Indicates membership of rdfsfa:LClass or rdfsfa:LProperty
  </rdfs:comment>
  <rdfsfa:mrange rdf:resource="#MClass"/>
</rdf:Description>
<rdf:Description rdf:ID="mtype">
  <rdfs:comment>Indicates membership of rdfsfa:MClass or rdfsfa:MProperty.
  </rdfs:comment>
</rdf:Description>
```

4. rdfs:subClassOf is a set of binary relationship between two instances of rdfs:Class, so rdfs:subClassOf is mapped to rdfsfa:osubClassOf and rdfsfa:lsubClassOf . E.g. in Figure 4, rdfsfa:LClass is an rdfsfa:lsubClassOf rdfsfa:LResource and rdfsfa:MClass is an rdfsfa:msubClassOf rdfs:MResource.

```
<rdfsfa:MProperty rdf:ID="osubClassOf">
  <rdfs:comment>Binary relationship between two ontology classes.
  </rdfs:comment>
  <rdfsfa:ldomain rdf:resource="#LClass"/>
  <rdfsfa:lrange rdf:resource="#LClass"/>
</rdfsfa:MProperty>
<rdf:Description rdf:ID="lsubClassOf">
  <rdfs:comment>Binary relationship between two language classes.
  </rdfs:comment>
  <rdfsfa:mdomain rdf:resource="#MClass"/>
  <rdfsfa:mrange rdf:resource="#MClass"/>
</rdf:Description>
<rdf:Description rdf:ID="msubClassOf">
  <rdfs:comment>Binary relationship between two meta classes.
  </rdfs:comment>
</rdf:Description>
```

5. Similarly, rdfs:subPropertyOf is a set of binary relationships between instances of rdf:Property, so it is mapped to rdfsfa:osubPropertyOf, rdfsfa:lsubPropertyOf and rdfsfa:msubPropertyOf .

```
<rdfsfa:MProperty rdf:ID="osubPropertyOf">
  <rdfs:comment>Binary relationship between two ontology properties.
```

```
    </rdfs:comment>
    <rdfsfa:ldomain rdf:resource="#LProperty"/>
    <rdfsfa:lrange rdf:resource="#LProperty"/>
  </rdfsfa:MProperty>
  <rdf:Description rdf:ID="lsubPropertyOf">
    <rdfs:comment>Binary relationship between two language properties.
    </rdfs:comment>
    <rdfsfa:mdomain rdf:resource="#MProperty"/>
    <rdfsfa:mrange rdf:resource="#MProperty"/>
  </rdf:Description>
  <rdf:Description rdf:ID="msubPropertyOf">
    <rdfs:comment>Binary relationship between two meta properties.
    </rdfs:comment>
  </rdf:Description>
```

6. The rdfs:comment, rdfs:label, rdfs:seeAlso and rdfs:isDefinedBy are treated as documentation in RDFS, and are not related to the semantics of RDFS(FA), so we are not going to discuss them in this paper.

Below is an RDFS(FA) version of the "Person–hasFriend" example. As with other Web ontology languages, these statements describe resources in the Ontology Layer and the Instance Layer.

```
<rdfsfa:LClass rdf:ID="Animal">
  <rdfs:comment>This class of animals is illustrative of a number of
  ontological idioms.</rdfs:comment>
</rdfsfa:LClass>
<rdfsfa:LClass rdf:ID="Person">
  <rdfs:osubClassOf rdf:resource="#Animal"/>
</rdfsfa:LClass>
<rdf:Property rdf:ID="hasFriend">
  <rdfsfa:odomain rdf:resource="#Person"/>
  <rdfsfa:orange rdf:resource="#Person"/>
</rdf:Property>
<rdf:Description rdf:ID="John">
  <rdfsfa:otype rdf:resource="#Person"/>
  <rdfs:comment>John is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description rdf:ID="Mary">
  <rdfsfa:otype rdf:resource="#Person"/>
  <rdfs:comment>Mary is a person.</rdfs:comment>
</rdf:Description>
<rdf:Description about="#John">
  <hasFriend rdf:resource="#Mary"/>
</rdf:Description>
```

In the Ontology Layer, "Animal" and "Person" are ontology classes, so they are instances of rdfsfa:LClass. The ontology class "Person" is the rdfsfa:odomain and rdfsfa:orange of the property "hasFriend", so both the values of and resource described by instances of "hasFriend" are instances of "Person". In the Instance Layer, the rdfsfa:otype of individuals such as "John" and "Mary" is the ontology class "Person". Figure 5 is a directed labeled graph of the above RDFS(FA) statements. Here the rdfsfa:mtype of rdfsfa:LClass is the meta class rdfsfa:MClass, the rdfsfa:ltype of "Person" is the language class rdfsfa:LClass and the rdfsfa:otype of "John" is the ontology class "Person". The language class rdf:Property is rdfsfa:lsubClassOf the language class rdfs:Resource while the ontology

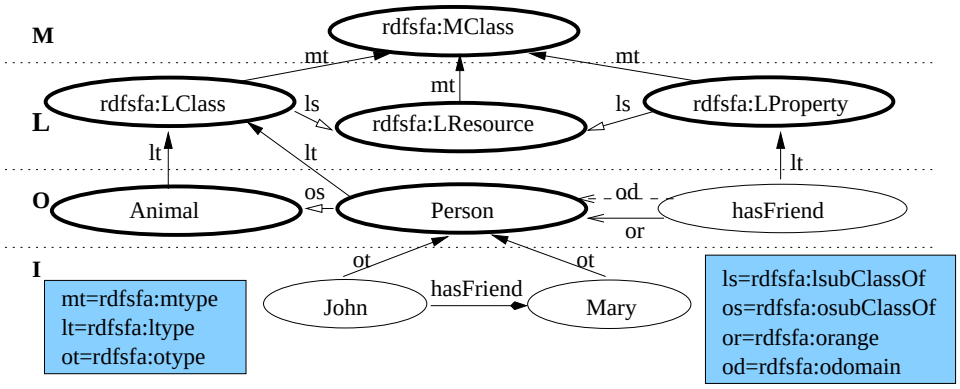


Figure 5: A "Person–hasFriend" example in RDFS(FA)

class "Person" is rdfsfa:osubClassOf the ontology class "Animal". This example clearly shows that the modelling primitives in RDFS(FA) no longer have dual roles. Thus a clear semantics can be given to them.

Note that, as in RDFS (see Section2.2), we can define rdfsfa:otype, rdfsfa:ltype and rdfsfa:mtype properties within RDFS(FA) in an implicit way as well. E.g., individual "John" can also be defined as

```
<Person rdf:ID="John">
  <rdfs:comment>John is a person.</rdfs:comment>
</Person>
```

Here "Person" is an *ontology* class, so the above expressions use an implicit way to define rdfsfa:otype property.

4.2 Data Model Semantics of RDFS(FA)

In this section, we use a Tarski style ([18]) model theoretic semantics to interpret the data model of RDFS(FA). Classes and properties are taken to refer to sets of objects in the domain of interests and sets of binary relationships (or tuples) between these objects.

In RDFS(FA), the meaning of individuals, pairs of individuals, ontology classes and properties is given by an interpretation $\mathcal{I}$, which is a pair$(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain (a set) and $\cdot^{\mathcal{I}}$ is an interpretation function, which maps every individual name x to an object in the domain $\Delta^{\mathcal{I}}$:

$$x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$$

every pair of individual names x, y to a pair of objects in $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

every ontology class name OC to a subset of $\Delta^{\mathcal{I}}$:

$$OC^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$

every ontology property name OP to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$OP^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}.$$

In the Language Layer, the interpretation function $\cdot^{\mathcal{I}}$ maps rdfsfa:LClass (LC) to $2^{\Delta^{\mathcal{I}}}$:

$$LC^{\mathcal{I}} = 2^{\Delta^{\mathcal{I}}}$$

rdfsfa:LProperty (LP) to $2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$:

$$LP^{\mathcal{I}} = 2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$$

rdfsfa:LResource (LR) to $LC^{\mathcal{I}} \cup LP^{\mathcal{I}}$:

$$LR^{\mathcal{I}} = LC^{\mathcal{I}} \cup LP^{\mathcal{I}}$$

so that the interpretation of every possible ontology class ($\text{OC}^{\mathcal{I}}$) is an element of the interpretation of rdfsfa:LClass ($\text{LC}^{\mathcal{I}}$), the interpretation of every possible ontology property ($\text{OP}^{\mathcal{I}}$) is an element of the interpretation of rdf:Property ($\text{LP}^{\mathcal{I}}$). Note that $\text{LR}^{\mathcal{I}}$ is interpreted as the union of $\text{LC}^{\mathcal{I}}$ and $\text{LP}^{\mathcal{I}}$, and not as $2^{\Delta^{\mathcal{I}} \cup (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})}$, so instances of rdfsfa:LResource must be either ontology classes (sets of objects), or ontology properties (sets of tuples), and can't be interpreted as a "mixture" of sets of objects and tuples.

In the Metalanguage Layer, interpretation function $\cdot^{\mathcal{I}}$ maps rdfsfa:MClass (MC) to $2^{LR^{\mathcal{I}}}$:

$$MC^{\mathcal{I}} = 2^{LR^{\mathcal{I}}}$$

rdfsfa:MProperty (MP) to $2^{LC^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LC^{\mathcal{I}} \times LP^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LP^{\mathcal{I}}}$:

$$MP^{\mathcal{I}} = 2^{LC^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LC^{\mathcal{I}} \times LP^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LC^{\mathcal{I}}} \cup 2^{LP^{\mathcal{I}} \times LP^{\mathcal{I}}}$$

rdfsfa:MResource (MR) to $MC^{\mathcal{I}} \cup MP^{\mathcal{I}}$:

$$MR^{\mathcal{I}} = MC^{\mathcal{I}} \cup MP^{\mathcal{I}}$$

rdfs:ConstraintResource (CR) to subset of $MR^{\mathcal{I}}$:

$$CR^{\mathcal{I}} \subseteq MR^{\mathcal{I}}$$

Predefined Property	Interpretation	Semantic Constraint
osubClassOf (OSC)	$OSC^{\mathcal{I}} \subseteq LC^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in OSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
lsubClassOf (LSC)	$LSC^{\mathcal{I}} \subseteq MC^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in LSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
msubClassOf (MSC)	$MSC^{\mathcal{I}} \subseteq 2^{MR^{\mathcal{I}}} \times 2^{MR^{\mathcal{I}}}$	$\langle C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \rangle \in MSC^{\mathcal{I}}$ iff. $C_1^{\mathcal{I}}, C_2^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
osubPropertyOf (OSP)	$OSP^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LP^{\mathcal{I}}$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in OSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in LP^{\mathcal{I}}$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
lsubPropertyOf (LSP)	$LSP^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MP^{\mathcal{I}}$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in LSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in MP^{\mathcal{I}}$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
msubPropertyOf (MSP)	$MSP^{\mathcal{I}} \subseteq \Phi \times \Phi$	$\langle P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \rangle \in LSP^{\mathcal{I}}$ iff. $P_1^{\mathcal{I}}, P_2^{\mathcal{I}} \in \Phi$ and $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
odomain (OD)	$OD^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in OD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in LP^{\mathcal{I}}$, $C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
ldomain (LD)	$LD^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in MP^{\mathcal{I}}$, $C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
mdomain (MD)	$MD^{\mathcal{I}} \subseteq \Phi \times 2^{MR^{\mathcal{I}}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MD^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in \Phi$, $C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
orange (ORG)	$ORG^{\mathcal{I}} \subseteq LP^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in ORG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in LP^{\mathcal{I}}$, $C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
lrange (LRG)	$LRG^{\mathcal{I}} \subseteq MP^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LRG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in MP^{\mathcal{I}}$, $C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
mrange (MRG)	$MRG^{\mathcal{I}} \subseteq \Phi \times 2^{MR^{\mathcal{I}}}$	$\langle P^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MRG^{\mathcal{I}}$ iff. $P^{\mathcal{I}} \in \Phi$, $C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $\forall x. \langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in P^{\mathcal{I}} \rightarrow x^{\mathcal{I}} \in C^{\mathcal{I}}$
otype (OT)	$OT^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times LC^{\mathcal{I}}$	$\langle x^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in OT^{\mathcal{I}}$ iff. $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}, C^{\mathcal{I}} \in LC^{\mathcal{I}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$
ltype (LT)	$LT^{\mathcal{I}} \subseteq LR^{\mathcal{I}} \times MC^{\mathcal{I}}$	$\langle R^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in LT^{\mathcal{I}}$ iff. $R^{\mathcal{I}} \in LR^{\mathcal{I}}, C^{\mathcal{I}} \in MC^{\mathcal{I}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$
mtype (MT)	$MT^{\mathcal{I}} \subseteq MR^{\mathcal{I}} \times 2^{MR^{\mathcal{I}}}$	$\langle R^{\mathcal{I}}, C^{\mathcal{I}} \rangle \in MT^{\mathcal{I}}$ iff. $R^{\mathcal{I}} \in MR^{\mathcal{I}}, C^{\mathcal{I}} \in 2^{MR^{\mathcal{I}}}$ and $x^{\mathcal{I}} \in C^{\mathcal{I}}$

Figure 6: Semantics of Predefined Properties in RDFS(FA)

rdfs:ConstraintProperty (CP) to subset of both $CR^{\mathcal{I}}$ and $MP^{\mathcal{I}}$:

$$CP^{\mathcal{I}} \subseteq CR^{\mathcal{I}} \cap MP^{\mathcal{I}}$$

so that the interpretations of rdfsfa:LClass ($LC^{\mathcal{I}}$), rdfsfa:LProperty ($LP^{\mathcal{I}}$) and rdfsfa:LResource ($LR^{\mathcal{I}}$) are all elements of the interpretation of rdfsfa:MClass ($MC^{\mathcal{I}}$), and all the possible pairs of subsets of $LC^{\mathcal{I}}$ and subsets of $LP^{\mathcal{I}}$ are elements of $MP^{\mathcal{I}}$.

Unlike rdfs:Class in RDFS, classes in RDFS(FA) have clear semantics. Clean semantics can also be given to the predefined properties of RDFS(FA) as shown in Figure 6, where

$$\Phi = 2^{MC^{\mathcal{I}} \times MC^{\mathcal{I}}} \cup 2^{MC^{\mathcal{I}} \times MP^{\mathcal{I}}} \cup 2^{MP^{\mathcal{I}} \times MC^{\mathcal{I}}} \cup 2^{MP^{\mathcal{I}} \times MP^{\mathcal{I}}}$$

As mentioned above, in order to specify which kind of class we want to refer to when we use the predefined properties, we add the layer prefixes to these properties. Subclass-of and subproperty-of are the subset relationship between the classes or properties within the same layer. Domain and range are foundation modelling primitives of RDFS(FA) properties, which can be used to specify two classes that a certain property can describe/use in descriptions in a certain layer. Type is a special cross-layer property, which is used to link instances to classes.

4.3 DAML+OIL Data Model with Fixed Metamodeling Architecture

With a fixed metamodeling architecture, RDFS(FA) has its own semantics and makes itself a fully qualified schema layer Semantic Web language. Thus, DAML+OIL (or any other logical layer Semantic Web language) can be built on both its syntax and semantics.

From the point of view of metamodeling architecture, the modelling primitives that DAML+ OIL introduces are mainly located in the Language Layer (a complete description of the DAML+OIL data model with fixed metamodeling architecture will be given in a forthcoming paper). daml:Class is rdfsfa:lsubClassOf rdfsfa:LClass and daml:ObjectProperty is rdfsfa:lsubClassOf rdfsfa:LProperty; both daml:Datatype and daml:DatatypeProperty are rdfsfa:lsubClassOf rdfsfa:LResource. The above four are disjoint with each other. The "birthday" property lies in the Ontology Layer and can be defined in the following way:

```
<daml:DatatypeProperty rdf:ID="birthday">
  <rdfsfa:ltype rdf:resource="http://www.daml.org/2001/03/daml+oil#Unique-
  Property"/>
  <damlfa:odatadomain rdf:resource="#Animal"/>
  <damlfa:odatarange rdf:resource="http://www.w3.org/2000/10/XMLSchema#da-
  te"/>
</daml:DatatypeProperty>
```

where damlfa:odatadomain is a set of binary relationships between instances of daml:DatatypeProperty and daml:Class, and damlfa:odatarange is a set of binary relationships between instances of daml:DatatypeProperty and daml:Datatype.

On the other hand, it is clear that Language Layer primitives can't be used to define/modify other Language Layer primitives, e.g. Uniqueproperty can not be used to restrict the numbers of values of the maxCardinality as follows:

```
<rdfsfa:MProperty rdf:about="#maxCardinality">
  <rdfsfa:ltype rdf:resource="http://www.daml.org/2001/03/daml+oil#Unique-
  Property"/>
</rdfsfa:MProperty>
```

In RDFS(FA), Language Layer properties can only be defined using Metalanguage Layer primitives, for which DAML+OIL doesn't provide any semantics. This is not clear in RDFS, where modellers

might be tempted to think that they can modify DAML+OIL in the above manner, exploiting the semantics of DAML+OIL itself. To solve the above problem, one can define MUniqueProperty in the Metalanguage Layer and then set daml:maxCardinality as its instance.

In short, RDFS(FA) not only provides a firm semantic basis for DAML+OIL, it also eradicates the possibility of the "layer mistake" mentioned above.

5 Discussion

A fixed layer metamodeling architecture for RDFS is proposed in this paper. We demonstrate how to map the original RDFS to RDFS under the Fixed metamodeling Architecture (RDFS(FA)) and give a clear model-theoretic semantics to RDFS(FA). We believe that although RDFS(FA) won't be as compact as RDFS, there will be several advantages if RDFS has a fixed metamodeling architecture:

1. We don't have to worry about Russell's Paradox. (Other ways of thinking may include non well-founded sets.)
2. RDFS(FA) has a clear formalized semantics.
3. DAML+OIL and other logical layer Semantic Web languages can be built on top of both the syntax and semantics of RDFS(FA).
4. The metamodeling architecture of RDFS(FA) is similar to that of UML, so it is easier for people to understand and use.

Some other papers have also talked about UML and the Web ontology language. Chang [4] summarized the relationship between RDF-Schema and UML. Melnik [14] tried to make UML "RDF-compatible", which allows mixing and extending UML models and the language elements of UML itself on the Web in an open manner. Cranefield and Purvis [5] investigated the use of UML and OCL (Object Constraint Language) for the representation of information system ontologies. Cranefield [6] proposed UML as a Web ontology language. Cranefield [7] described technology that facilitates the application of object-oriented modelling, and UML in particular, to the Semantic Web. However, none of these works address the problem of the metamodeling architecture of RDFS itself.

It is well known that UML has a well-defined metamodeling architecture (Kobryn [12]). It refines the semantic constructs at each layer, provides an infrastructure for defining metamodel extensions, and aligns the UML metamodel with other standards based on a four-layer metamodeling architecture, such as the Case Data Interchange Format (EIA [8]), Meta Object Facility (MOF-Parners [15]) and XMI Facility for model interchange (XMI-Parners [23]).

However, We believe Semantic Web languages and UML have different motivation and application domain. Besides the metamodeling architecture, Semantic Web languages also have a *functional architecture*. Within this functional architecture, RDF is a good candidate for the metadata layer language, while UML is obviously not designed as a metadata language. The schema layer languages must support global properties (anyone can say anything about anything) rather than the local ones, while the considerations of UML mainly focus on the local properties. The modelling primitives of logical layer languages, e.g. OIL and DAML+OIL, are carefully selected so that they can be mapped onto very expressive description logics (DLs), so as to facilitate the provision of reasoning support; on the UML side, reasoning over OCL is still under research.

Therefore, we prefer to enhance Web ontology languages by using the methodologies in UML, rather than making UML a component in Web ontology languages. Accordingly, we have used the metamodeling methods of UML to build a fixed layer metamodeling architecture for RDFS in this paper. Further research will include a detailed study of the data model of DAML+OIL based on RDFS(FA) and the reasoning support provided by corresponding Description Logics.

References

[1] Tim Berners-lee. Semantic Web Road Map. W3C Design Issues. URL http://www.w3.org/DesignIssues/Semantic.html, Oct. 1998.

[2] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommentdation, URL http://www.w3.org/TR/rdf-schema, Mar. 2000.

[3] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. can Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by extending RDF Schema, Nov. 2000.

[4] Walter W. Chang. A Discussion of the Relationship Between RDF-Schema and UML. W3C Note, URL http://www.w3.org/TR/NOTE-rdf-uml/, Aug. 1998.

[5] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *IJCAI-99 Workshop on Intelligent Information Integration*, 1999.

[6] Stephen Cranefield. Networked Knowledge Representation and Exchange using UML and RDF. In *Journal of Digital Information*, volume 1 issue 8. Journal of Digital Information, Feb. 2001.

[7] Stephen Cranefield. UML and the Semantic Web, Feb. 2001. ISSN 1172-6024. Discussion Paper.

[8] EIA. CDIF Framework for Modeling and Extensibility, EIA/IS-107. Nov. 1993.

[9] I. Horocks, D.Fensel, J.Broestra, S.Decker, M.Erdmann, C.Goble, F.van Harmelen, M.Klein, S.Staab, R.Studer, and E.Motta. The Ontology Inference Layer OIL. Aug. 2000.

[10] I. Horrocks. Benchmark Analysis with FaCT. In *TABLEAUX-2000*, number 1847 in LNAI, pages 62–66. Springer-Verlag, 2000.

[11] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[12] Cris Kobryn. UML 2001: A Standardization Odyssey. In *Communications of the ACM*, Vol.42, No. 10, Oct. 1999.

[13] Ora Lassila and Ralph R.Swick. Resource Description Framework (RDF) Model and Syntax Specification. Feb. 1999.

[14] S. Melnik. Representing UML in RDF. URL http://www-db.stanford.edu/~melnik/rdf/uml/, 2000.

[15] MOF-Parners. Meta Object Facility Revision 1.1b1. Jan. 1997.

[16] W. Nejdl, M. Wolpers, and C. Capella. The RDF Schema Specification Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000*, Apr. 2000.

[17] OMG. OMG Unified Modeling Language Specification vertion 1.3. Jun. 1999.

[18] A. Tarski. *Logic, Semantics, Mathemetics: Papers from 1923 to 1938*. Oxford University Press, 1956.

[19] PLATINUM technology Inc. Object Analysis and Design Facility, Response to OMG/OA&D RFP-1,Version 1.0, Jan. 1997.

[20] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *The Knowledge Engineering Review*, 1996.

[21] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. A Model-Theortis Semantics of DAML+OIL(March 2001). Mar. 2001.

[22] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference Description of the DAML+OIL(March 2001) Ontology Markuk Language. DAML+OIL Document, `URL http://www.daml.org/2000/12/reference.html`, Mar. 2001.

[23] XMI-Parners. XML Metadata Interchange (XMI) v. 1.0. Oct. 1998.

DAML+OIL is not Enough

Sean Bechhofer Carole Goble Ian Horrocks
Information Management Group
Computer Science Department
Kilburn Building
University of Manchester
Oxford Road
Manchester M13 9PL

http://img.cs.man.ac.uk

seanb@cs.man.ac.uk

1 Introduction

As is well recognised within the Semantic Web community, ontologies will play a crucial part in the delivery of the Semantic Web, facilitating the sharing of information between communities, both of people and software agents.

In order to support this use of ontologies, a number of representational formats have been proposed, including RDF Schema (RDF(S)) [RDF], the Ontology Interchange Language (OIL) [OIL] and the Darpa Agent Markup Language (DAML) [DAM]. These last two have been brought together to form DAML+OIL, a language now being proposed as a W3C standard for ontological and metadata representation.

DAML+OIL draws heavily on the original OIL specification, but has some key differences. In this paper, we highlight some of those differences, in particular the contrast in modelling primitives available in OIL and DAML+OIL, and discuss the impact that this may have on the use of DAML+OIL as a format for exchange, modelling and delivery of ontologies.

The original purpose of OIL was to enable the sharing – the exchange – of ontologies. This sharing and exchange can be seen to have (at least) two dimensions:

- the unequivocal sharing of semantics so that when the ontology is deployed it can be interpreted in a consistent manner;
- ensuring that when the ontology is viewed by an agent (in particular here a person) other than the author, the intention of the author is clear.

The latter is essential for cases where a) the ontology will be reused by another ontologist; or b) the ontology will be exposed through some kind of browser, editor or query interface.

These issues have been raised in [Euz00], which considers three levels of understanding when considering exchange languages: **syntactic**, **semantic** and **semiotic**. The first two levels are required in order to support our first requirement, and are not under debate in this particular context as languages such as DAML+OIL and OIL provide a well defined syntax and semantics for the constructions in the language. The third level is of more interest, as it is this semiotic level that impacts on the clarity of an ontology when presented to someone other than the author.

The motivation behind OIL's adoption of frame's modeling constructs was to facilitate the faithful capture of the epistemology of the modelling process. Other information such as *argumentation* can be seen as an important part of the acquisition and development of reusable or shareable ontologies [UG96]. We should also bear in mind Gruber's principles for ontology design [Gru93] which include the desire for properties such as *clarity*, *extendability* and a *minimal encoding bias*.

The drift of DAML+OIL away from constructs that are epistemologically supportive only serves the *deployment* purpose and fails somewhat in supporting authoring. In this paper we argue why this is relevant, and show how this drift damages the ease of construction of tools such as **OilEd** (see Section 4).

2 OIL

It is of use at this point to revisit the motivation for OIL and look at the factors which had an influence on the language. The Ontology Inference Layer (OIL) is a language developed by the OIL consortium. It has been discussed in a number of papers [FHvH$^+$00, BKD$^+$00] and we do not intend to present it in detail here. In our discussion of OIL here, please note that we refer to the language as defined by [OIL00]. For good reasons, OIL draws on three roots as depicted in Figure 1:

- Frame-based Representations;
- Description Logics;
- Web based languages;

2.1 Frame-based Representations

Frame-based and object-oriented approaches to modelling employ modelling primitives based on classes (or frames) with certain properties known as attributes. These attributes have local, rather than global, scope, and are applicable to the classes they are defined for. OIL embraces this approach and allows the definition of a class in terms of a collection of superclasses and a collection of attribute or slot constraints.

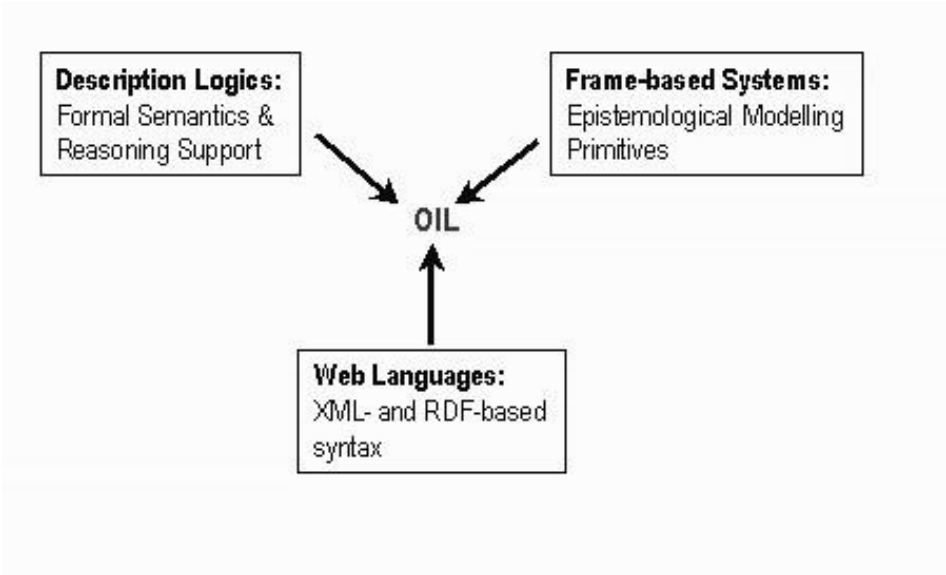


Figure 1: The Roots of OIL

Frames thus supply what is arguably a “natural” style and “friendly” face to the modeller. Frame-based representations can suffer, however, from a lack of a well-defined semantics. for example it is sometimes not clear whether a slot constraint represents a universal quantification – all fillers must take a particular value – or an existential quantification – there is a filler with a particular value. This makes reasoning or computation over a frame-based representation troublesome.

OIL itself was influenced by XOL, an early proposed ontology standard from the BioOntology Core Group[1] based on OKBC-Lite. Frame based representations have successfully been used within the Bioinformatics community for some time [SGB00], for example EcoCyc [Eco] and RiboWeb [Rib].

2.2 Description Logics

Description Logics (DLs) describe knowledge in terms of concepts and role restrictions that can then be used to automatically derive classification hierarchies. DLs allow the definition of classes in terms of descriptions that specify the properties satisfied by objects belonging to the concept. DLs will, in general, supply a range of concept forming operators that can be used in these descriptions, including conjunction, disjunction, negation, and various forms of role quantification. A key aspect of DLs is their formal semantics and reasoning support. DLs define fragments of first-order logic which in general have high expressive power but which still allow for decidable and efficient inference procedures.

Description logics are hard to interact with directly, however. In the past, DLs have been delivered as large, monolithic systems requiring users to model in the underlying syntax.

[1] `http://smi-web.stanford.edu/projects/bio-ontoloogy`

UK-Animal-lover	
superclass	Person
has-pet	$>$ 3 Animal
lives-in	UK

Figure 2: Example Frame

OIL draws from DL languages and provides a range of expressive concept forming operators. In addition, OIL inherits a formal semantics and reasoning procedures from the DL world but without compromising usability. OIL adds extra mechanisms such as recursive class definitions and more general axioms to the basic frame-based modelling primitives, producing a powerful hybrid. This relationship with DL languages is made explicit through the provision of a mapping from OIL to the Descripion Logic $\mathcal{SHIQ}$. Of course, one could argue that via this mapping from OIL to $\mathcal{SHIQ}$, OIL itself is simply an alternative syntax for a DL. This is true in some respects, but the thesis of this paper is that the alternative presentation of OIL *is* important and offers a different modelling experience to the user than that obtained when using the underlying raw logic.

2.3 Web based languages

In addition to the definition of modelling primitives and their semantics, an ontology representation and exchange language requires a delivery format and concrete syntax. Schemas have been defined for OIL in terms of both XML-Schema and RDF schema, allowing OIL to sit happily alongside existing standards. In particular, as OIL extends RDF Schema, an RDFS-aware application may be able to read OIL ontologies and extract basic class hierarchies without necessarily being OIL-aware.

Figure 2 shows an informal example of a frame. This describes a UK animal lover as a person with at least 3 pets who lives in the UK. Note that within such a description it is not always clear whether the intended interpretation of a slot fillers is as a universal or existential quantification.

3 DAML+OIL

DAML+OIL is a more recent proposal for an ontology representation language that has emerged from work under DARPA's Agent Markup Language (DAML) initiative along with input from leading members of the OIL consortium. DAML+OIL draws heavily on the original OIL language, but differs in a number of ways. In particular, DAML+OIL has moved away from the original frame-like ideals of OIL and is, in a much stronger sense than OIL, an alternative syntax for a Description Logic.

Assertions in a DAML+OIL ontology (such as the superclasses or slot constraints applying to a class) are couched in terms of general axioms. The idea of a "frame", a single place in

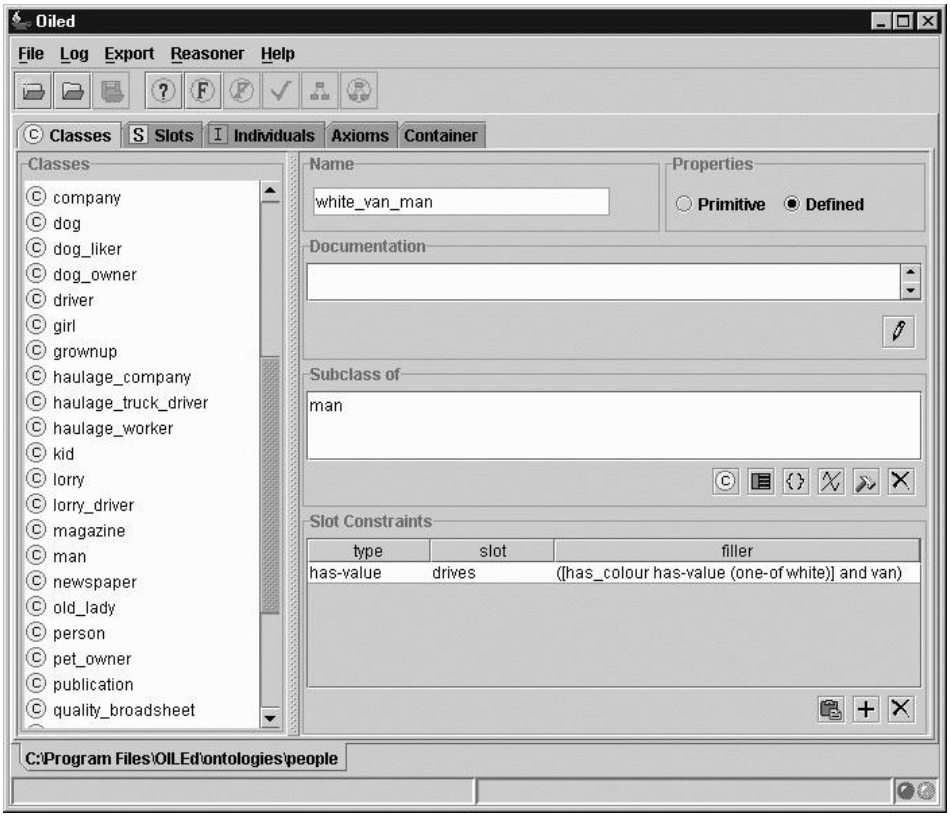


Figure 3: OilEd

which facts about a class are gathered is lost, or is at least not inherent in the language.

4 The OilEd Experience

OilEd is a simple ontology editor[2]. It was developed initially as a demonstration of the possibilites and benefits of using as reasoner to classify ontologies, but has enjoyed some success as an ontology editor in its own right. A major factor in this success was the adoption of a "frame-based" paradigm, closely tied to the underlying OIL language description. As discussed in [SHGB01], the use of OIL and its frame-like approach proved vital in supporting biologists involved in a modelling exercise.

Although frames can have their associated problems (for example the problems of interpretation as introduced earlier), OIL's well defined semantics has helped to alleviate these, allowing the use of DL-based semantics and a reasoner. **OilEd** can communicate with the FaCT reasoner using its CORBA interface [BHPST99]. This allows **OilEd** to classify and organise concept hierarchies, spot inconsistencies, and means that the modeller can construct the model through the use of descriptions rather than explicitly building hierarchies.

Figure 3 shows an example class description panel from OilEd. It shows the description of a class in terms of its explicit superclasses, along with a collection of slot constraints. Tools

[2]Sometimes described as the "NotePad" of ontology editors.

class-def defined White-van-man
  subclass-of Man
  slot-constraint drives
    has-value White-van

covered White-van-man *by* Aggressive-driver

Figure 4: Definition of White Van Man

such as Protege 2000 [GEF+99] (from which **OilEd** draws much influence) and OntoEdit [SM00] also use the frame-based paradigm. Note that in contrast to Figure 2, the drives slot here is explicitly typed as *has-value*, the OIL primitive for an existential quantification.

The original implementation of **OilEd** predates the definition of DAML+OIL, and the internal representations used for ontologies follow very closely those in the original OIL language. **OilEd** can now read and write DAML+OIL (using the RDFS format), but during the development of the tool, a number of issues came up, in particular a mismatch between the underlying models of OIL and DAML+OIL.

5 DAML+OIL vs. OIL

As introduced above, assertions in DAML+OIL are couched in terms of axioms. This has the effect that all descriptions of concepts are collapsed into a collection of axioms, possibly losing information about the way in which the model was constructed. Returning to our original motivations, if we are simply considering the delivery of ontologies to applications that then need to use that information for reasoning or queries, this is unlikely to be an issue.

However, if we consider the activity of modelling and exchange of ontologies between, for example, ontologists, this is of importance. OIL allows the modeller to state things in more than one way. For example, we can define White-van-man[3] as a man who drives a white van. In addition, we can add an axiom that states that White-van-man is an aggressive driver. The corresponding OIL (in terms of OIL's textual representation) appears in Figure 4.

Alternatively, we could simply introduce the class White-van-man, and then make a number of assertions (through equivalence and covering axioms) about the class, as shown in Figure 5.

The semantics of both of these sets of definitions (in terms of their mapping to the underlying DL) are identical. However, we can argue that the alternative organisation of the facts carries some extra information about the way that the ontologist has chosen to produce the model.

This is not, in itself, a problem. When modellers choose to use tools such as **OilEd** or Protege to construct ontologies, however, it becomes more important. In order to read in an ontology from a DAML+OIL description, we need to be able to reconstruct frame-style descriptions

[3]The term White Van Man was first coined in 1997, and has come to represent a particular class of driver in the UK. For more information, see `http://www.sirc.org/publik/white_van_man.html`.

class-def primitive White-van-man

equivalent
White-van-man
(man *and*
(*slot-constraint* drives
has-value White-van))

covered White-van-man *by* Aggressive-driver

Figure 5: Alternative definition of White Van Man through axioms

of concepts. This is not always possible to do in a consistent manner. In our example, when faced with the alternative presentation, we cannot tell that the original intention was that the first axiom should be taken as the definition, while the second is some "extra" information.

Both of these descriptions would map to the same set of DAML+OIL axioms as shown in a DAML+OIL form in Figure 6, and an editor (or ontologist) would be unable to determine which was the original construction. The issue here is concerned with the levels of understanding as discussed in [Euz00]. There is no debate over the lexical or semantic levels of understanding as these are well catered for in the language. Here, as discussed in Section 1 we are concerned with the **semiotic** level which is particularly important when dealing with exchange of models between people and the *faithful* reproduction of these representations.

This has ramifications not just for the process of exchange, but impacts on tool developers wishing to use DAML+OIL as a representational format. The prevalence of frame-based ontology editors and their popularity among users suggests that the frame-based paradigm is appropriate for such tools. Description Logic languages certainly have a place in the toolkit of the conceptual modeller but they have not gained much popularity as raw tools for conceptual modelling in the past. This is unlikely to change.

If the developer of tool X wishes to preserve information about the way in which the model is constructed, then information (representing for example whether class definitions or axioms were used) will need to be kept in addition to the DAML+OIL encoding of the ontology. If ontologies are then shared between users of tool X, this extra information must be shared too. The extensible nature of RDF makes this feasible (if we use an RDF Schema based exchange format), and DAML+OIL ontologies with this extra information could then be used. If the users of tool Y also wish to use this information, though, the developers of tool Y will also need to be aware of X's (non-standard) extensions to the DAML+OIL format. In effect, we are introducing a new standard that extends the original. Care must be taken if these extensions are to be maintained consistently – this can place barriers on the ease with which exchange can be supported between and within communities.

```
<rdfs:Class rdf:ID="White-van-man">
  <rdfs:subClassOf>
    <rdfs:Class rdf:about="Aggressive-driver"/>
  </rdfs:subClassOf>
</rdfs:Class>

<rdfs:Class rdf:about="White-van-man">
  <daml:sameClassAs>
    <rdfs:Class>
      <daml:intersectionOf>
        <rdfs:Class rdf:about="man"/>
        <daml:Restriction>
          <daml:onProperty rdf:resource="drives"/>
          <daml:hasClass rdf:resource="White-van"/>
        </daml:Restriction>
      </daml:intersectionOf>
    </rdfs:Class>
  </daml:sameClassAs>
</rdfs:Class>
```

Figure 6: DAML+OIL description of White Van Man

6 Conclusions

It must be stressed that this paper is not intended as a general criticism of DAML+OIL. Languages like OIL and DAML+OIL are crucial to the success of the Semantic Web – without well-defined semantics and inference procedures, agents will not be able to consistently process information. As a delivery platform for ontologies, DAML+OIL is quite satisfactory and indeed, in the opinions of the authors, is a great improvement over alternative representations such as simple RDF Schema or Topic Maps. However, as an exchange and modelling format, DAML+OIL is lacking in the areas outlined above. To quote from [Euz00], *"good understanding cannot be ensured by meaning preservation"*.

The authors would be the first to admit that this is neither a radical discovery nor a shocking conclusion. We must, however, be careful that in adopting DAML+OIL we do not lose the features that made OIL such an attractive proposition as a language not only for Ontology representation and delivery, but also for sharing and exchange.

7 Acknowledgements

This work was supported by EPSRC Grant GR/M75426.

References

[BHPST99] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris. A proposal for a description logic interface. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 33–36, 1999.

[BKD+00] J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding formal semantics to the web: building on top of rdf schema. In *Proc. SemWeb 2000*, 2000.

[DAM] DAML Project. `http://www.daml.org`.

[Eco] EcoCyc Web site. `http://ecocyc.PangeaSystems.com/ecocyc/ecocyc.html`.

[Euz00] Jrme Euzenat. Towards formal knowledge intelligibility at the semiotic level. In *ECAI 2000 Workshop Applied Semiotics: Control Problems, Berlin (DE)*, pages 59–61, 2000.

[FHvH+00] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*. Springer-Verlag, 2000.

[GEF+99] William E. Grosso, Henrik Eriksson, Ray W. Fergerson, John H. Gennari, Samson W. Tu, and Mark A. Musen. Knowledge modeling at the millennium (the design and evolution of protégé-2000). In *Proc. of KAW99*, 1999.

[Gru93] T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In R. Guarino, N. Poli, editor, *International Workshop on Formal Ontology*, Padova, Italy, 1993.

[OIL] Ontology Inference Layer. `http://www.ontoknowledge.org/oil/`.

[OIL00] OIL RDF Schema. `http://www.ontoknowledge.org/oil/rdf-schema/2000/11/10-oil-standard`, 2000.

[RDF] Resource Description Format. `http://www.w3.org/RDF/`.

[Rib] RiboWeb Web site. `http://smi-web.stanford.edu/projects/helix/riboweb.html`.

[SGB00] Robert Stevens, Carole A. Goble, and Sean Bechhofer. Ontology-based knowledge representation for bioinformatics. *Briefings in Bioinformatics*, 1(4):398–414, nov 2000.

[SHGB01] Robert Stevens, Ian Horrocks, Carole Goble, and Sean Bechhofer. Building a Reason-able Bioinformatics Ontology Using OIL. In *Some IJCAI Workshop*, 2001.

[SM00] S. Staab and A Maedche. Ontology Engineering beyond the Modeling of Concepts and Relations. In *ECAI'2000 Workshop on Applications of Ontologies and Problem-Solving Methods, Berlin, 2000.*, 2000.

[UG96] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.

Semantic Web Modeling and Programming with XDD

Chutiporn Anutariya[1], Vilas Wuwongse[1], Kiyoshi Akama[2] and Vichit Wattanapailin[1]
[1] *Computer Science & Information Management Program,*
School of Advanced Technologies, Asian Institute of Technology
Pathumtani 12120, Thailand

[2] *Center for Information and Multimedia Studies,*
Hokkaido University, Sapporo 060, Japan

Abstract. *XML Declarative Description* (*XDD*) is a unified modeling language with well-defined declarative semantics. It employs XML as its bare syntax and enhances XML expressive power by provision of mechanisms for succinct and uniform expression of Semantic Web contents, rules, conditional relationships, integrity constraints and ontological axioms. Semantic Web applications, offering certain Web services and comprising the three basic modeling components: application data, application rules and logic, and users' queries and service requests, are represented in XDD language as *XDD descriptions*. By integration of XDD language, *Equivalent Transformation* computational paradigm and XML syntax, *XML Equivalent Transformation* (*XET*)—a declarative programming language for computation of XDD descriptions in *Equivalent Transformation* computational paradigm—is developed. By means of XDD and XET languages, a new declarative approach to the development and the execution of Semantic Web applications is constructed.

Keywords. Semantic Web, Semantic Web applications, Semantic Web services, XML Declarative Description, XML Equivalent Transformation.

1 Introduction

The *Semantic Web* [7] is a vision of the next-generation Web which enables Web applications to automatically collect Web contents from diverse sources, integrate and process information, and interoperate with other applications in order to execute sophisticated tasks for humans. For the current Web to evolve from a global repository of information primarily designed for human consumption into the Semantic Web, tremendous effort has been devoted to definition and development of various supporting standards and technologies. Prominent markup languages with an aim to define a syntax convention for descriptions of the semantics of Web contents in a standardized interoperable manner include *XML*, *RDF*, *RDF Schema*, *OIL* [6,8,12] and *DAML+OIL* [13]. Moreover, for Web applications to effectively communicate and interoperate in the heterogeneous environment, a standard *Agent Communication Language* (*ACL*) [15] becomes a necessity. Two major current ACLs are *Knowledge Query and Manipulation*

Language (*KQML*) [9] and *Foundation for Intelligent Physical Agents ACL* (*FIPA-ACL*) [10,15].

With an emphasis on the modeling and the development of Semantic Web applications offering certain Web services, there arises a need for a tool which is capable of modeling their three major components: *application data*, *application rules and logic*, and *queries and requests*. *XML Declarative Description* (*XDD*) [5,17]—a unified, XML-based Semantic Web modeling language with well-defined semantics and a support for general inference mechanisms—aims to fulfill such a requirement. XDD does not only allow direct representation and manipulation of machine-comprehensible Web contents (such as documents, data, metadata and ontologies, encoded in XML, RDF, OIL or DAML+OIL syntax), but also provides simple, yet expressive means for modeling their conditional relationships, integrity constraints and ontological axioms as well as Semantic Web applications. XDD serves the three important roles: *content language*, *application-rule language* and *query or service-request language*, in modeling such three main components of Semantic Web applications.

Based on XDD language, a declarative programming language, i.e., *XML Equivalent Transformation* (*XET*) is constructed. Given an application's model specification, represented in terms of an XDD description, an XET program capable of executing and handling the application's queries as well as service requests can be obtained directly.

Thus, the developed technologies—XDD and XET languages—present a new paradigm for modeling and programming Semantic Web applications. By integration with existing Web and agent technologies, XDD and XET also allow both syntactic and semantic interoperability among Web applications, and hence enable the development of intelligent services as well as automated software agents.

Section 2 formalizes an extended XDD language with set-of-reference functions, Section 3 presents an XDD approach to modeling Semantic Web resources and applications, Section 4 describes XET programming language and outlines an approach to its employment in Web application development, Section 5 demonstrates a prototype system which adopts the developed technologies, Section 6 reviews current related works, and Section 7 draws conclusions.

2 XML Declarative Description

XDD [5,17] is a language the *words* and *sentences* of which are *XML expressions* and *XML clauses*, respectively. XML expressions are used to express explicit and implicit as well as simple and complex facts, while XML clauses are employed to represent ontology, implicit and conditional relationships, constraints and axioms. First, the data structure of XML expressions and their sets, characterized by an *XML Specialization System*, will be given and then followed by the syntax and semantics of XML clauses.

2.1 XML Specialization System

XML expressions have a similar form to XML elements except that they can carry variables for representation of implicit information and for enhancement of their expressive power. Every component of an XML expression—the expression itself, its tag name, attribute names and values, pairs of attributes and values, contents, sub-expressions as well

Table 1: Variable types.

Variable Type	Variable Names Beginning with	Instantiation to
N-variables: Name-variables	`$N`	Element types or attribute names
S-variables: String-variables	`$S`	Strings
P-variables: Attribute-value-pair-variables	`$P`	Sequences of zero or more attribute-value pairs
E-variables: XML-expression-variables	`$E`	Sequences of zero or more XML expressions
I-variables: Intermediate-expression-variables	`$I`	Parts of XML expressions
Z-variables: Set-variables	`$Z`	Sets of XML expressions

as some partial structures—can contain variables. XML expressions without variables are called *ground XML expressions* or simply *XML elements*, those with variables *non-ground XML expressions*. Table 1 defines all types of variables and their usages.

An *XML expression* takes formally one of the following forms:

1. *evar*,
2. $<t\ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k />$,
3. $<t\ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k> v_{m+1} </t>$,
4. $<t\ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k> e_1 \ldots e_n </t>$,
5. $<ivar> e_1 \ldots e_n </ivar>$,

where
- *evar* is an *E*-variable,
- $k, m, n \geq 0$,
- t, a_i are names or *N*-variables,
- $pvar_i$ is a *P*-variable,
- v_i is a string or an *S*-variable,
- *ivar* is an *I*-variable,
- e_i is an XML expression.

The domain of XML expressions and their sets can be defined as follows:

- $\mathcal{A}_X$: the set of all XML expressions,
- $\mathcal{G}_X$: the subset of $\mathcal{A}_X$ which comprises all ground XML expressions in $\mathcal{A}_X$,
- $\mathcal{A} = \mathcal{A}_X \cup 2^{(\mathcal{A}_X \cup V_Z)}$: the set of all XML expressions in $\mathcal{A}_X$ and sets of XML expressions and *Z*-variables in $2^{(\mathcal{A}_X \cup V_Z)}$, and
- $\mathcal{G} = \mathcal{G}_X \cup 2^{\mathcal{G}}$: the set of all ground XML expressions in $\mathcal{G}_X$, and sets of ground XML expressions in $2^{\mathcal{G}_X}$.

Note that elements of the sets $\mathcal{A}$ and $\mathcal{G}$ may be at times referred to as *objects* and *ground objects*, respectively, and when it is clear from the context, a singleton $\{X\}$ where $X \in V_Z$ is a *Z*-variable, will be written simply as X.

Instantiation of those various types of variables is defined by *basic specializations*, each of which has the form (v, w) where v specifies the name of the variable to be specialized and w the specializing value. For example, (`$N:tag1`, `$N:tag2`), (`$N:tag2`, `Name`) and (`$E:e`, (`$E:e1`, `$E:e2`)) are basic specializations which rename the *N*-variable `$N:tag1` to `$N:tag2`, instantiate the *N*-variable `$N:tag2` into the tagname `Name`, and expand the *E*-variable `$E:e` into the sequence of the *E*-variables `$E:e1` and `$E:e2`, respectively. There are four types of basic specializations:

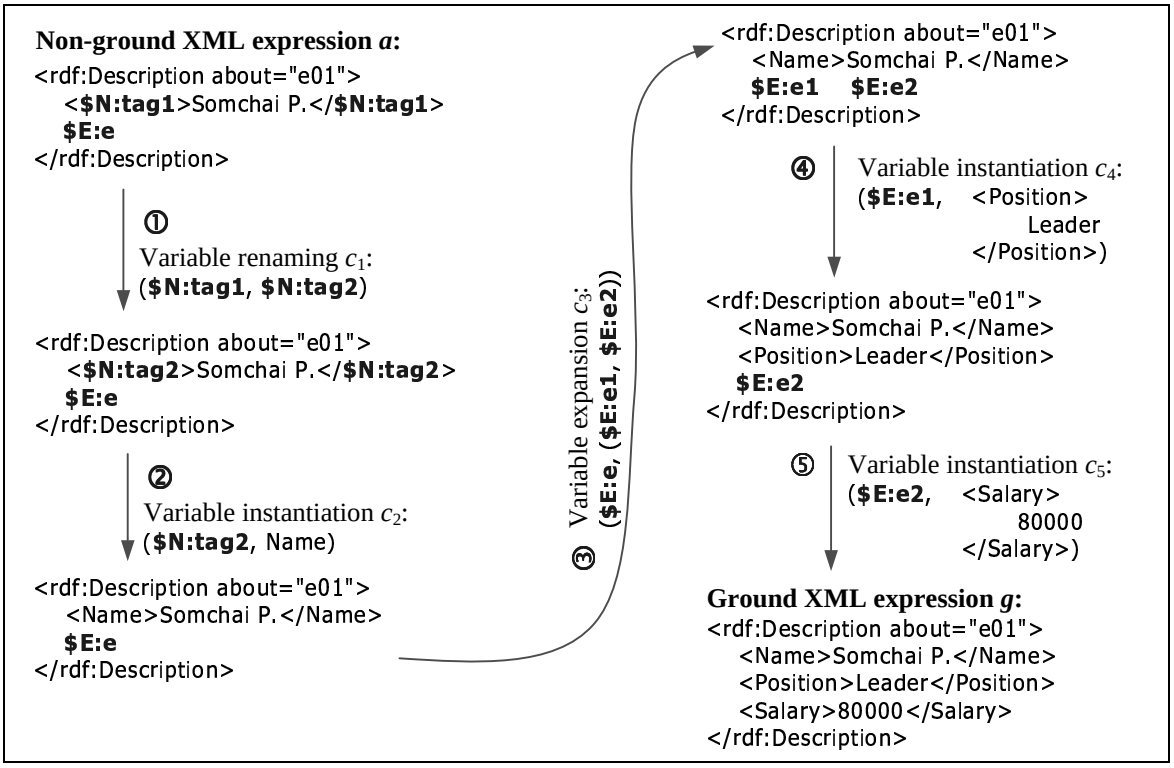


Figure 1: Successive applications of basic specializations c_1, …, c_5 to a non-ground XML expression a in $\mathcal{A}_X$ by the operator μ, yielding a ground XML expression g in $\mathcal{G}_X$.

1. Rename variables.
2. Expand a *P*- or an *E*-variable into a sequence of variables of their respective types.
3. Remove *P*-, *E*- or *I*-variables.
4. Instantiate variables to XML expressions or components of XML expressions which correspond to the types of the variables, i.e., instantiate:
   - *N*-variables to element types or attribute names,
   - *S*-variables to strings,
   - *E*-variables to XML expressions in $\mathcal{A}_X$,
   - *I*-variables to XML expressions which contains their sub-elements at an arbitrary depth, or
   - *Z*-variables to sets of XML expressions and *Z*-variables.

The *data structure* of XML expressions and sets of XML expressions are characterized by a mathematical abstraction, called **XML Specialization System**, which will be defined in terms of *XML specialization generation system* $\Delta = \langle \mathcal{A}, \mathcal{G}, \mathcal{C}, \nu \rangle$, where

- $\mathcal{C}$ is the set of all *basic specializations*, and
- ν is a mapping from $\mathcal{C}$ to *partial_map*($\mathcal{A}$) (i.e., the set of all partial mappings on $\mathcal{A}$), called the *basic specialization operator*; it determines, for each basic specialization c in $\mathcal{C}$, the change of objects in $\mathcal{A}$ caused by c.

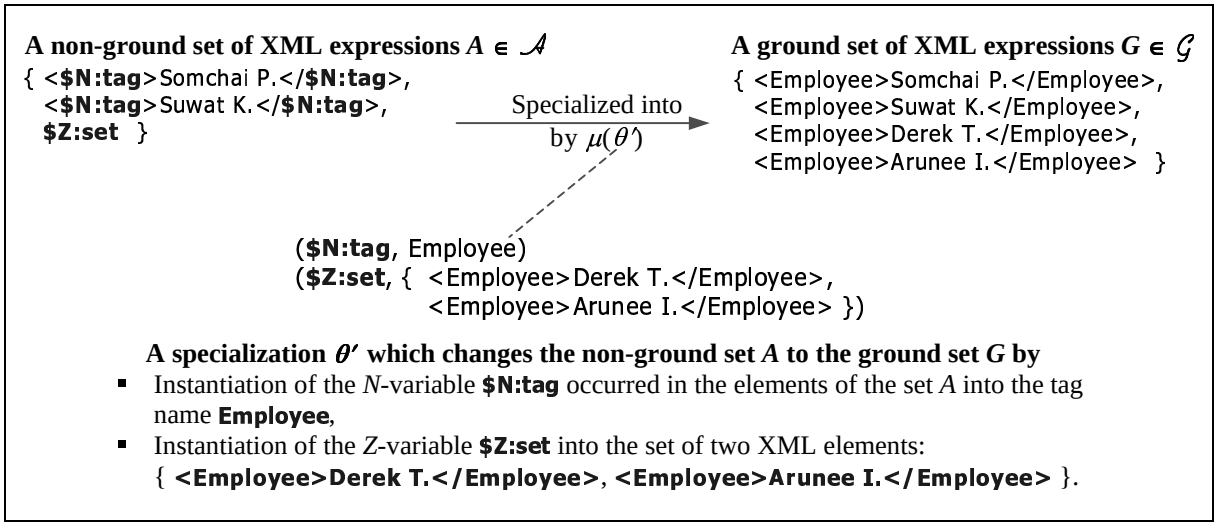


Figure 2: Specialization of a non-ground set of XML expressions in $\mathcal{A}$ into a ground set of XML expressions in $\mathcal{G}$ by the operator μ using a specialization θ' in $\mathcal{S}$.

Figure 1 illustrates examples of a non-ground XML expression a in $\mathcal{A}$, basic specializations c_1, …, c_5 in $\mathcal{C}$ and their successive applications to a by the operator ν in order to obtain a ground XML expression g in $\mathcal{G}$.

Denote a sequence of zero or more basic specializations in $\mathcal{C}$ by a *specialization.*

Based on the XML specialization generation system $\Delta = \langle \mathcal{A}, \mathcal{G}, \mathcal{C}, \nu \rangle$, the **XML Specialization System** is $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$, where

- $\mathcal{S} = \mathcal{C}^*$ is the set of all *specializations*, and
- $\mu : \mathcal{S} \rightarrow partial\_map(\mathcal{A})$ is the *specialization operator* which determines, for each specialization s in $\mathcal{S}$, the change of each object a in $\mathcal{A}$ caused by s such that:
  - $\mu(\lambda)(a) = a$, where λ denotes the null sequence,
  - $\mu(c \cdot s)(a) = \mu(s)(\nu(c)(a))$, where $c \in \mathcal{C}$ and $s \in \mathcal{S}$.

Intuitively, the operator μ is defined in terms of the operator ν such that for each $a \in \mathcal{A}$ and $s = (c_1 \ldots c_n) \in \mathcal{S}$, $\mu(s)(a)$ is obtained by successive applications of $\nu(c_1)$, …, $\nu(c_n)$ to a. Note that, when μ is clear from the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$.

With reference to Figure 1, let a specialization θ in $\mathcal{S}$ denote the sequence of the basic specializations c_1, c_2, c_3, c_4 and c_5; by the definition of μ, $g = \mu(\theta)(a) = a\theta$. Similarly, Figure 2 shows examples of a non-ground set of XML expressions A in $\mathcal{A}$, a specialization θ' in $\mathcal{S}$ and its application to A by the operator μ, in order to obtain a ground set of XML expressions G in $\mathcal{G}$, i.e., $G = \mu(\theta')(A) = A\theta'$.

2.2 XDD: Syntax

The definitions of *XML declarative descriptions with references* and its related concepts are given next in terms of the XML specialization system Γ.

Table 2: Definitions of the concepts *constraints*, *references* and *XML clauses* on Γ.

Concept	Being in Ground Form IFF	Application of a Specialization $\theta \in \mathcal{S}$ Yielding
A constraint: $q(a_1, \ldots, a_n)$ where $n > 0$, $q \in K$ and $a_i \in \mathcal{A}$	$a_i \in \mathcal{G}$ for $1 \le i \le n$	$q(a_1, \ldots, a_n)\theta = q(a_1\theta, \ldots, a_n\theta)$
A reference: $r = \langle a, f, P \rangle$ where - $a \in \mathcal{A}$, - $f \in F$ and - P is an XML declarative description which will be called the *referred description* of r	$a \in \mathcal{G}$	$r\theta = \langle a, f, P \rangle\theta = \langle a\theta, f\theta, P \rangle$
An XML clause: $H \leftarrow B_1, B_2, ..., B_n$ where - $n \ge 0$, - H is an XML expression in $\mathcal{A}_X$, - B_i is an XML expression in $\mathcal{A}_X$, a *constraint* or a *reference* on Γ, and - the order of B_i is immaterial.	Comprising only ground objects, ground constraints and ground references	$H\theta \leftarrow B_1\theta, B_2\theta, ..., B_n\theta$

Let K be a set of *constraint predicates* and F the set of all mappings: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$, the elements of which are called *reference functions*. An *XML declarative description* on Γ, simply called an *XDD description*, is a (possibly infinite) set of *XML clauses* on Γ. Table 2 defines concepts of *constraints*, *references* and *XML clauses* on Γ.

The notion of constraints introduced here is useful for defining restrictions on objects in $\mathcal{A}$, i.e., both on XML expressions in $\mathcal{A}_X$ and on sets of XML expressions in $2^{(\mathcal{A}_X \cup V_Z)}$. Given a *ground constraint* $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}$, its truth or falsity is assumed to be predetermined. Denote the set of all true ground constraints by *Tcon*. For instance:

- Define GT(a_1, a_2) as a constraint which will be true iff a_1 and a_2 are XML elements of the forms <Num>v_1</Num> and <Num>v_2</Num>, respectively, where v_1, v_2 are numbers and $v_1 > v_2$. Obviously, a constraint GT(<Num>10</Num>, <Num>5</Num>) is a true ground constraint in *Tcon.*
- Define a constraint Count(G, g) which will be true, iff G is a set of XML elements and g the XML element <Result>v</Result>, where v denotes G's cardinality.

The concept of references defined here together with an appropriate definition of a *set-of-reference function* in F will be employed to describe complex queries/operations on sets of XML expressions such as set construction, set manipulation and aggregate functions, e.g., min, max and count in SQL. Given $a, x \in \mathcal{A}_X$, let $f_{x,a} \in F$ denote a *set-of-reference function* and be defined as follows: For each $G \subset \mathcal{G}_X$,

$$f_{x,a}(G) = \{\{x\theta \in \mathcal{G}_X \mid \theta \in \mathcal{S},\ a\theta \in G\}\}. \quad (1)$$

In other words, for each subset G of $\mathcal{G}_X$, $f_{x,a}(G)$ is a singleton set, the element of which is a set of ground XML expressions of the form $x\theta$, for any specialization $\theta \in \mathcal{S}$, which makes $a\theta$ become a ground XML expression in G. Intuitively, a and x are used to define the condition for constructing a set and to determine the elements comprising that set,

respectively, i.e., $x\theta \in f_{x,a}(G)$ iff $a\theta \in G$. The objects a and x will be referred to as *filter* and *constructor objects*, respectively. Given a specialization θ in $\mathcal{S}$, application of θ to $f_{x,a}$ yields $f_{x,a}\theta = f_{x\theta, a\theta}$. For example, assuming that G is the set

```
{  <Employee division="IT">Somchai</Employee>,
   <Employee division="HR">Malee</Employee>,
   <Employee division="IT">Suwat</Employee>    },
```

then

$f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}(G)$

```
        =  { {<ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }.
```

In other words, such a set-of function filters the XML elements of the set G with the pattern

```
<Employee division="IT">$S:n</Employee>    — the filter object
```

and then constructs the resulting set of XML elements using the pattern

```
<ITstaff name=$S:n/>                       — the constructor object
```

Note that the effect of the binding of the variable `$S:n` in the filter object will also cascade to the constructor object.

Based on the definition of the set-of function, a reference $r = \langle S, f_{x,a}, P\rangle$, for $x, a \in \mathcal{A}_X$ and $S \in 2^{(\mathcal{A}_X \cup V)}$, is called a *set-of reference*.

Given an XML clause $C = (H \leftarrow B_1, B_2, ..., B_n)$, H is called the *head* and $(B_1, B_2, ..., B_n)$ the *body* of C, denoted by *head*(C) and *body*(C), respectively. The sets of all XML expressions, constraints and references in the body of C are denoted by *object*(C), *con*(C) and *ref*(C), respectively. Thus, *body*(C) = *object*(C) ∪ *con*(C) ∪ *ref*(C). If $n = 0$, such a clause is called a *unit clause*, if $n > 0$, a *non-unit clause*. When it is clear from the context, a unit clause ($H \leftarrow$) is written simply as H, i.e., the left-arrow symbol is omitted. Therefore, every XML element can be considered as a ground XML unit clause, and moreover every XML document can be modeled as an XDD description comprising solely ground XML unit clauses.

The *heights* of an XML clause C and of an XDD description P, denoted by *hgt*(C) and *hgt*(P), are defined as follows:

- If *ref*(C) = ∅ (C contains no reference), then *hgt*(C) = 0;
  Otherwise *hgt*(C) is the maximum height of all the referred descriptions contained in its body plus one.
- *hgt*(P) is the maximum height of all the clauses in P.

2.3 XDD: Declarative Semantics

Given an XDD description P on Γ, its declarative semantics, denoted by $\mathcal{M}(P)$, is defined inductively as follows:

1. Given the meaning $\mathcal{M}(Q)$ of an XDD description Q with the height m, a reference $r = \langle g, f, Q\rangle$ is a true reference, iff $g \in f(\mathcal{M}(Q))$. For any $m \geq 0$, define *Tref*(m) as the set of all *true references*, the heights of the referred descriptions of which are smaller than or equal to m, i.e.:

$$Tref(m) = \{\langle g, f, Q\rangle \mid g \in \mathcal{G},\ f \in F,\ hgt(Q) \le m,\ g \in f(\mathcal{M}(Q))\} \quad (2)$$

2. The meaning $\mathcal{M}(P)$ of the description P is a set of XML elements defined by:

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} [T_P]^n(\varnothing) \quad (3)$$

where - $\varnothing$ is the empty set,
- $T_P{}^1(\varnothing) = T_P(\varnothing)$ and $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$ for each $n > 1$, and
- the mapping T_P: $2^{\mathcal{G}} \to 2^{\mathcal{G}}$ is:

For each $G \subset \mathcal{G}$, $g \in T_P(G)$ iff there exist a clause $C \in P$ and a specialization $\theta \in \mathcal{S}$ such that $C\theta$ is a ground clause, with head g and all objects, constraints and references in its body belong to G, $Tcon$ and $Tref(n)$, for some $n < hgt(P)$, respectively, i.e.:

$$T_P(G) = \{head(C\theta) \mid C \in P,\ \theta \in \mathcal{S},\ C\theta \text{ is a ground clause},\ object(C\theta) \subset G,\ con(C\theta) \subset Tcon,\ ref(C\theta) \subset Tref(n),\ n < hgt(P)\} \quad (4)$$

Intuitively, the meaning of a description P, i.e., $\mathcal{M}(P)$, is a set of all XML elements, which are directly described by and derivable from the unit and the non-unit clauses in P, respectively, i.e.:

- Given a unit clause ($H \leftarrow$) in P, for $\theta \in \mathcal{S}$:

  $H\theta \in \mathcal{M}(P)$ if $H\theta$ is an XML element.

- Given a non-unit clause ($H \leftarrow B_1, ..., B_i, B_{i+1}, ..., B_j, B_{j+1}, \ldots, B_n$) in P, assuming without loss of generality that $B_1, ..., B_i$ are XML expressions, $B_{i+1}, ..., B_j$ are constraints, and $B_{j+1}, ..., B_n$ are references, for $\theta \in \mathcal{S}$:

  $H\theta \in \mathcal{M}(P)$ if - $H\theta$ is an XML element,
  - $B_1\theta, ..., B_i\theta \in \mathcal{M}(P)$,
  - $B_{i+1}\theta, ..., B_j\theta$ are true constraints, and
  - $B_{j+1}\theta, ..., B_n\theta$ are true references.

Based on the formalized concepts of XDD language, Figure 3 demonstrates two XDD descriptions denoted by Q and P, and then determines their semantics, which is sets of XML elements denoting certain objects and their relationships in a real-world domain.

3 Modeling Semantic Web Resources and Applications

XDD language allows collections of Semantic Web resources, such as documents, data, metadata and ontologies, encoded in XML, RDF, OIL or DAML+OIL syntax, to be represented in terms of XDD descriptions. In the descriptions, explicit information items are directly expressed as ground XML unit clauses, while rules, conditional relationships, integrity constraints and ontological axioms are formalized as XML non-unit clauses. The descriptions' semantics, which can be directly determined under the language itself, is defined as sets of XML elements—surrogates of objects and their relationships in a real-world domain.

{ C_1: <Employee division="IT">Somchai</Employee> ←
C_2: <Employee division="HR">Malinee</Employee> ←
C_3: <Employee division="IT">Suwat</Employee> ← }

(a) XML Declarative Description *Q*.

- $hgt(C_1) = hgt(C_2) = hgt(C_3) = hgt(Q) = 0$
- $\mathcal{M}(Q)$ = { <Employee division="IT">Somchai</Employee>
  <Employee division="HR">Malee</Employee>
  <Employee division="IT">Suwat</Employee> }

(b) The height and the meaning of *Q*.

{ C_4: <ITStaffNumber>$S:num</ITStaffNumber>
← ⟨$Z:set, $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}$, Q⟩,
Count($Z:set, <Result>$S:num</Result>). }

(c) XML Declarative Description *P*.

- $hgt(C_4) = hgt(Q) + 1 = 1$, and $hgt(P) = 1$.
- $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}(\mathcal{M}(Q))$ =
  { { <ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }
- ⟨$Z:set, $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}$, Q⟩
  is a true reference, iff $Z:set is specialized into the set
  { { <ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }
- The constraint Count is a true constraint, iff $S:num is specialized into the number 2.
- $\mathcal{M}(P)$ = {<ITStaffNumber>2</ITStaffNumber>}.

(d) The height and the meaning of *P*.

Figure 3:XDD descriptions *Q* and *P* and their declarative semantics.

Besides its employment to model various kinds of Semantic Web resources, XDD can also be applied to describe certain operations such as queries, document transformation and processing rules as well as program specifications. Figure 4 illustrates examples of Semantic Web resource modeling and a query formulation and shows that although information about senior employees and their bonuses is not explicit, it can be deductively inferred based on the clause R_3; this representation offers a more compact form and provides an easier way of information modification than explicit enumeration of such information. With this simple, yet expressive modeling mechanism, XDD can readily be applied to model Semantic Web applications.

A Semantic Web application, offering certain Semantic Web services, comprises three main components: *application data*, *application rules or logic*, and *users' queries or requests for services*. For instance: In a Semantic Web search engine, offering an information-gathering service,

- its application data: a catalog or descriptions of Semantic Web contents,
- its application rules: domain-model ontologies and axioms, and
- its requests: user queries describing their informational needs.

In a business-2-business (B2B) commerce application, its three components are

- a catalog of available products and services,
- business rules and policies such as price discounting and refund rules, and
- queries and business transactions such as a request for a quotation and an order placement.

XDD language provides means for modeling Semantic Web applications in that it enables direct representation of:

- application data (facts), encoded in XML, RDF, OIL or DAML+OIL syntax, in terms of XML ground unit clauses,
- application rules or logic in terms of XML non-unit clauses—the heads and bodies of the clauses describe the consequences and antecedents of the rules, respectively—and
- users' queries or service requests in terms of XML non-unit clauses—the heads of the clauses describe the structure of the query results or the service responses and the bodies specify the queries' selection conditions or the service requests and their constraints.

Thus, XDD language has the three vital roles:

- *content language,*
- *application-rule language,* and
- *query* or *service-request language*.

See Figure 4 for an example of each role. Basically, each query/request will be executed on a specified collection of application data and rules and will return as its answer a set of XML elements, derivable from such a collection and satisfying all of its conditions. More precisely, given a set of application data and rules, modeled as an XDD description P, and a query/request, formulated as an XML clause Q: $(H \leftarrow B_1, B_2, ..., B_n)$, the response to Q is the set

$$\{H\theta \mid H\theta \in \mathcal{M}(P \cup \{Q\}), \theta \in \mathcal{S}\}.$$

By employment of *Equivalent Transformation* (*ET*) computational paradigm [2,3], which is based on semantics-preserving transformations (*equivalent transformations*) of declarative descriptions, the *computation* of an answer/response to such a query/request Q is carried out by successive transformations of the XDD description $P \cup \{Q\}$ into a simpler but equivalent description, from which the answer can be obtained readily and directly. In brief, $P \cup \{Q\}$ will be successively transformed until it becomes the description

$$P \cup \{Q_1, \ldots, Q_n\},$$

where $n \geq 0$ and the Q_i are ground XML unit clauses.

Note that in order to guarantee correctness of a computation, only equivalent transformations are applied at every step. The unfolding transformation, a widely-used program transformation in conventional logic programming, is a kind of equivalent transformation. Other kinds of equivalent transformations can also be devised, especially for improvement of computation efficiency. Thus, ET provides a more flexible, efficient computational framework.

XET, a declarative programming language for computation of XDD descriptions in ET paradigm, will be presented next.

E_1:
```
<rdf:Description about="e01">
    <rdf:type resource="Employee"/>
    <e:Name>Somchai P.</e:Name>
    <e:Salary>80000</e:Salary>
</rdf:Description>
```

E_2:
```
<rdf:Description about="e02">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e01"/>
    <e:Name>Sawat K.</e:Name>
    <e:Salary>50000</e:Salary>
</rdf:Description>
```

E_3:
```
<rdf:Description about="e03">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e02"/>
    <e:Name>Derek T.</e:Name>
    <e:Salary>40000</e:Salary>
</rdf:Description>
```

E_4:
```
<rdf:Description about="e04">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e02"/>
    <e:Name>Arunee I.</e:Name>
    <e:Salary>30000</e:Salary>
</rdf:Description>
```

(a) Modeling of application data – descriptions of employee objects, based on RDF infrastructure.

R_1:
```
<EmpRelation boss=$S:X subordinate=$S:Y level=1/>
    ←  <rdf:Description about=$S:Y>
           <rdf:type resource="Employee"/>
           <e:Boss resource=$S:X/>
           $E:emp
       </rdf:Description>.
```
% If Y is described as a resource of the type Employee and its Boss property is another resource X, then one can derive that X is the first-leveled boss of Y.

R_2:
```
<EmpRelation boss=$S:X subordinate=$S:Z level=$S:n1/>
    ←  <EmpRelation boss=$S:X subordinate=$S:Y level=$S:n/>,
       <rdf:Description about=$S:Z>
           <rdf:type resource="Employee"/>
           <e:Boss resource=$S:Y/>  $E:emp
       </rdf:Description>,
       Add( <Num>$S:n</Num>, <Addendum>1</Addendum>,
           <Result>$S:n1</Result>).
```
% If X is the nth-leveled boss of Y and Y is referred to as a direct boss of an Employee Z, then one can imply that X is the (n+1)th-leveled boss of Z.

R_3:
```
<rdf:Description about=$S:X>
    <rdf:type resource="SeniorEmployee"/>
    <e:Bonus>$S:bonus</e:Bonus>
    <e:Subordinate>
        <rdf:Bag>$Z:set</rdf:Bag>
    </e:Subordinate>
    <e:Salary>$S:sal</e:Salary>  $E:emp
</rdf:Description>
  ←  <rdf:Description about=$S:X>
         <rdf:type resource="Employee"/>
         <e:Salary>$S:sal</e:Salary>  $E:emp
     </rdf:Description>,
     ⟨$Z:set, f<rdf:Li resource=$S:Y/>,<EmpRelation boss=$S:X subordinate=$S:Y level=$S:any/>,
      {E1,..., E4, R1, R2}⟩,
     Count($Z:set, <Result>$S:num</Result>),
     GT(<Num>$S:num</Num>, <Num>3</Num>),
     Mul(<Num>$S:sal</Num>, <Multiplier>2</Multiplier>,
         <Result>$S:bonus</Result>).
```
% For an employee X who has more than 3 subordinates (of any level), then X is considered to be a Senior-Employee and will receive a double-salary bonus. A list of all subordinates of X is also included in X's description.

(b) Modeling of application rules and logic – descriptions of relationships among employee objects.

Q:
```
<Answer>$S:name</Answer>
    ←  <rdf:Description about=$S:X>
           <rdf:type resource="Employee"/>
           <e:Name>Somchai P.</e:Name>  $E:emp1
       </rdf:Description>,
       <EmpRelation boss=$S:X subordinate=$S:Y level=2/>,
       <rdf:Description about=$S:Y>
           <rdf:type resource="Employee"/>
           <e:Name>$S:name</e:Name>  $E:emp2
       </rdf:Description>.
```
% A query Q finds names of all Somchai P.'s second-leveled subordinates.

(c) Modeling of a query.

Figure 4: Modeling of an application.

4 XET Programming Language

XET (*XML Equivalent Transformation*) [18] is a declarative programming language which can directly and succinctly manipulates XML data. By integration of XDD language, ET computational paradigm and XML syntax, XET possesses XDD's expressiveness and ET's computational power as well as XML's flexibility, extensibility and interchangeability. Therefore, XET naturally unifies "Documents', "Programs" and "Data", and with its computational and reasoning services, it also unifies "Document Processing (Transformation)", "Program Execution" and "Query processing". Available XML editing and validating tools can also be employed to edit and validate XET programs. The syntax of XET language, described by XML Schema, is available in [18]. XET provides useful sets of built-in operations including:

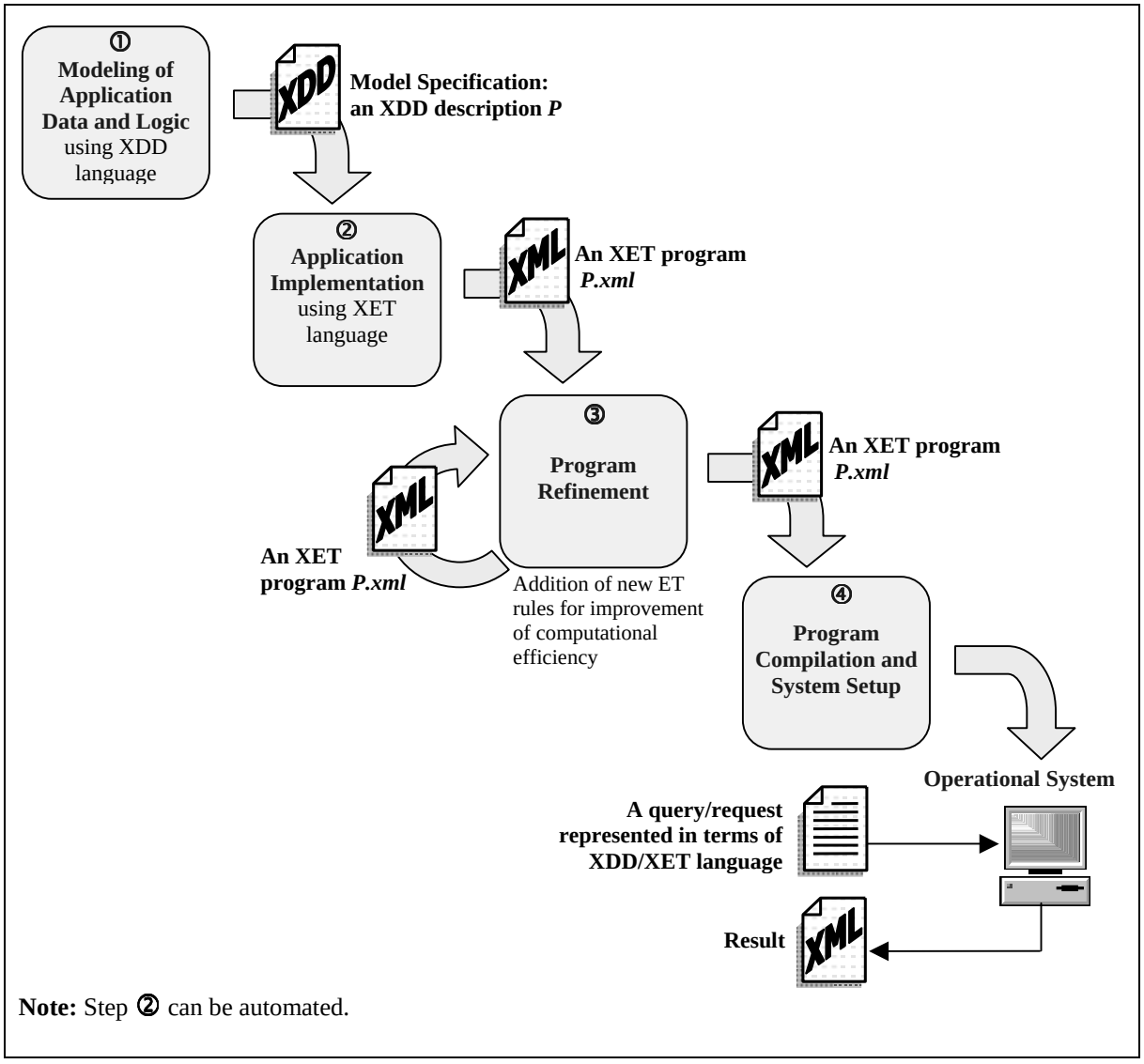


Figure 5: A declarative approach to Semantic Web application development.

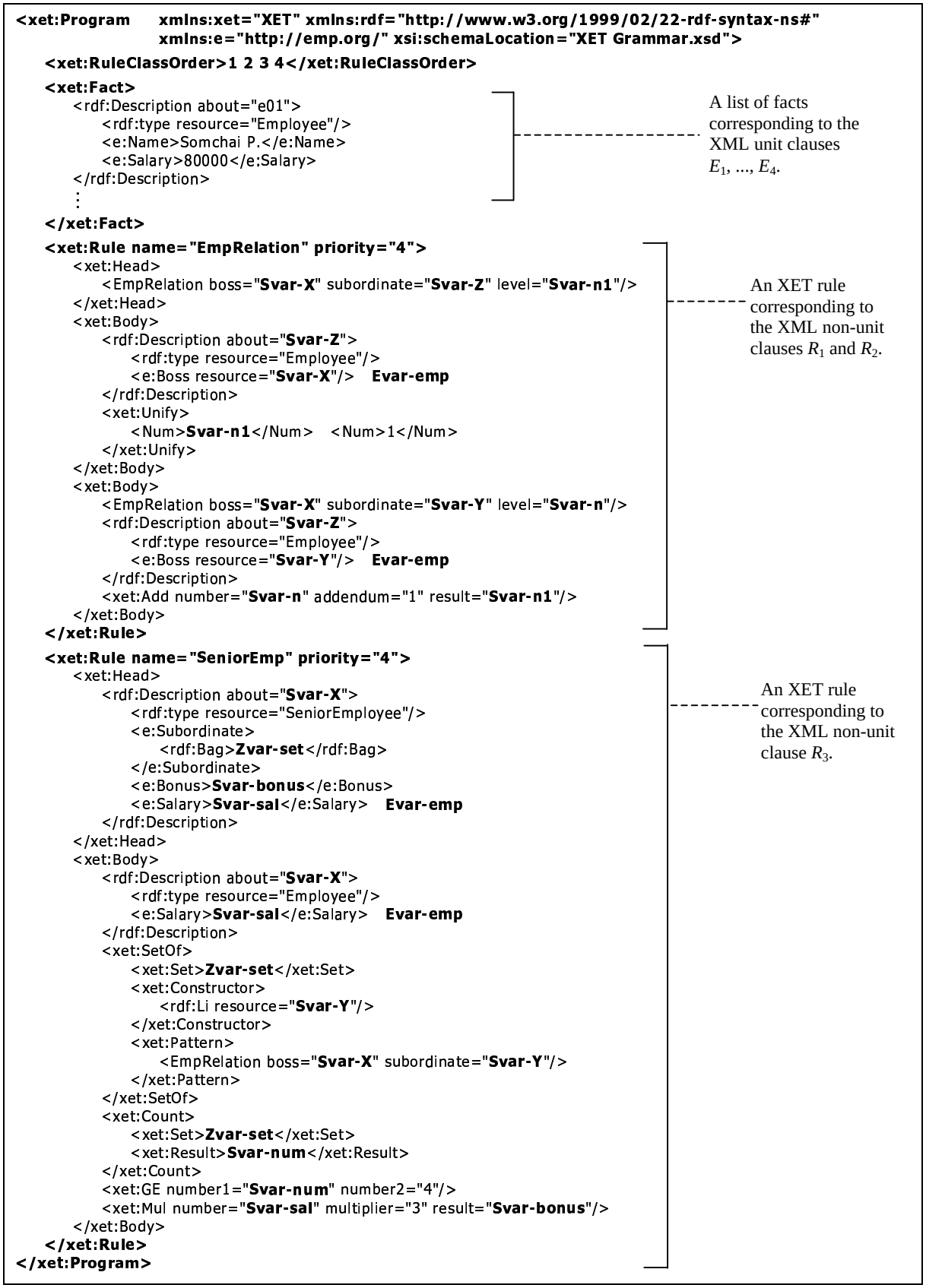
- Data type checking
- Document well-formedness and validity checking
- Arithmetic operations and relations
- String manipulation operations
- XML expression unification and matching
- XML expression operations
- Input/Output operations
- File manipulation operations
- Communication services

Once an XDD description which models a Semantic Web application's data and logic has been formulated, an XET program corresponding to such a description can be obtained directly. The obtained XET program can be iteratively refined, if new ET rules have been devised for improvement of computational efficiency. Finally, using *XET compiler* [18], the program is compiled and an operational system obtained. In response to a query/request submitted in terms of XDD or XET language, the system executes it and returns its result. Based on such processes, Figure 5 outlines a new declarative approach to Semantic Web application development. With reference to the XDD description and the query *Q* of Figure 4, an XET program corresponding to such a description is given by Figure 6 and the answer to the query *Q* by Figure 7.

Note that the declarative semantics of an XET program can be determined based on that of its respective XDD description.

Figure 8 depicts an example scenario of Semantic Web service execution, which starts when a user or an application *A* issues a query describing a service need together with constraints and preferences to a Semantic-Web-Service Search Engine *B*, which will then searches, from its database of Web services, for Semantic Web applications offering the demanded services with the requested properties and restrictions. Based on the returned list of applications, *A* selects an appropriate one, say Semantic Web application *C*, and sends it a query or a service-request as well as user constraints and preferences. *C* then executes such a query or request with respect to its data and application rules and logic. During its execution, *C* may forward corresponding sub-queries and/or sub-requests to other related applications based on its defined rules and logic and wait for their replies and responses. Once the execution has been finished, a reply to *A*'s query/request is returned. Note that Steps 1 and 2 can be skipped, if the user/application *A* knows at a priori which application provides the desired service. Similarly, at Steps 5.1 and 5.3 if the application *C* does not know which application it should interoperate, it may ask *B* for a list of applications offering the required services. In addition, during the execution, it is often a case that communicating parties may involve in a negotiation for modification of service conditions.

From the example scenario, one may observe that XDD and XET can serve as a tool for modeling and implementing a wide diversity of Semantic Web applications offering various kinds of services. Consider, for instance, the Semantic-Web-Service Search Engine *B*, which maintains a database of Web services described by means of Web-service metadata and provides a search facility for finding of particular services satisfying some specified criteria. Such a search engine is simply modeled as an XDD description comprising XML unit clauses, describing a collection of registered services and their properties/capabilities (in terms of Web-service metadata), and XML non-unit clauses, modeling Web-service ontologies as well as implicit relations among services in the collection. From such a description, an XET program which is capable of searching for services with desired properties and constraints can be obtained directly. Other applications *C*, *D* and *E* serving certain specific services can also be modeled and implemented in a similar manner.




```
<xet:Program    xmlns:xet="XET" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:e="http://emp.org/" xsi:schemaLocation="XET Grammar.xsd">
   <xet:RuleClassOrder>1 2 3 4</xet:RuleClassOrder>
   <xet:Fact>
      <rdf:Description about="e01">
         <rdf:type resource="Employee"/>
         <e:Name>Somchai P.</e:Name>
         <e:Salary>80000</e:Salary>
      </rdf:Description>
      ⋮
   </xet:Fact>
   <xet:Rule name="EmpRelation" priority="4">
      <xet:Head>
         <EmpRelation boss="Svar-X" subordinate="Svar-Z" level="Svar-n1"/>
      </xet:Head>
      <xet:Body>
         <rdf:Description about="Svar-Z">
            <rdf:type resource="Employee"/>
            <e:Boss resource="Svar-X"/>   Evar-emp
         </rdf:Description>
         <xet:Unify>
            <Num>Svar-n1</Num>   <Num>1</Num>
         </xet:Unify>
      </xet:Body>
      <xet:Body>
         <EmpRelation boss="Svar-X" subordinate="Svar-Y" level="Svar-n"/>
         <rdf:Description about="Svar-Z">
            <rdf:type resource="Employee"/>
            <e:Boss resource="Svar-Y"/>   Evar-emp
         </rdf:Description>
         <xet:Add number="Svar-n" addendum="1" result="Svar-n1"/>
      </xet:Body>
   </xet:Rule>
   <xet:Rule name="SeniorEmp" priority="4">
      <xet:Head>
         <rdf:Description about="Svar-X">
            <rdf:type resource="SeniorEmployee"/>
            <e:Subordinate>
               <rdf:Bag>Zvar-set</rdf:Bag>
            </e:Subordinate>
            <e:Bonus>Svar-bonus</e:Bonus>
            <e:Salary>Svar-sal</e:Salary>   Evar-emp
         </rdf:Description>
      </xet:Head>
      <xet:Body>
         <rdf:Description about="Svar-X">
            <rdf:type resource="Employee"/>
            <e:Salary>Svar-sal</e:Salary>   Evar-emp
         </rdf:Description>
         <xet:SetOf>
            <xet:Set>Zvar-set</xet:Set>
            <xet:Constructor>
               <rdf:Li resource="Svar-Y"/>
            </xet:Constructor>
            <xet:Pattern>
               <EmpRelation boss="Svar-X" subordinate="Svar-Y"/>
            </xet:Pattern>
         </xet:SetOf>
         <xet:Count>
            <xet:Set>Zvar-set</xet:Set>
            <xet:Result>Svar-num</xet:Result>
         </xet:Count>
         <xet:GE number1="Svar-num" number2="4"/>
         <xet:Mul number="Svar-sal" multiplier="3" result="Svar-bonus"/>
      </xet:Body>
   </xet:Rule>
</xet:Program>
```

Figure 6: An XET program P.xml.

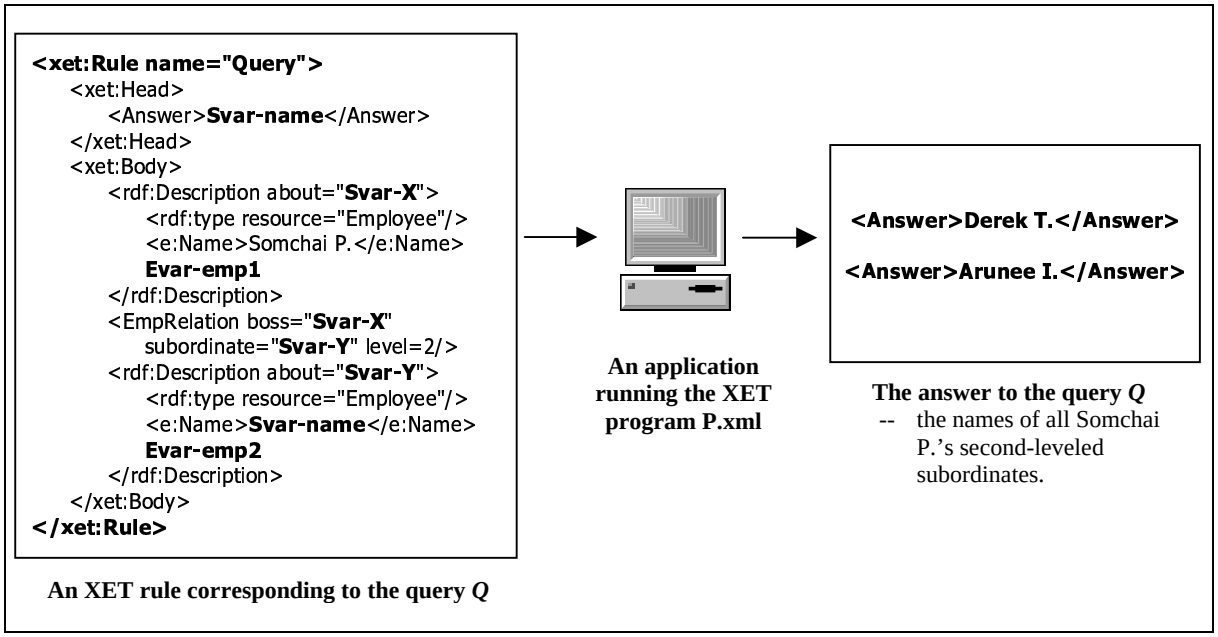


Figure 7: The answer to the query *Q*.

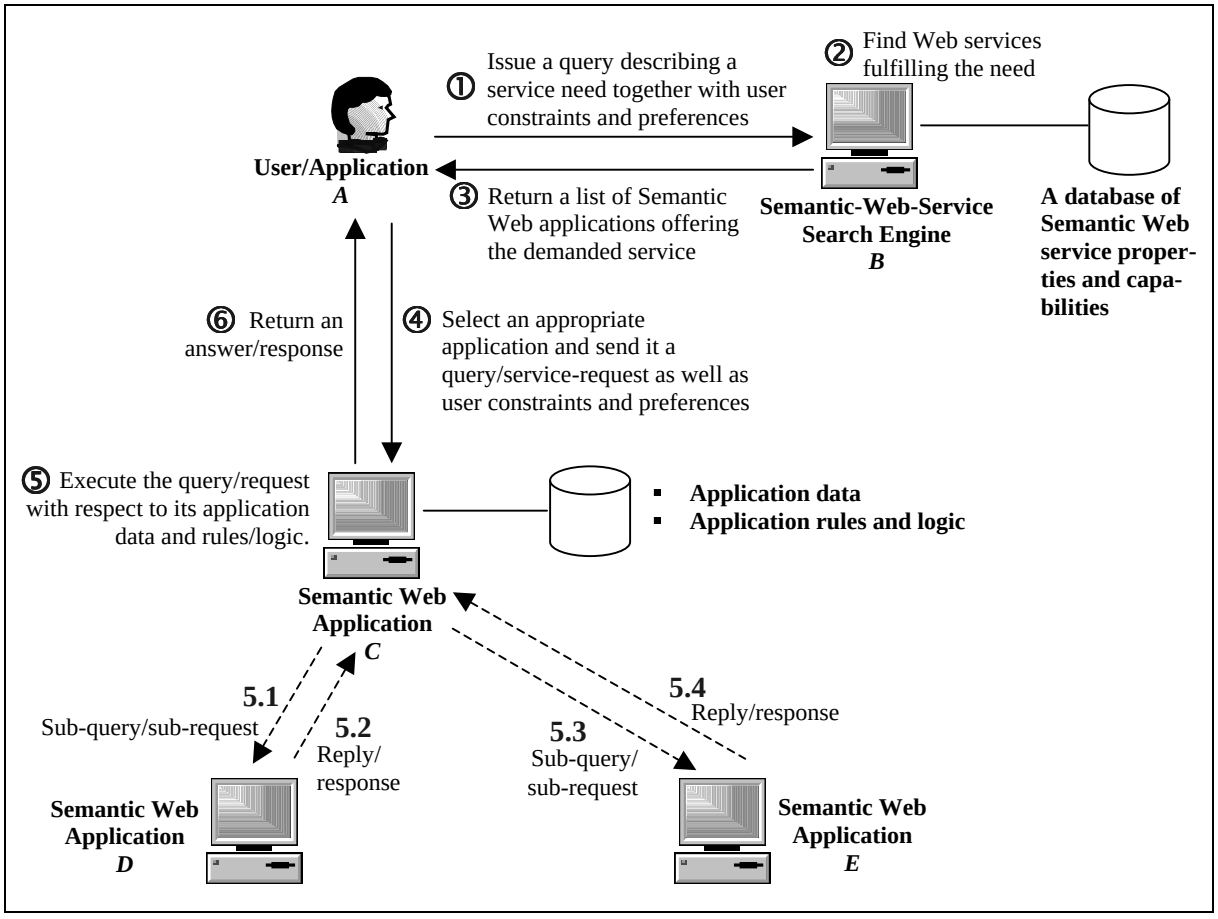


Figure 8: A scenario for Semantic Web service execution.

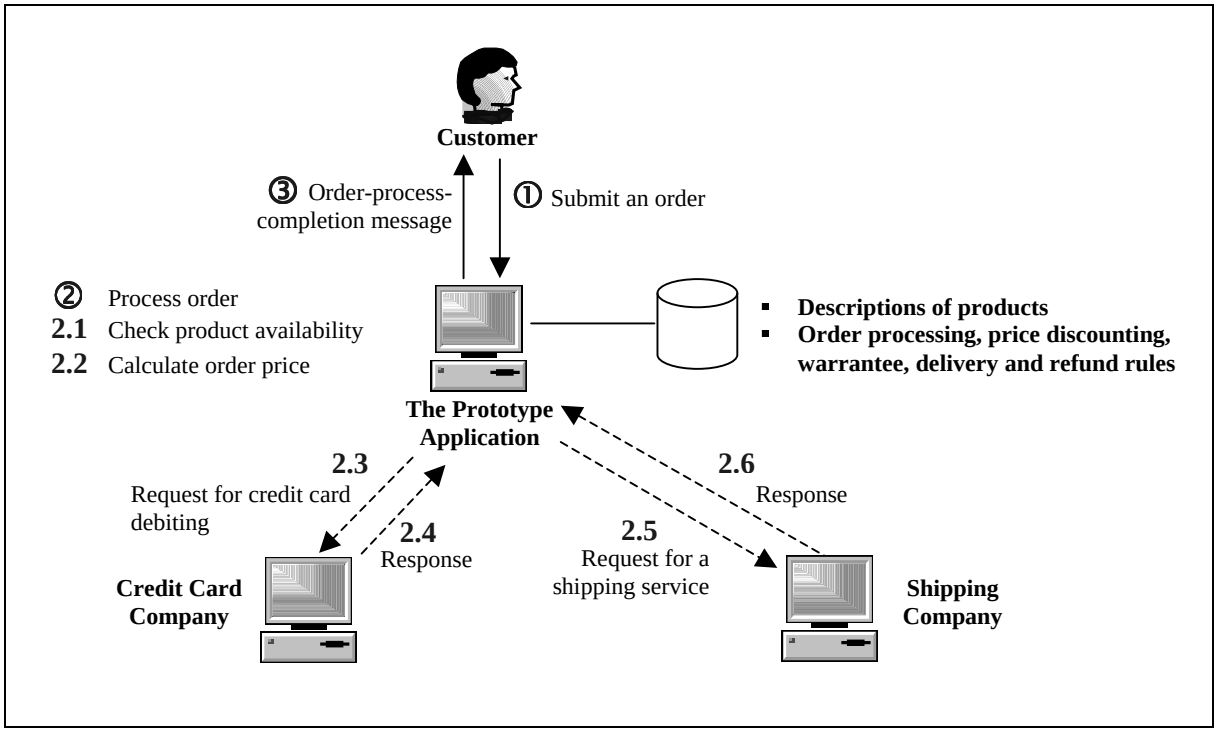


Figure 9: Prototype Semantic Web application.

5 Prototype Application

Founded on the proposed framework, a prototype Semantic Web application, which provides product-information-gathering as well as e-shopping services, has been implemented by means of XDD modeling language and XET programming language. The employed XDD and XET languages have been enhanced with the ability to handle rule conflicting problems using rule prioritization information [11]. Such an additional feature enables, for example, a formalization of a discounting policy stating that if a customer is a member of the store, a 10 percent discount is offered, and if a customer has a late-payment history, no discount is offered and that the latter has higher priority than the former [11]. Due to space limitation, declarative semantics of *prioritized XDD descriptions and XET programs* is omitted; its formal definition is available in [18].

To buy some products (Figure 9), a customer may fill out an order form and submit it on-line to the application Web site or directly send an HTTP request with appropriate parameters encoded in XML to the application URL. The application first checks its stock, and if the products are available, it will calculate the order's price, send a request to a credit card company (another Semantic Web application) for a debit of the customer's account, send a request to a shipping company and then wait for their responses. After the order process has been finished, the application notifies the customer of the completed process.

6 Related Works

Business Rule Markup Language (*BRML*) [11] is an XML-based language for encoding of *Courteous Logic Programs* (*CLP*)—an extension of conventional logic programs by inclusion of the ability to express prioritized conflict handling. BRML is a language in the RuleML Initiative, which has been specifically designed to represent business rules and policies. However, since BRML provides merely an XML embodiment of CLPs, its expressive power is relatively limited in that its sole permitted representation is atomic formulae with simple-structure terms. Complex XML data with nesting structures cannot be directly represented in BRML. Instead they require appropriate translations into corresponding sets of atomic formulae. Figure 10, for example, shows the CLP's and BRML's representations of the XML element E_1 and the XML clause R_1 of Figure 4. Comparing XDD with BRML, one can readily observe that XDD provides a more direct and succinct modeling mechanism; While possessing sufficient expressive power to represent simple as well as complex statements and relations, its representation is still readable.

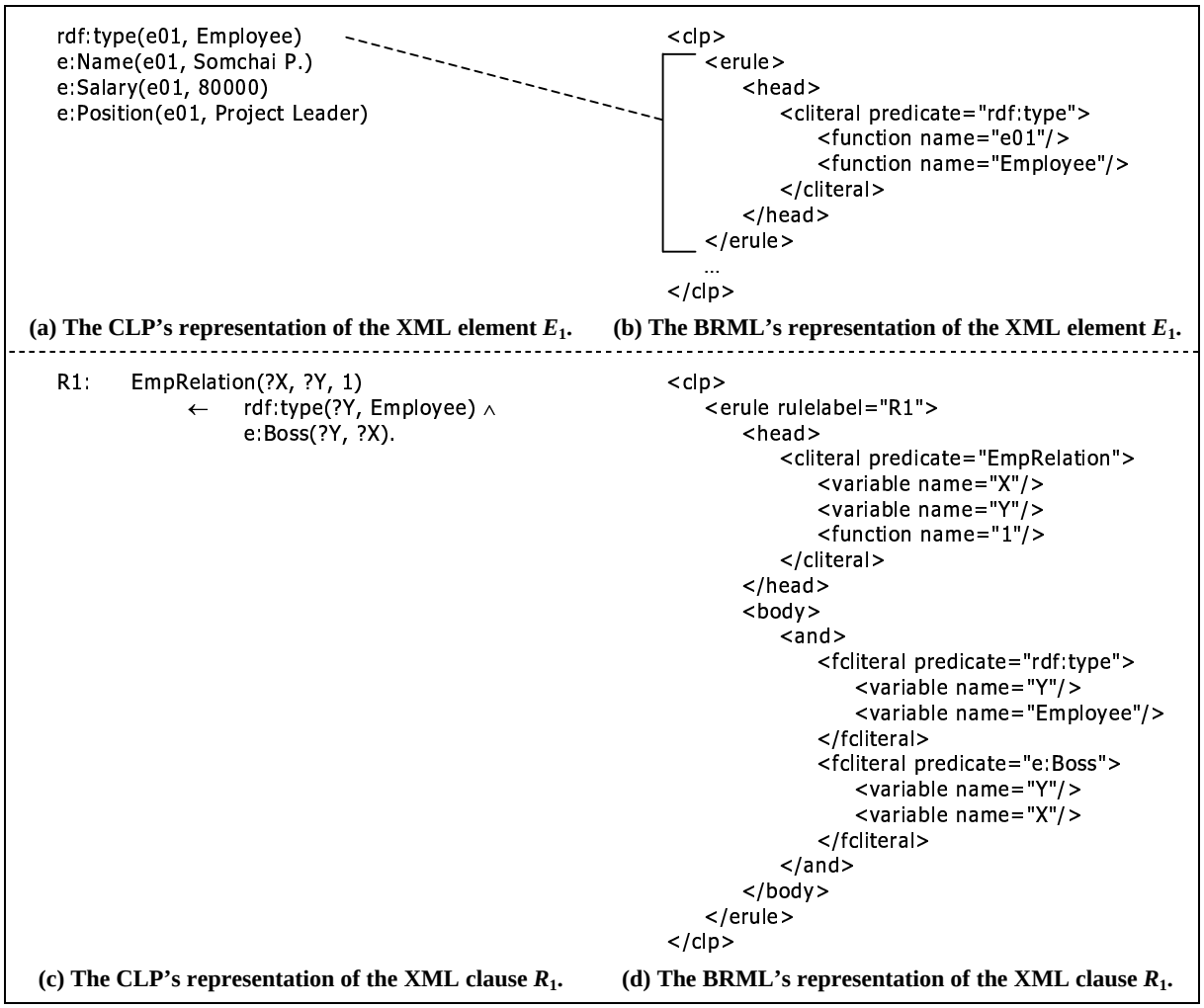


Figure 10: The CLP's and BRML's representations of the element E_1 and the clause R_1 of Figure 4.

OIL [6,8,12] and *DAML+OIL* [13] are the two most recent, improved ontology-based semantic markup languages for Web resources which extends RDF Schema by richer sets of modeling primitives. However, their current versions still lack expressive power in that arbitrary rules and axioms cannot be described [6]. Since these languages' schemas and instances, which are encoded in RDF/XML serialization, can be directly represented in XDD as XML unit clauses, XDD can be employed to serve as their foundation, in order to help enhance their expressiveness [17]. Therefore, resources and applications modeled by these languages become immediately instances of XDD language and hence directly programmable by XET language. Note that with the awareness of the DAML+OIL's limitation in representing rules and axioms, the language is being extended with the ability to express Horn clauses and will be called *DAML-L* [16].

With reference to the overall process of Semantic Web service implementation defined in [11,16], XDD can be uniformly employed to materialize each step of the process:

- *Service Advertisement and Discovery*: A collection of service properties and capabilities described by means of metadata or *Semantic Web Service Markup Language* [16] can be directly represented as XML unit clauses, and their additional constraints and relations modeled in terms of XML non-unit clauses. Based on such declarative advertisements of Web services, discovery of a particular service having specific properties and capabilities is expressed as an XML non-unit clause, which will be evaluated on the Web service database and return as its reply a list of applications or service providers offering the requested service.
- *Negotiation*: Given particular negotiation rules and procedures for selection of Web services (e.g., response time, data accuracy and cost conditions), corresponding XML non-unit clauses can be declaratively defined. Besides definition of such rules, an appropriate employment of *Agent Communication Language* (*ACL*) [15] which allows the negotiating parties to effectively communicate and interoperate must also be considered. By a careful formulation and implementation of the two major current ACLs (i.e., *KQML* [9] and *FIPA-ACL* [10,15]) in XDD [14], XDD readily provides an effective communication in the negotiation stage, allowing every negotiating party to communicate with one another via XDD uniform interface, and hence enabling a higher level of interoperability.
- *Service Execution*: Execution of a service according to a given procedure can be represented in XDD by appropriate XML clauses. Based on such a declarative specification, an application can automatically execute the service.
- *Service Composition and Integration*: It is often a case that a service is designed as a composition or an integration of other existing services. The execution of such a composite service often requires interaction with those related services in terms of request-for-service calls and returned responses. Using XDD, one can represent service composition and integration by an XML clause, the head of which specifies the composite service and the body of which describes the composition rule as well as the service calls and the data to be exchanged with other services. Such service calls and exchanged data could be embodied in an ACL.
- *Service Customization*: In order to increase the level of share-ability, reusability and user's satisfactory of provided services, a service may be defined by a particular generic procedure which can then be customized for execution of a specific service request. Such a generic procedure is described by a set of XML clauses, and its

customization is realized by either parameter instantiation or by addition of XML clauses into it—this latter case is equivalent to program refinement. Note that different customizations normally lead to different sequences and results of service execution.

In summary, with these supports, XDD readily provides sufficient Semantic Web modeling facilities for development of intelligent, automated Web applications requiring interoperation with other independently-developed applications.

7 Conclusions

The proposed XDD language is an expressive modeling language which allows collections of Semantic Web resources (modeled in terms of XML, RDF, OIL or DAM+OIL) to be directly represented with their semantics precisely and formally determined. In addition to such a resource modeling facility, XDD also provides a means for descriptions of Web resource manipulations, service provisions and business rules and processes. Moreover, its extension [18] by the ability to handle rule conflicting problems has enhanced its expressive power to be sufficient to capture and describe complex and conflicting rules and logic in Semantic Web applications.

Founded on XDD's expressive power and ET's computational efficiency, XET programming language and its compiler have also been developed. By means of XDD and XET languages, the paper has proposed a declarative framework for Semantic Web application development and has demonstrated that a variety of Semantic Web applications and services is simply expressible by XDD and hence programmable by XET. Moreover, the development of the prototype system based on the proposed framework has helped prove the framework's viability and potential in real applications. Integration of the proposed framework with appropriate Web and agent technologies allows intelligent as well as automated Web services, which demand syntactic and semantic interoperability, to be easily and rapidly developed. Note also that an XET program which performs particular tasks can be exchanged, shared and reused by multiple applications.

References

[1] K. Akama, Declarative Semantics of Logic Programs on Parameterized Representation Systems. Advances in Software Science and Technology **5** (1993) 45–63.

[2] K. Akama, Declarative Description with References and Equivalent Transformation of Negative References. Technical Report, Department of Information Engineering, Hokkaido University, Japan (1998).

[3] K. Akama, T. Shimitsu and E. Miyamoto, Solving Problems by Equivalent Transformation of Declarative Programs. Journal of Japanese Society of Artificial Intelligence (JSAI) **13(6)** (1998) 944–952 (in Japanese).

[4] K. Akama, C. Anutariya, V. Wuwongse and E. Nantajeewarawat, A Foundation for XML Databases: Query Formulation and Evaluation. Technical Report, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)

[5] C. Anutariya, V. Wuwongse, E. Nantajeewarawat and K. Akama. Towards a Foundation for XML Document Databases. Proc. 1st Int. Conference on Electronic Commerce and Web Technologies (EC-Web 2000), London, UK. Lecture Notes in Computer Science, Springer Verlag **1875** (2000) 324–333.

[6] S. Bechhofer *et al.*, An Informational Description of Standard OIL and Instance OIL. White Paper (Nov. 2000). Available at http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf

[7] T. Berners-Lee, Weaving the Web. Harpur, San Francisco (1999).
[8] S. Decker *et al.*, The Semantic Web: The Roles of XML and RDF, IEEE Internet Computing, (Sep./Oct. 2000) 63–74.
[9] T. Finin, Y. Labrou and J. Mayfield, KQML as an Agent Communication Language. Software Agents, AAAI/MIT Press (1997).
[10] FIPA: FIPA Specification, Version 2.0, Part 2: Agent Communication Language (1997) Available at http://www.fipa.org/spec/f8a22.zip
[11] B.N. Grosof, Y. Labrou, and H.Y. Chan, A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. Proc. 1st ACM Conf. on Electronic Commerce (EC99), ACM Press (1999).
[12] F.V. Harmelen, and I. Harrocks, FAQs on OIL: The Ontology Inference Layer. IEEE Intelligent Systems **15(2)** (Nov./Dec. 2000) 69–72.
[13] J. Hendler and D. McGuinness, The DARPA Agent Markup Language. IEEE Intelligent Systems **15(2)** (Nov./Dec. 2000) 72–73.
[14] S. Jindadamrongwech, An Agent Communication Language using XML Declarative Description. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2000).
[15] Y. Labrou, T. Finin, and Y. Peng, Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems, **14(2)** (Apr./May 1999) 45–52.
[16] S.A. McIlraith, T.C. Son, and H. Zeng, Semantic Web Services. IEEE Intelligent Systems, **16(2)** (Mar./Apr. 2001) 46–53.
[17] V. Wuwongse, C. Anutariya, K. Akama and E. Nantajeewarawat: XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems (to appear).
[18] V. Wattanapailin, A Declarative Programming Language with XML. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2000).

Utilizing Host Formalisms to Extend RDF Semantics

Wolfram Conen[+] and Reinhold Klapsing[++]

[+]XONAR GmbH,
Wodanstr. 7
D-42555 Velbert, Germany,
Conen@gmx.de

[++]Information Systems and Software Techniques,
University of Essen, Universitätsstraße 9,
D-45141 Essen, Germany,
Reinhold.Klapsing@uni-essen.de

Abstract. RDF may be considered as an application of XML intended to interoperably exchange semantics between Web applications. In its current form, this objective may be hard to reach. Even if the semantical gems hidden in the RDF/RDFS specification are precisely captured, as, for example, in the axiomatic formalizations currently available, the useabilty of RDF's concepts and constraints is limited: RDF offers a data model but does not specify the processing of RDF-encoded data. RDFS describes some basic (ontological) concepts and constraints but does not specify the processing of RDFS-encoded ontological information. The expressiveness of the constraints is rather limited and no clear means of providing semantics for new concepts and constraints are specified. This paper presents one possible approach to overcome this weaknesses. The definition and interpretation of semantics and the processing of the RDF-encoded information will be delegated to a host formalism (first order logic). An elaborated example specifies an extended set-algebraic range constraint and applies the extended vocabulary to a security management task. The definition of semantics is made explicit in the RDF Schemata. The new constraints and concepts are added to the concepts and constraints of an underlying axiomatic interpretation of RDF(S). A Prolog-based implementation of the approach, the RDF Schema Explorer, which is available on-line, is presented. The tool allows to process, validate, query and extend a FOL interpretation of (extended) RDF Schemata.[1]

Keywords: Semantic Web, RDF, Semantic Extensibility, Host Formalism, Prolog

[1]This paper draws from an earlier paper that we will present at the German Wirtschaftsinformatik conference.

1 Exchanging Semantics on the Web

Semantic annotation of data becomes increasingly important, as increasingly complex interactions, involving a multitude of actors, call for a shared and common understanding of the exchanged information. Semantic annotation may enable intelligent search instead of keyword matching, query answering instead of information retrieval [7][2], knowledge base definition instead of data format exchanges etc. The Semantic Web Activity of the World Wide Web Consortium (W3C) emphasizes the importance of semantics for the further development of the Web. The Resource Description Framework (RDF) [9, 2] may develop into one of the foundations of the Semantic Web by enabling semantic interoperability. RDF intends to provide a standard for describing the semantics of information via metadata descriptions (compare [7]).

For the Semantic Web to scale, independent and heterogenous actors (users, agent, tools) must be able to exchange and process (meta-)data based on a common semantic interpretation. One may question if RDF provides the means to achieve this. We want to emphasize two issues here: (1) most aspects of the RDF Schema specification are expressed informally, and (2) the concepts and constraints of the RDF Schema specification do not provide sufficient expressiveness and lack a clear extension mechanism.

The first issue has been addressed before[3] – in [4] we chose first order logic (FOL) to express the main concepts and constraints defined in the RDF specifications. The main benefit of using FOL is that it is a well-studied expression mechanism with a commonly agreed-upon interpretation. This has been utilized in the *RDF Schema Explorer*, a Prolog-based tool we developed that integrates Jan Wielemaker's RDF parser [11] and the axioms given in [4]. A Web-based version of the RDF Schema Explorer is accessible online [10]. It allows to query and validate RDF descriptions not only on the statement level but also with respect to the facts and rules that capture the semantic concepts and constraints of RDF.

The second issue has been discussed in the context of modeling ontologies in RDF(S), see [5]. Staab et al. state about RDF(S) that "the lack of capabilities for describing the semantics of concepts and relations beyond those provided by inheritance mechanisms makes it a rather weak language for even the most austere knowledge-based systems". They propose an approach that extends the semantics of vocabularies expressed in RDF(S) via axioms which are considered as objects that are describable in RDF(S).

Our work, to be discussed below, can be seen as a combination of the work cited above[4]. We also provide means to explicitly specify the (axiomatic) semantic of properties from within RDF, compare Figure 1. This capability is implemented and available in the RDF Schema Explorer [10].

Furthermore, the definition of extended vocabularies is based on the axioms that capture the core RDF(s) concepts and constraints. These axioms are also available accessibly and explicitly. This tight integration of the RDFS concepts / constraints with the extended seman-

[2] Fensel provides an instructive overview of rationales for (ontology-driven) semantics in different networking contexts.

[3] Though, unfortunately, it is not yet on the issue list of the current RDF working group to provide some more formal (axiomatic) semantics for RDFS, so this effort documents only one possible, not-standardized attempt to capture the meaning of RDF Schema

[4] While we implemented the RDF Schema Explorer without knowledge of the approach of Staab et al., we nevertheless very much agree with their rationales for making axioms available "as objects that are describable in RDF(S)". We would like to recommend their paper as a complementary source of well-chosen arguments for extending RDFS with explicitly available axioms

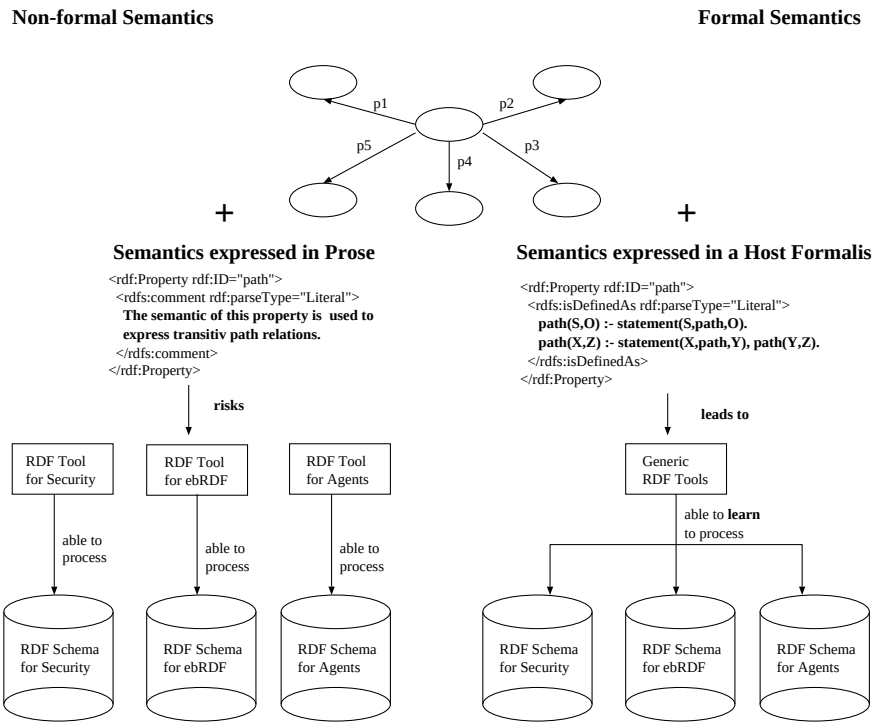


Figure 1: Defining more sophisticated semantics with a host formalism. In the left part of the figure, semantics are informally described within rdfs:comments. This may lead to the development of a plethora of interpretation-specific RDF tools. This is contrasted with the approach to make (axiomatic) meaning explicitly available, thus making it generally accessible for precise and interoperable interpretation (within the limits of the chosen host formalism as far as it extends RDFS).

tics, as well as the availability of a prolog-based implementation maybe considered as the main difference to the work of Staab et al.

Below, we will demonstrate this integration by means of an application that especially emphasizes the use of an extended range constraint in an access-control context. The remainder of this paper is structured as follows. In Section 2 the extension mechanisms is presented. We describe how the *RDF Schema Explorer* operates and which basic predicates are provided to query an RDF description. In Subsection 2.1 the extension mechanism, used to formally define more sophisticated semantics in RDF schemata, is explained. An example, taken from an access control context, is presented in Subsection 2.3 to demonstrate the extension mechanism and the related RDF syntax. We include a brief discussion of one of the core concepts of RDFS, the range constraint. In Section 3 the paper is concluded with a brief discussion of the presented approach.

2 Specifying Extensible Semantics in RDF

Below, the RDF Schema Explorer [10] is presented that allows to query RDF models not only on a statement level but also with respect to the facts and rules that capture the semantic concepts and constraints of RDFS. For this purpose, a number of pre-defined predicates is available. This also allows to validate the models against this RDFS rule set. In addition, it is possible to define the semantics of newly introduced predicates from within RDF and to query/check/validate these extended models.

Predicate	Purpose
`statement(S,P,O)`	Contains the basic facts of the knowledge base.
`res(R)`	Gives the resources.
`lit(O)`	Gives the literals.
`reifies(R,S,P,O)`	R reifies the (not necessarily present) triple `[S,P,O]`.
`reifyingStatement(R)`	R fulfills `reifies/4` for some S,P,O.
`reifies_fact(R)`	R fulfills `reifies/4` for some S,P,O and the triple `[S,P,O]` is indeed in the knowledge base.
`subClassOf(C,D)`	Transitive predicate that captures the relation that is expressed with the `rdfs:subClassOf` property.
`instanceOf(R,C)`	Transitive predicate that captures the relation that is expressed with the `rdf:type/rdfs:subClassOf` properties.
`subClass_cycle_violation(C)`	This is true if the knowledge base allows to infer `subClassOf(C,C)`.
`subPropertyOf(X,Y)`	A transitive predicate that captures the relation that is expressed with the `subPropertyOf` predicate.
`subProperty_cycle_violation(P)`	This is true if the knowledge base allows to infer `subPropertyOf(P,P)`.
`domain_constrained_property(P)`	At least one statement that specifies a domain constraint is present for property P.
`domain(X,P)`	X is an instance of one of the classes that are in the domain of P.
`domain_violation(S,P,O)`	This is true if a statement `[S,P,O]` is in the knowledge base, and P is domain-constrained and S is not in the domain of P.
`is_range(C,P)`	C is (one of) the range restriction(s) for P.
`range_cardinality_violation(P)`	There are (at least) two different range restrictions for P.
`has_range(P)`	P is range-constrained.
`range(X,P)`	X is an instance of (one of) the class(es) to which the range of P is constrained to.
`range_violation(S,P,O)`	P is range-constrained, the statement `[S,P,O]` is in the knowledge base and O is not in the range of P.
`violation(T,S,P,O)`	A convenience predicate that collects the above violations. T will show the type of the violation and S,P,O will be the violating triple - with the exception of `range_cardinality`, where S will be the violating predicate and O will be one of the ranges S should be constrained to. In this case, all ranges that are given will be shown as different instances of `violation`.

Table 1: A collection of the predicates that axiomatize the RDF Schema constraints.

The tool works as follows. First, some RDF-File will be fed into the SWI-Prolog-based RDF parser[5]. This file will be parsed and a relation will be created that contains the triples, e.g.[S,P,O], in a relation `statement(S,P,O)`).[6]

The slightly modified parser tries to *normalize* the URIs–no matter, if a resource is given in subject, predicate, or object position, the parser tries to transform it into the format `namespace:resource_name`. This makes querying much easier. Furthermore, some form of normalization is necessary to be able to discover that `xxx:yyy` and `URI_of_xxx#yyy` are (or better: "represent") indeed the same resource.

Now, one could already query this simple triple database. The tool offers a query field allowing to ask the Prolog engine things like `statement(S,rdf:type,O)` or

[5]Credits go to Jan Wielemaker. Some minor modifications have been made related to namespaces.

[6]Note that we do not assume *per se* that every triple encodes an instance of a binary relation. As has been discussed in [5], a triple plus a reification and a simple negated truth predicate may easily be used to imply intentions that render the mapping to binary relations faulty – e.g. triple [S,P,O], plus Reification R representing [S,P,O], plus triple [R hasTruthValue FALSE] may express that it is known that [S,P,O] is not true.

`setof(O,statement(S,P,O),Z)`. While it is certainly useful to know a little bit about Prolog, it is not necessary, because the tool offers a choice of predefined queries from a preselection list.

However, this would not be completely satisfying. As one will normaly use concepts/constructs from RDFS, the fact and rule base that has been outlined in the paper "A logical interpretation of RDF" ([4]) is provided. The effect is that the knowledge level predicates that are briefly explained in Table 2 can be used to check and query a model with respect to the RDF schema constraints.

In addition, we have defined a number of additional *convenience predicates*. Most of them can be chosen from the pre-selection menu on the query form. An example is `show_statements(S,P,O)` where a value for any of the variables S,P, or O cab be substituted in and a list of the triples containing the substituted value at the corresponding position will be generated.

While this all makes it rather easy to play with the effects of RDF schema concepts and constraints, one will soon discover that the semantics implied by RDFS are pretty general (not to say "weak"). We therefore allow to introduce *semantics on top of the basic facts and rules* which makes it possible to specify more precisely what a modeler intends with her predicates. This can be done in two ways:

1. Either, some Prolog rules may be directly keyed into the query field, for example

   ```
   assert(trans_rel(S,O):- statement(S,path,O)).
   assert(trans_rel(S,O):- statement(S,path,Z), trans_rel(Z,O)).
   ```

   which defines the predicate `trans_rel` to represent a transitive property `path`. This would allow to inquire if two resource are transitively related, or

2. the RDF-level mechanism that we provide to define the semantics of predicates within RDF documents is used. This mechanism will be discussed in some detail in the following subsections.

2.1 The Extension Mechanism

The mechanism to be described allows to provide the semantics for properties within RDF schema declarations. A special predicate `rdfs:isDefinedAs` is available to extend the basic rule set with additional semantics for newly defined properties (it is also possible to define the basic rule set this way). The interpretation of the schemata will rely on a suitably chosen host formalism. For the current implementation, the Prolog-flavor of first-order logic has been selected.

The example below, defining the transitive property `path`, can be fed directly into the RDF Schema Explorer.[7]

```
<?xml version="1.0"?>
<RDF
 xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

[7] Note, however, that to make it exiting, some resource that are related via the path property would be required.

```
 xmlns:rdfs="http://.../TR/2000/CR-rdf-schema-20000327#">

 <rdf:Property rdf:ID="path">
   <rdfs:isDefinedAs rdf:parseType="Literal">
      path(S,O) :- statement(S,path,O).
      path(X,Z) :- statement(X,path,Y), path(Y,Z).
   </rdfs:isDefinedAs>
 </rdf:Property>
</RDF>
```

Note the use of `statement` above, which is meaningful because all predicates that are defined in the basic rule set are accessible.

In the current version of the RDF Schema Explorer, only *Prolog code* may be provided (to be read by SWI-Prolog in the sequence that is implied by the XML serialization[8]). In future versions, other languages (such as implementations of Description Logics [1]) may be allowed as well.

2.2 An Advocacy for a set-algebraic `range`*-constraint*

In RDFS, the applicability and expressiveness of the range constraint is rather limited. To see this, first a brief review of (our version of) the intended semantics of the range constraint in the current version of RDFS is given. In [4] the range constraint has been captured as

```
is_range(X,P) :- statement(P,rdfs:range,X).
has_range(P)  :- is_range(_,P).
range(X,P)    :- is_range(C,P), instanceOf(X,C).
range_violation(S,P,O) :- statement(S,P,O), has_range(P), not(range(O,P)).
```

In RDFS, the following further restrictions apply.

1. At most one range constraint is allowed.
2. Only two distinguished sets of entities, namely *Resources* and *Literals* exist.
3. The semantics of subclassing can be captured with the rule

   ```
   instanceOf(I,C) :- statement(I,rdf:type,C).
   instanceOf(I,D) :- statement(I,rdf:type,C), subClassOf(C,D).
   ```

With an open-world assumption, not much could be deduced from a range constraint[9], because knowing that the range of a property p is constrained to the set $X \subset Resources$ and

[8] Unfortunately, in standard SLD-resolution-based Prolog, sequence does matter. This matches, however, naturally with XML (and not quite so naturally with RDF, which does not use sequence information with the notable exception of Seq-type containers). If one would parse the XML serialization, compute triples from it, scramble the triple sequence and subsequently start to assert the property definitions, this might lead to a behaviour that was not intended – however, it would conform to the notion of RDF as being set-oriented.

[9] We do not infer types from range *constraints*. Rationales: Two possible interpretations of the *range constraint* have been discussed (RDF-IG, Rdf-logic), (a) the *constraint* and the (b) *axiom* interpretation. Roughly, (a) says that a property p may (only) be applied to instances of classes that are in the range of p while (b) states that, from using a resource r as a value of a range-constrained property p, it can be infered that r has the type of the range of p. Formally, both interpretation can be formulated as $instanceOf(O, C) \leftarrow statement(S, P, O), range(P, C)$, with the difference that, with the constraint interpretation, we have to ask if this is a (logical) *consequence* of the known statements (facts) and rules (axioms) while, with the axiom interpretation, this will be treated as one of the rules/axioms that allows us to infer type information (and no

knowing that a resource r is an element of a set $Y \subset Resources$ does not allow to conclude that attaching a value r to p would violate the range constraint. This would only be reasonable if it would be known that X and Y are disjoint. However, this information is only available for *Literals* and *Resources* and is not expressible in RDF for the relation between two (or more) arbitrary subsets of *Resources*. Assuming that the world is closed and complete, one could argue that two subclasses X, Y of a class R are disjoint if no entity is known that is an instance of both classes. Nevertheless, two problems remain: schemata are mostly used to guide the design/evolution of models, ie. not all instances will be known at schema design time – and introducing further information may render earlier decisions inconsistent (because adding a type information to a resource may show that two classes are in fact not distinct but overlapping etc.) – SO, considering a world as complete is dangerous with respect to inter-temporal validity. In addition, only a richer set of constraints (including set-union, set-difference and set-disjunction) would allow to specify all constraints that seem reasonable if the range of a property should be restricted. To see this consider the following: The are two classes, $C1$ and $C2$, and a property p. With "reasonable" we mean the following range constraints: for `[x,p,y]`, `range(p,Exp)` may constrain y to be an element of Exp defined as

$Exp := C1 \cup C2$	(y in $C1$ OR y in $C2$)
$Exp := C1 \cap C2$	(y in $C1$ AND y in $C2$)
$Exp := C1 \backslash C2$	(y in $C1$ AND y NOT in $C2$)
$Exp := C2 \backslash C1$	(y in $C2$ AND y NOT in $C1$)
$Exp := (C1 \backslash C2) \cup (C2 \backslash C1)$	(y in $C1$ XOR y in $C2$)
$Exp := !(C1)$	(y not in $C1$)

An often suggested extension of RDFS is to allow multiple range constraints and to interpret these constraints as binding the allowed range to the disjunction of the classes. However, this would restrict the interpretation of multiple range constraints to one (limited) case of the cases given above[10]. Below, we will suggest a solution that not only conforms to RDF but also offers a flexible and general way to specify range constraints. The required interpretation can be encoded on schema level, making it possible to specify and enforce different *types* of range constraints in different application domains.

Below, only one range constraint will be allowed. This is sufficient if classes (or class expressions) can be constructed from other classes (or class expressions). In this case, each range constraint will point to exactly one class and the *construction* of the class directly expresses the constraint. Above, the *Exp* term represents the constructed class and the right hand side gives the construction expression. An example for applying a range constraints using a constructed class is:

validation will be possible). We adopt the practice of the examples in (Sec. 3.1, Sec. 7.1 of [2]), where types are assigned to resources with the *rdf:type*/*rdfs:subClassOf* properties, and the range-constraint is used to "state that a ... property only ´makes sense' when it has a value which is an instance of the class ... ", allowing for **validation**. This conforms to interpretation (a) above. Please note that now, no types of resources will nor should be infered, instead it is possible to check (with the range constraint) if properties are applied to resources of the correct type (with rdf:type, rdfs:subClassOf or subproperties of these properties as the available devices to provide typing information).

[10] A solution could be to introduce specific range constraints / range constraint types for all of the above cases. This is, however, problematic, because it does not scale very good to "mixed" range dependencies with $3, 4, \ldots, n$ classes.

```
[C1,rdf:type,rdfs:Class]
[C2,rdf:type,rdfs:Class]
[A,rdf:type,ConstructedClass]
[A,isConstructedFrom,"C2 \ C1"]
[p, rdfs:range, A]
```

With `[X, rdf:type, C1]`, X would violate the intended range constraint if it would be chosen as a value for p.

If it is assumed that the object "`C2 \ C1`" is modeled as a literal, the above solution can be formulated as well-formed RDF easily. However, to interpret it, an application-level check of the class construction semantics would be required. This is not really nice, because range constraints seem to be too important to leave their semantics to "proprietary" vocabularies and interpretations, but this might be a matter of taste. With respect to the intended interoperability based on RDF schemata, making the semantics of the constructs expressible within RDF seems to offer a more interoperable solution. In fact, the property `isConstructedFrom` denotes a multi-ary relation between classes. This can be transformed (generally) into a sequence of (3-ary) "atomic" set-algebraic operations (expressed below as nested tuples), as in the following example that expresses $A = (C1 \cap C2) \backslash C3$.

```
[ A1, intersection, [C1,C2] ]
[ A,  difference, [A1, C3] ]
```

In RDF, this is expressible using reification and a suitable interpretation of the reified statements:

```
[ A1, rdf:type, rdf:Statement ]
[ A1, rdf:subject, C1 ]
[ A1, rdf:predicate, rdfsets:intersection]
[ A1, rdf:object, C2 ]

[ A, rdf:type, rdf:Statement ]
[ A, rdf:subject, A1 ]
[ A, rdf:predicate, rdfsets:difference]
[ A, rdf:object, C3 ]
```

Suitably interpreted, this allows to express a set algebraic range constraint like:

```
[ p, rdfsets:range, A ]
```

2.3 Sharing Security Schemata – An Example

In the following we demonstrate how such set constructs can be defined in an RDF-conform manner by applying the above introduced extensions mechanism to the domain of role-based access control. The semantics are build upon the basic RDF rules given in [4]. In the example below[11], the task is to decide if access to certain documents should be granted to certain users. The decision depends on the membership of users in certain groups[12]. Figure 2 depicts the specific situation.

[11] The RDF source of the following example is easily accessible as part of the RDF Schema Explorer on-line demonstration [10].

[12] Conceptually, membership in groups or role assignment can both be represented with set-algebraic class expression – and this is the mechanism used in this example.

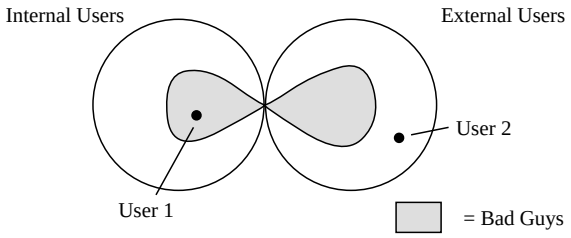


Figure 2: Access shall only be granted to users in the white section of the above venn diagrams, i.e., *bad guys* like user 1 should not get access.

Three new predicates are introduced, namely *union*, *difference* and *intersection*[13]. These predicates can be used to construct classes from other classes with the help of binary relations and reification, both being completely valid RDF constructs. This will be utilized to construct classes from set-algebraic expressions over other (constructed) classes.

The extension is based on the already introduced semantic primitive *isDefinedAs* (to ease the demonstration, we assume that the property is in the `rdfs` namespace). To make it possible to mix meta-schema, schema and instance expressions in the example below, we adopted the following convention: if a namespace `this#` is introduced, the namespace abbreviation will be omitted during the parsing process. This makes it possible to use the namespace within the document while still being able to normalize the resource names to make them easily useable for querying the model.[14].

First, a subclass of `rdfs:Class`, `ConstructedClass` is introduced. The rules described above are used to define the semantics of the newly introduced predicates. Additionally, the semantics of both the `type` and the `range` property are (monotonically) extended to be able to cope with constructed classes.

```
<?xml version="1.0"?>
<RDF
 xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://.../TR/2000/CR-rdf-schema-20000327#"
 xmlns:rdfsets="this#">

 <!-- Meta Schema definitions -->
 <rdfs:Class rdf:ID="ConstructedClass">
  <rdfs:subClassOf rdf:resource=
    "http://.../TR/2000/CR-rdf-schema-20000327#Class"/>
 </rdfs:Class>

 <Description
  about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    constructed_class(C):-instanceOf(C,'ConstructedClass').
  </rdfs:isDefinedAs>
 </Description>
```

[13] A NOT will not be introduced because it allows to formulate unbounded class expressions, ie. expressions that depend on an (unknown) universal set. Set-difference contains implicit (bounded) NOT constraints and is sufficient for most purposes.

[14] The reader may adopt this practice with self-developed extension schemata to make it easy to feed schemata and instances as one document into the RDF Schema Explorer [10].

```
<Property rdf:ID="union">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = union, instanceOfSet(X,S).
    in(X,S,P,O) :- P = union, instanceOfSet(X,O).
  </rdfs:isDefinedAs>
</Property>

 <Property rdf:ID="difference">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = difference,
       instanceOfSet(X,S), not(instanceOfSet(X,O)).
  </rdfs:isDefinedAs>
</Property>

<Property rdf:ID="intersection">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    in(X,S,P,O) :- P = intersection,
       instanceOfSet(X,S), instanceOfSet(X,O).
  </rdfs:isDefinedAs>
</Property>

<Description about=".../CR-rdf-schema-20000327#range">
  <rdfs:isDefinedAs rdf:parseType="Literal">
    instanceOfSet(X,A) :- constructed_class(A),
       reifies(A,S,P,O), in(X,S,P,O).
    instanceOfSet(X,A) :- instanceOf(X,A).
    range(X,P) :- is_range(C,P), instanceOfSet(X,C).
  </rdfs:isDefinedAs>
</Description>
```

Now the schema definitions follow, expressing that `Internal_Users`, `External_Users`, and `Bad_Guys` are plain classes and that `All_Users` and `Trusted_Users` are constructed classes, with `All_Users` $=$ `Internal_Users` $\cup$ `External_Users` and `Trusted_Users` $=$ `All_Users` $\setminus$ `Bad_Guys`.

```
<rdfs:Class rdf:ID="Internal_Users"/>
<rdfs:Class rdf:ID="External_Users"/>
<rdfs:Class rdf:ID="Bad_Guys"/>

<rdfsets:ConstructedClass rdf:ID="All_Users">
  <subject   rdf:resource="#Internal_Users"/>
  <predicate rdf:resource="#union"/>
  <object    rdf:resource="#External_Users"/>
  <type      rdf:resource=".../22-rdf-syntax-ns#Statement"/>
</rdfsets:ConstructedClass>

<rdfsets:ConstructedClass rdf:ID="Trusted_Users">
  <subject   rdf:resource="#All_Users"/>
  <predicate rdf:resource="#difference"/>
  <object    rdf:resource="#Bad_Guys"/>
  <type      rdf:resource=".../22-rdf-syntax-ns#Statement"/>
</rdfsets:ConstructedClass>
```

Access will be granted according to a closed security policy that is, all accesses have to

be allowed explicitly. This will be expressed by attaching a property AccessAllowedFor to resources that is constrained to the range Trusted_Users.

```
<Property rdf:ID="AccessAllowedFor">
  <rdfs:range rdf:resource="#Trusted_Users"/>
</Property>
```

The following instance definitions will entail a range constraint violation.

```
<Description rdf:ID="user_1">
  <type rdf:resource="#Internal_Users"/>
</Description>

<Description rdf:ID="user_1">
  <type rdf:resource="#Bad_Guys"/>
</Description>

<Description rdf:ID="user_2">
   <type resource="#External_Users"/>
</Description>

<!-- Objects to restrict access to: -->
<rdfs:Class rdf:ID="Important_Documents"/>

<rdfsets:Important_Documents rdf:ID="Weak_Secret_1">
  <rdfsets:AccessAllowedFor rdf:resource="#user_1"/>
  <rdfsets:AccessAllowedFor rdf:resource="#user_2"/>
</rdfsets:Important_Documents>
</RDF>
```

Here, user_1 is known as a bad guy, accordingly, he should not be granted access. In fact, the range constraint on AccessAllowedFor is violated. To see this, consider the extended rule set for the set-algebraic range constraint:

```
/* RDFS rule set */
is_range(X,P) :- statement(P,rdfs:range,X).
has_range(P)  :- is_range(_,P).
range(X,P)    :- is_range(C,P), instanceOf(X,C).

/* Extension */
range(X,P) :- is_range(C,P), instanceOfSet(X,C).

/* Detecting the violation (from RDFS rule set) */
range_violation(S,P,O) :-  statement(S,P,O), has_range(P), not(range(O,P)).
```

The RDF descriptions above allow to derive that user_1 is *not* a member of the constructed class Trusted_Users and thus, is not in the range of AccessAllowedFor.

We hope that this simple example may already demonstrate that the above mechanism, together with a Prolog engine, is a pretty powerful instrument to *define/extend semantics*, to *validate documents* against RDFS and user-provided constraints, and to *query a model on the knowledge level*. This may help to leave the simplistic triple structure behind and to capture the meaning of (extended) vocabularies more precisely. It allows to develop domain specific vocabularies build upon the formalized RDF/RDFS constraints. These vocabularies can be re-used in schema definitions for other domains as well. The RDF Schema Explorer will support this with dynamic loading and incremental interpretation of schema definitions (via HTTP).

3 Discussion

The approach outlined above allows to define RDF (meta-)schemata that precisely capture the semantic intentions if interpreted within a suitable host formalism. The approach represents the intended semantics of RDF schemata explicitly, making it possible to treat the definition as first-class resources within RDF[15]. The approach is paradigm-independent, as it allows to select different host formalisms for specific purposes [16]. The specific Prolog-based instantiation of the approach is expressive as it allows to utilize the available expressiveness of Prolog. Furthermore, production-quality implementations of Prolog are widely available. It may be asked why pure Prolog or any other KR Language (like KIF/SKIF) has not been chosen as an implementation language for the semantic web. We think that constraining people to a certain implementation language may not always be a good idea. There are always pros and cons for a certain implementation language. We propose to give an implementer the possibility to use a suitable implementation language for her application domain. Pure RDF/RDFS remains to be an exchange mechanism for (rudimentary) knowledge while an implementer should have the choice to integrate this basic knowledge (for example based on an axiomatization of RDFS) with more elaborate semantics defined on top of a suitable host formalism (with the consequence that this part of the knowledge may not be interpretable in different host formalisms).

To summarize: We presented a detailed example that demonstrates the use of the involved techniques in an access control context. The Prolog-based RDF Schema Explorer that we developed allows to validate and query such extended models. Both, the tool and a workable version of the example are accessible on-line. Besides being able to interpret (extended) RDF schemata, the tool is suitable to support the prototyping of domain-specific schemata, as the semantics of the defined properties can be changed on the fly and the consequences can be inspected utilizing the convenience predicates (such as `violation`, `show_classes`, etc.).

We expect that the interoperable definition of meta-schemata will develop into a necessity, once the formulation of complex semantic constraints in various emerging application domains such as cooperative security management, automated business contract negotiation etc. – all involving a number of autonomous partners and, thus, exhibiting a need for semantic interoperability – is identified as a key requirement for the success of the underlying collaborations.

References

[1] Alexander Borgida. Description Logics in Data Management. *Knowledge and Data Engineering*, 7(5):671–682, 1995. http://citeseer.nj.nec.com/borgida95description.html.

[2] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[15] This allows to apply the RDF concepts to describe/relate the semantic definitions as well. For example, new properties expressing containment, semantic dependencies, abstraction etc. can be defined and used, which may ease to maintain and re-use the (meta-) schemata.

[16] Both, making the semantics of the underlying (meta-)concepts explicit and being not bound to a specific world view / paradigm (such as, for example, ontology-based agent modeling), renders our approach different from such languages as OIL [3, 6] or DAML [8] that offer a set of non-manipulable primitives whose semantics are not expressed in the RDF-based languages themselves. This necessarily restricts the applicability of the languages to domains/applications that exhibit a “natural” and “close” fit with the concepts the languages offer.

[3] Jeen Broekstra, Michel Klein, Dieter Fensel, Stefan Decker, and Ian Horrocks. OIL: a case-study in extending RDF-Schema. Technical report, ontoknowledge.org, 2000. http://www.ontoknowledge.org/oil/oil-rdfs.pdf.

[4] Wolfram Conen and Reinhold Klapsing. A Logical Interpretation of RDF. *Linköping Electronic Articles in Computer and Information Science, ISSN 1401-9841*, 5(13), December 2000. http://www.ep.liu.se/ea/cis/2000/013/.

[5] Wolfram Conen, Reinhold Klapsing, and Eckhart Köppen. Rdf m&s revisited: From reification to nesting, from containers to lists, from dialect to pure xml. In *Proceedings of the Semantic Web Working Symposium (SWWS)*, Stanford, August 2001.

[6] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a Nutshell. Technical report, ontoknowledge.org, 2000. http://www.cs.vu.nl/ dieter/oil/oil.nutshell.pdf.

[7] Dieter Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, Heidelberg, 2001.

[8] Ian Horrocks, Frank van Harmelen, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb McGuinness, Peter Patel-Schneider, and Lynn Andrea Stein. DAML+OIL Language. http://www.daml.org/2000/12/daml+oil-index.html, December 2000.

[9] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[10] Web-based RDF Schema Explorer. http://wonkituck.wi-inf.uni-essen.de/rdfs.html.

[11] SWI-Prolog. http://www.swi.psy.uva.nl/projects/SWI-Prolog/.

RDF M&S revisited: From Reification to Nesting, from Containers to Lists, from Dialect to pure XML

Wolfram Conen[+], Reinhold Klapsing[++], and Eckhart Köppen[+++]

[+]XONAR GmbH,
Wodanstr. 7
D-42555 Velbert, Germany,
Conen@gmx.de

[++]Information Systems and Software Techniques,
University of Essen, Universitätsstraße 9,
D-45141 Essen, Germany,
Reinhold.Klapsing@uni-essen.de

[+++]40Hz.org
343A Nature Drive
San Jose, CA 95123
eck@40hz.org

Abstract. The basic result presented is the following: with two (hopefully reasonable) assumptions about the intentions behind the RDF model, it can be shown that the RDF model and a model allowing for nested triple and lists of resources and triples, can be mapped to one another (in both directions). This allows to establish a close link between the RDF model and extended models recently suggested (SlimRDF [3], XRDF [4]). Further, the approach may help to clarify some problems related to interpreting the roles of reification and containers in the RDF model.

1 Introduction

As RDF is considered to be a key ingredient of the evolving semantic web, lack of clarity should be avoided. *Reification* and *Containers* gave rise to a number of discussions. In this paper, we propose an interpretation of these two constructs that may help to clarify this issue. It also demonstrates, how complex expressions can be constructed from RDF that allow a straightforward representation of the modeler's intentions. The basic idea is as follows: In RDF, if someone wants to express a relation between a statement and a resource or two statements, she has to utilize reification. If a relation between an entity (be it a resource or a triple or another group of entities) and a group of entities should be expressed, rdf:Bag, rdf:Seq or rdf:Alt have to be used. Essentially, both constructs are needed to allow expressing

nested or grouped constructs with flat triples. Both constructs are not properly tied into the RDF model, for example, the meaning of attaching a predicate to a reificant[1] is not fixed in the model (if r reifies $[s, p, o]$ and $[r, p_2, o_2]$ is given, is the intention to express $[s, p, o]$ *is* p_2*-related to* o_2 or is the intention to take the triple literally, that is r *is* p_2*-related to* o_2?). Fig. 1 and Fig. 2 demonstrate the interpretation of a collection of flat-triple statements as one nested triple.

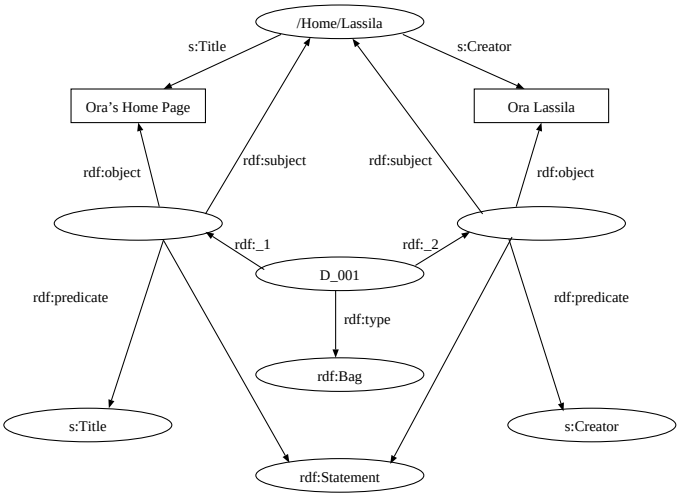


Figure 1: Demonstrating bags and reification (Figure 9 in [2])

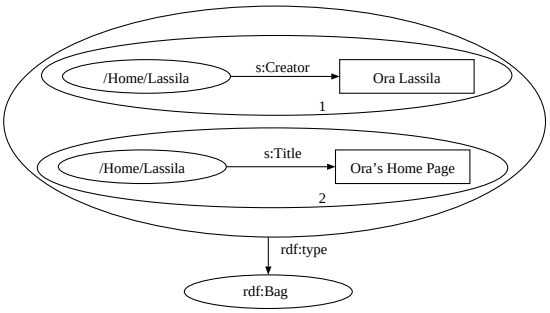


Figure 2: An intention-equivalent nested representation.

We decided to fix the possible interpretation of the flat-triple constructs *reification* and *containers* by assuming that each reification is only a surrogate for the triple it represents and each container is only a surrogate for a list of entities[2], that is, in a natural representation of the intentions, each reification and each container will be substituted by the represented triple or list and all only technically necessary triples of the flat model will be eliminated. We will argue that such a non-flat model captures the *essence* of the initial set of statements. In the following, the two underlying assumptions will be presented more precisely.

1.1 Assumptions

Let S be a set of flat-triple statements.

[1] This terminology will be explained shortly.

[2] An entity may be a resource, a literal, a statement or a list of entities.

Assumption 1: Be r the *reificant*[3] of the triple $[s, p, o]$ as defined in [2], that is, the set S contains the triples $[r, \text{rdf:subject}, s]$, $[r, \text{rdf:predicate}, p]$, $[r, \text{rdf:object}, o]$ and $[r, \text{rdf:type}, \text{rdf:Statement}]$ (we will generally present triples in an infix[4] sequence, that is as $[subject, predicate, object]$). The reification is used in another statement, say $[s_1, p_1, r]$. *Now, we assume that the intentions behind this subset of statements is to express that p_1 relates s_1 to $[s, p, o]$. This intention can easily be captured in a model allowing nested triples as $[s_1, p_1, [s, p, o]]$.*

Assumption 2: Be c a container, for example of type Seq, that is $[c, \text{rdf:type}, \text{rdf:Seq}] \in S$. The n-ary sequence $R_C = (r_1, \ldots, r_n)$ of resources contains the elements of c, that is, $[r, \text{rdf:}\_i, r_i] \in S$ for all $i, 1 \leq i \leq n$. The container is used in another element of S, say $[s, p, c]$. *Now, we assume that the intentions behind this subset of statements is to express that p relates s to $(r_1, \ldots, r_n)$. This intention can easily be captured in a model allowing for lists of resources as $[s, p, (r_1, \ldots, r_n)]$.*[5]

Based on these two assumptions, the role of containers and reification can intuitively be described as allowing to express structure of arbitrary complexity with the limited instrument of a model based on flat triples.

In the next section, an extended model will be suggested that directly captures the underlying intentions of the constructs *reification* and *container* by introducing nesting and lists. In Section 3, we will prove that every RDF model can be expressed as an extended model *and* that every extended model can be expressed as an RDF model. This may suggest that the more comprehensive representation of the RDF model (that is: the extended model) may be prefered when RDF models are used in applications. In Section 4 this aspect is briefly explored and two representation of the extended model, namely a graph notation and a straightforward XML DTD are suggested. In Section 5, two issue that explore the relation between structural and semantical aspects may give hints on possible directions for developing semantics for the extended models. Section 6 concludes the paper with a brief discussion.

2 An Extended Model

Let A be a reasonably selected alphabet. Let A^* be the set of strings defined over A. The following grammar defines expressions over A^* of the form R recursively as

$$R \quad ::= \quad r \quad | \quad '(' \; R \; ',' \; R \; ')' \quad | \quad '[' \; R \; ',' \; R \; ',' \; R \; ']'$$

Here, r denote elements of A^*. (Sub-)Expressions of this form will be called *atomic*. A (sub-)expression of form R is called *resource*. A (sub-)expression of form R which matches the pattern $[R, R, R]$ is called *statement*. A (sub-)expression of form R which matches the pattern (R, R) is called *list*. Note that we will frequently use $(r_1, r_2, \ldots, r_n)$ instead of the more cumbersome $(r_1, (r_2, (\ldots, (r_{n-1}, r_n) \ldots)$ where this can be done without the risk of misinterpretation. We will also leave out the comma regularly. Furthermore, we will only consider

[3] In [2], this is called *reified statement*, which might be a bit confusing–there is something that is reified, yes, but that is the "original" statement $[s, p, o]$. Instead of *reificant*, *reifying resource* could be used. Note, that it would not be very useful to say *reifying statement* because r is **not** defined to be a member of the set **statement** (which is a concept of the RDF model defined in [2]), instead r has the **type** *rdf:Statement*, which is, for the core RDF model, only a string representing a resource, and requires an interpretation in the context of RDF Schema and its constraints and concepts.

[4] infix with respect to the predicate.

[5] Note, that both assumptions together can be used to build list of statements etc.

finite sets of finite expressions.

3 Relation between Extended model and RDF model

Let us begin with a remark: we will assume that the sets of statements of the RDF model that we will consider below are, in a certain sense, well-behaved, that is, we will assume that no reificant nor container is part of the (possibly complex) structure that it represents[6]. We will capture this more precisely in an algorithm that tries to order resources in strata so that each resource in a stratum represents structures that consist of resources of lower ranking strata. The algorithm can also be used to detect whether its input (a set of statements of the RDF model), is not well-behaved.

Below, we will show that extended model and RDF model can be mapped to another. First, some definitions are needed to prepare the stage. In the following, $s, p, o, s_1, o_1, r_1, \ldots, r_n$ will all be entities, that is, either an atom, a statement or a list if the extended model is considered or resources or literals (only possible in object position) if the RDF model is considered.

Note that the definitions 1 and 3 can be applied to both models.

Definition 1 (Reification).
Given a resource r and the following set of statements, T^r:
$T^r = \{ \quad [r, \textit{rdf:subject}, s], [r, \textit{rdf:predicate}, p],$
$[r, \textit{rdf:object}, o], [r, \textit{rdf:type}, \textit{rdf:Statement}] \quad \}$
Then, r is called a reificant *of $[s, p, o]$ and T^r is called a* reification *of $[s, p, o]$.*

Definition 2 (Reification: Derivation, Consequence).
Let u be a nested statement of the extended model of the form $[[s, p, o], p_1, o_1]$. Let r be a reificant of $[s, p, o]$ and T^r be the corresponding reification. The set $D = [r, p_1, o_1] \cup T^r$ is called a derivation *of u. u, in turn, is called a* consequence *of D (analogously defined for $u = [s_1, [s, p, o], o_1]$ and $[s_1, p_1, [s, p, o]]$). With respect to a set $\mathcal{C}$ of statements, we say that u is* derivable *in $\mathcal{C}$ if $D \subseteq \mathcal{C}$.*

Definition 3 (Container).
Given a resource c and the following set of statements (with $X \in \{$rdf:Seq, rdf:Alt,rdf:Bag $\}$): $T^c = \{ [c,$ rdf:type, $X]\} \cup \{[c,$ rdf:_i, $r_i] \mid i \in \mathbb{N}, 1 \leq i \leq n\}$. Then, c is called a container, *T^c is called an n-ary* container definition *and the n-ary sequence $R_c = (r_1, \ldots, r_n)$ of entities is called the* elements *of c.*

Definition 4 (Container: Derivation, Consequence).
Let u be a nested statement of the extended model of the form $[(r_1, \ldots, r_n), p, o]$. Let c be a container for the elements $(r_1, \ldots, r_n)$ of the Seq-type and let T^c be the corresponding container definition. The set $D = [c, p, o] \cup T^c$ is called a derivation *of u. u, in turn, is called a* consequence *of D (analogously defined for $u = [s, (r_1, r_2, \ldots, r_n), o]$ and $[s, p, (r_1, r_2, \ldots, r_n)]$). With respect to a set $\mathcal{C}$ of statements, we say that u is* derivable *in $\mathcal{C}$ if $D \subseteq \mathcal{C}$.*

[6] In the RDF model, as it is described in [2], it is, for example, possible to reify a statement that contains the representing reificant–which should, almost certainly, not be allowed. The same goes for containers containing themself.

Note that a *consequence* can not be a statement from the flat RDF model. However, the *derivation* of a statement with only one level of nesting can completely consist of statements from the flat model. To be able to define some notion of equivalence between sets of statements from the extended and the flat model, we have to define how a deeply nested statement can be derived recursively from a set of flat statements.

Definition 5 (Rooted, Root, Hull). *Be $\mathcal{O}$ a set of flat statements from the RDF model. Let $\mathcal{N}$ be a set of statements from the extended model. Let u be a statement from the extended model. We say that u is* rooted *in $\mathcal{O}$ if either*
(1) $u \in \mathcal{O}$ or
(2) a derivation D of u can be given such that each statement $t \in D$ is rooted *in $\mathcal{O}$.*
We say that $\mathcal{N}$ is rooted *in $\mathcal{O}$ if every statement n of $\mathcal{N}$ is* rooted *in $\mathcal{O}$. $\mathcal{O}$ is called* root *of u if u is rooted in $\mathcal{O}$; it is called* minimal root *of u if it is a root of u and for any statement $t \in \mathcal{O}$, u is not rooted in $\mathcal{O}\backslash\{t\}$. $\mathcal{O}$ is called* root *of $\mathcal{N}$ if every statement $n \in \mathcal{N}$ is rooted in $\mathcal{O}$; it is called* minimal root *of $\mathcal{N}$ if it is a root of $\mathcal{N}$ and for any statement $t \in \mathcal{O}$ a statement $n \in \mathcal{N}$ can be found such that n is not rooted in $\mathcal{O}\backslash\{t\}$. We say that $\mathcal{N}$ is a* hull *of $\mathcal{O}$, if $\mathcal{O}$ is a minimal root of $\mathcal{N}$ – we will alternatively say that $\mathcal{N}$ and $\mathcal{O}$ are* intention-consistent *(or simply* consistent*).*

Example The following set $\mathcal{O}$ of flat statements is a *minimal root* (that is, no statement can be removed from $\mathcal{O}$) for the nested statement [Gustaf says [Ecki likes (Reinhold Wolfram)]]: { [Gustaf says r_1], [r_1 rdf:type rdf:Statement], [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes] [r_1 rdf:object l_1], [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] }. The sets { [Gustaf says [Ecki likes (Reinhold Wolfram)]] } or { [Gustaf says [Ecki likes (Reinhold Wolfram)]], [Gustaf says r_1], [r_1 rdf:predicate likes} are, among finitely many others[7], are *hulls* of $\mathcal{O}$.

We look for hulls that contain only the minimally necessary number of statements to capture, with respect to the above asumptions, the intentions of the underlying set of flat statements.

Definition 6 (Essence). *Be $\mathcal{O}$ a set of flat statements from the RDF model. Let $N^{\mathcal{O}} = \{\mathcal{N} | \mathcal{N}$ is a set of statements from the extended model $\wedge$ $\mathcal{N}$ is a hull of $\mathcal{O}\}$ be the* set of possible hulls *of $\mathcal{O}$. The subset of $N^{\mathcal{O}}_{\min} = \{\mathcal{N} \in N^{\mathcal{O}} | \nexists \mathcal{M} \in N^{\mathcal{O}}$ with $|\mathcal{M}| < |\mathcal{N}|\}$ is the* set of minimal hulls. *An element of $N^{\mathcal{O}}_{\min}$ is called a* minimal hull *or* essence *of $\mathcal{O}$ – we will alternatively say that $\mathcal{N}$ and $\mathcal{O}$ are* intention-equivalent *(or simply* equivalent*).*

Note that due to the definitions of derivations, the minimal hull of a given set of statements from the RDF model is unique.

Example The minimal hull for the set $\mathcal{O}$ of the above example is { [Gustaf says [Ecki likes (Reinhold Wolfram)]] }.

Now, the following two propositions can be proved. The first one essentially states, that each set of extended statements can be expressed as an intention-consistent set of flat statements, while the second will show that each set of flat statements can be expressed as an intention-equivalent set of extended statements.

Proposition 1: *For each set $\mathcal{N}$ of statements from the extended model, a set $\mathcal{O}$ of statements from the RDF model can be found such that $\mathcal{O}$ is a minimal root of $\mathcal{N}$.*

[7] In contrast, for a set of nested statements there is usually an infinite set of possible minimal roots due to the possible variations in naming the necessary containers and reificants.

Proof: Intuitively, each nested statement can be expressed with a set of flat statements that allows to derive, possibly incurring intermediate nested statements, the initial statement (some care has to be taken not to confuse the *symbols* used in the model). Let us consider an example:

Initial Expression:	[Gustaf says [Ecki likes (Reinhold Wolfram)]]
First Step:	[Gustaf says [Ecki likes l_1]]
(add derivation of	[l_1 rdf:type rdf:Seq]
the list)	[l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram]
Second Step:	[Gustaf says r_1]
(add derivation of	[r_1 rdf:type rdf:Statement], [r_1 rdf:subject Ecki]
the embedded	[r_1 rdf:predicate likes], [r_1 rdf:object l_1]
statement)	[l_1 rdf:type rdf:Seq]
	[l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram]

This can be formalized as follows. Be C^E a set of statements[8] from the extended model. With the following construction, an intention-consistent set C^R of statements from the RDF model can be determined.

Algorithm *Flaten(In: C^E)*
(1) $C^R = \emptyset$. Foreach $t \in C^E$ do
(2) Expand(t,0,C^R)

(1) Function *Expand* (In: Expression t, In: Int l, InOut: Set of Statements E) returns a *Symbol*
(2) **If** $t \in A^*$ **then** return t
(3) **If** $Form(t)$ =Statement (matching $[s, p, o]$) **then**
(4) s_r = Expand(s,l+1,E); s_p =Expand(p,l+1,E); s_o = Expand(o,l+1,E)
(5) r =Symbol[9](t);
(6) **if** $(l = 0)$[10] **then** $E = E \cup \{ [s_r, s_p, s_o]\}$; return *EmptySymbol* **else**
$E = E \cup \{ [r, \text{rdf:type}, \text{rdf:Statement}]$,
$[r, \text{rdf:subject}, s_r]$, $[r, \text{rdf:predicate}, s_p]$ $[r, \text{rdf:object}, s_o]$ $\}$; return r
(7) If $Form(t)$ =List (matching $(r_1, \ldots, r_n)$) then
(8) r =Symbol(t);
(9) $E = E \cup \{ [r, \text{rdf:type}, \text{rdf:Seq}]\}$
(9) For $1 \leq i \leq n$ do
(10) s_i =Expand(r_i,l+1,E)
(11) $E = E \cup \{ [r, \text{rdf:}\_i, s_i] \}$
(12) return r;

Let us sketch the proof of the correctness of the algorithm: (1) The algorithm terminates. To see this, consider the following: The function *Expand* recursively descents through the structure of its input expression. It will stop the descent in each branch of the structure as soon as

[8] Arbitrary sets of expressions resp. resources could also be allowed. This would require only a simple, but unnecessary (for this presentation) extension.

[9] The function $Symbol$ returns a new symbol for each subexpression t that is not already represented in the flat model, otherwise, the already known symbol will be returned. This will be discussed below

[10] The top-level expression is always a statement. There is no need to reify this statement because the reificant would be left unused (in this particular expansion).

an element of A^* is found (which will ultimately be the case, as the expressions have been assumed to be finite). Furthermore, each subexpression branch will be considered exactly once. (2) C^E and and the computed set C^R are intention-consistent. To see this, consider the following. First note that all statements added to C^R are flat. The algorithm constructs a derivation for each top-level statement by constructing derivations for each embedded expression while returning from the descent. Thus, each statements of C^E is rooted in the constructed C^R. C^R is a minimal root because no other statements but elements of derivations are added to C^R. $\boxminus$

A note regarding the function *Symbol*. In the algorithm, we have chosen to compute one *unique* name for literally identic subexpressions, that is if, say, the statement [Ecki likes RDF] is encountered twice, it will be reified only once (although, to keep the algorithm above simple, it will be flatened twice but this will result in an identic set of flat statements and the redundancy will thus vanish due to considering sets). This makes it easy to identify literally equivalent expressions in the flat model (they have the same "name"), it, however, may make it more difficult to explore the differences in the meaning of multiple occurences of literally identic expressions (this will have to consider the structural context of such expressions – sensibly dealing with this kind of context should be made possible in the semantics build upon this models, so, for a full discussion of the implications, precise objectives for semantics are required. It is, nevertheless, easy to generate a new symbol for each occurence of literally identic subexpressions, if this is found to be the better way to go).

Proposition 2: *For each set $\mathcal{O}$ of statements from the RDF model, a set $\mathcal{N}$ of statements from the extended model can be found such that $\mathcal{N}$ is a minimal hull of $\mathcal{O}$.*

Proof: (constructive) The resources used in the RDF model can be arranged in strata if it is assumed that no circular definitions of reifications resp. containers exist (see below for details). The following algorithm will either determine a stratification or detect that circular references exist.

Algorithm Stratification(In: C^E)
Initially, all resources and literals are unmarked.
[Compute Stratum 0] Mark all literals as being in stratum 0. Mark all resources that are neither a reificant nor a container as being in stratum 0.
[Compute further Strata] **while** there is a resource r that is *unmarked* **and** all the resources or literals that are represented by r (this set of entities will be called E)[11] are marked **do**
  Determine the highest marking, say j, of a resource in R.
  Mark r to be in stratum $j + 1$.
[Check validity] **if** an unmarked resource exists
  then return "ERROR: there are mutually referencing structures"
  else return "OK - a stratification has been determined".

With the above assumption of a finite input set and finite expressions, the algorithm eventually terminates (in each round, an unmarked resource is marked). If the algorithm prints out the "OK" message, the following condition will hold: each resource r that represents a structure

[11] If r is a reificant and $[s, p, o]$ is a statement that r reifies then s,p and o are in E. If c is a container then the elements of R_c (as defined above) are in E. Note that with the definition of container above, a set of flat statement that defines a n-ary container also defines $(n-1)$, $(n-2)$ ... -ary containers. We assume that E contains all eintities that are elements of the container with the largest arity. Besides this solution for the case in which r represents more than one structure, all other cases should probably be considered an error (for example, a resource that represents two statements or a statement and a container, or two non-inclusive containers).

belongs to a higher stratum then the resources/literals that are part of the represented structure. If the algorithm returns an ERROR message, at least one resource represents a structure that contains either the resource or a structure that, if recursively dereferenced, contains the resource.[12]

From finiteness follows that a highest ranking non-empty stratum, say k exists. Furthermore, it follows from the construction that the strata that are formed by marking resources and literals as their elements, are consecutively numbered.

Example This is an example of a mapping from a stratified set of flat statements to an extended statement:
Input: { [Gustaf says r_1] [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object l_1], [r_1 rdf:type rdf:Statement], [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] }
Stratum 0: { Gustaf Reinhold Wolfram Ecki says likes rdf:object rdf:type rdf:subject rdf:predicate rdf:Statement rdf:Seq rdf:_1 rdf:_2 }
Stratum 1: $\{l_1\}$
Stratum 2: $\{r_1\}$
Now a mapping along the stratification can be performed. First, { [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] } is mapped to (Reinhold Wolfram), the statements are removed from the initial set, and each occurence of l_1 is replaced by (Reinhold Wolfram), leading to the next set of statements (now already extended): { [Gustaf says r_1] [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object (Reinhold Wolfram)], [r_1 rdf:type rdf:Statement], [(Reinhold Wolfram) rdf:type rdf:Seq] }. Next, the set { [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object (Reinhold Wolfram)], [r_1 rdf:type rdf:Statement] } is mapped to [Ecki says (Reinhold Wolfram)] which replaces r_1, resulting in the minimal hull, [Gustaf says [Ecki likes (Reinhold Wolfram)].

This is captured in the following algorithm. It will determine an intention-equivalent extended model, C^E from a set of statements of the RDF model, C^R.

Algorithm Nest(In: C^R)
For stratum $s = 1$ to k **do**
  For all resources r in s **do**
    if r is a reificant **then**
      Remove from C^R the four statements defining the reification
       which has r as a reificant.
      Replace all occurences of r in expressions in C^R by the statement that r reifies.
    if r is a container **then**
      Remove from C^R all statements of the form $[r, \_i, r_i]$ and
       build a list R_r from the resources r_i
      Replace all occurences of r in expressions in C^R by the list R_r
$C^E = C^R$.
Again, the proposition follows from the construction. ⊟

The relation between the RDF model and the extended model relies on the two assumptions. If these assumptions are not accepted as being a reasonable interpretation of the intentions behind the RDF model, then the propositions and proofs given above do not hold. However, the newly introduced model may still be considered as a reasonable, comprehensive alterna-

[12]Note that this can also be a chain of reifications and containers, that is, we consider it to be an error if a container contains a reificant that reifies a statement that contains the container etc.

tive to the RDF model due to its ability to capture complex expressions naturally. We will try to illustrate this by suggesting two alternative representations in the next section, namely a graphical notation and a XML DTD, which both may be considered as advantageous[13] if compared to the alternatives offered in the RDF M&S specification.

4 Graphical and XML Representations of the Extended Model

The following graphical examples and the XML DTD largely follow the presentation in the XRDF discussion paper [4]. Both offer (reversibly mapable) alternatives to the statement/list notation of the extended model.

4.1 The Extended Model as a Graph

The graphical language for the extended model provides the constructs oval (representing resources) and directed labeled arcs (representing relations). Each oval representing a resource of the atom-type has inscribed a content taken from the alphabet A^*. Each embedded oval will be augmented with a number that is unique within the oval it is directly embedded in. Numbers will be left out where possible (ie., in statements, where the ordinal number follows from the direction of the arc, and in lists with one element only). A precise transformation to and from the nested-triple notation of the extended model is straightforward (compare [4]). Some examples of the graphical notation are given in the figures below.

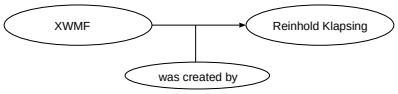


Figure 3: Representing "XWMF was created by Reinhold Klapsing"

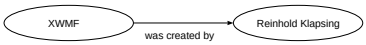


Figure 4: The same statement, now neglecting the fact that the predicate is also a resource (which is the usual way).

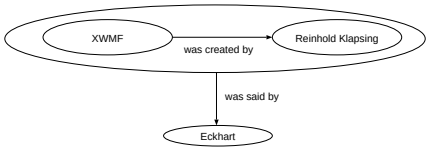


Figure 5: Representing "'XWMF was created by Reinhold Klapsing' was said by Eckhart.", a statement about the previous statement.

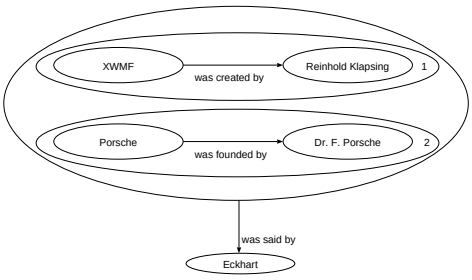


Figure 6: Eckhart made two statements.

It is left to the reader to flaten the graphically represented statement of the extended model to corresponding sets of flat RDF statements. This may suffice to demonstrate that already mildly complex examples of modeling tasks are much more straightforward to formulate (either graphical or in triple notation) with the extended model than with the flat model. Given

[13] Aware: subjective judgement. We hope, however, that some readers may share our opinion.

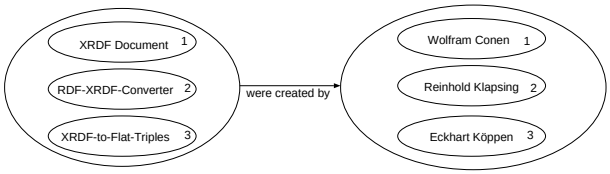


Figure 7: A group of people jointly created a collection of artifacts.

the intention-equivalence of the two models (based on the two assumptions stated above), the extended model seems the more convenient way to express the intentions of set of flat model statement in which reification and containers are used.

4.2 A pure XML syntax for the Extended Model

It is straightforward to represent the (few) ingredients of the nested/list model as an XML-DTD (compare [4] with slightly different list and predicate definitions).

```
<!ELEMENT statement (subject, predicate, object)>
<!ELEMENT list      (statement | atom | list)+>
<!ELEMENT atom      (#PCDATA)>
<!ELEMENT subject   (atom|statement|list)>
<!ELEMENT predicate (atom|statement|list)>
<!ELEMENT object    (atom|statement|list)>
```

A conversion of an XML document that conforms to the above DTD into an extended model is immediate. The algorithm *Flaten* from above gives a conversion to an RDF model. From this, a XML/RDF representation (at least a direct, explicit representation where each statement results in one description) can be derived easily.

Example: The statement [(XWMF was_created_by Reinhold_Klapsing) (Porsche was_founded_by Dr._F._Porsche) was_said_by Eckhart] , compare Figures 6 above, can be “serialized” as follows:

```
<statement>
  <subject><atom>Eckhart</atom></subject>
  <predicate><atom>says</atom></predicate>
  <object>
    <list>
      <statement>
        <subject><atom>XWMF</atom></subject>
        <predicate><atom>was_created_by</atom></predicate>
        <object><atom>Reinhold Klapsing</atom></object>
      </statement>
      <statement>
        <subject><atom>Porsche</atom></subject>
        <predicate><atom>was_founded_by</atom></predicate>
        <object><atom>Dr. F. Porsche</atom></object>
      </statement>
    </list>
  </object>
</statement>
```

There is a number of reasons that make pure XML an attractive alternative to the RDF/XML serialization dialect, we refer the interested reader to the XRDF discussion paper for further details. We will now briefly turn our attention to basic semantic aspects that are closely related to the nested structure of expression of the extended model.

5 Some brief considerations of Semantics

To facilitate the authoring and deployment of meta-data, syntactical and structural simplicity is needed. A layered approach has proven to be useful for the definition of models and techniques (this is especially obvious in the context of XML-based standards, where XML is the basis for other standards like namespaces which in turn are used in the definition of XSLT).With the extended model proposed above, we define the syntax of a lowest layer, make use of the structural primitives statement and list. On top of this basic structural model, semantic definitions and interpretations can be layered. Though this is not the main topic of the paper, we will briefly discuss two aspects that are related to semantical explorations of nested structures.

5.1 Exploring/Propagating Meaning from Outside to Inside

For the task of designing suitable semantics with the extended model we will have to consider a number of design options. We will propose one possible route and point out a few more things that might come in handy. Our route makes use of the following key observation: the semantics related to (sub-)expressions depend on their position within the surrounding structure – that is, the semantics will be explored starting from the outermost part of the structure and proceeding to the innermost part. Let's consider an example that demonstrates a simple kind of truth predicate.

[[sky color blue] hasTruthValue FALSE]]

or, in a flatened version

[r type statement][r subject sky][r predicate color][r object blue] [r hasTruthValue FALSE]

The following transformation and constraints will give the flatened version some meaning in a FOL representation:

Transformation: Map each triple [s,p,o] into an instance of a predicate triple(s,p,o).

Constraints :

(1) reifies(R,S,P,O) ←
triple(R,type,statement) ∧ triple(R,subject,S) ∧
triple(R,predicate,P) ∧ triple(R,object,O).

(2) falsified_resource(R) ←
statement_known_as_true(R,hasTruthValue,FALSE).

(3) statement_known_as_false(S,P,O) ←
triple(S,P,O) ∧ reifies(R,S,P,O) ∧ falsified_resource(R).

(4) statement_known_as_true(S,P,O) ←
triple(S,P,O) ∧ not(statement_known_as_false(S,P,O)).

The statement_ ... -predicates could be used for further inferences. Note that further embedding works out fine also, ie. falsifying falsified statements is possible. The key point here is that the truth of the information contained in a triple will be propagated from the outermost expression to the innermost. This principle can be used to define more sophisticated semantics as well, as would be necessary to give a proper meaning to expressions like

[Reinhold believes [Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]]

It is clear that the actual meaning of embedded expressions depends on the "semantical" scope that is propagated from the outer predicates.

```
Scope 3: (believes  Reinhold
Scope 2:      (assumes  Ecki
Scope 1:            (hasTruthValue FALSE
Scope 0:                  (is Wolfram nice))))
```

What is actually *done* with this information will depend on the proper definition of semantics for predicates and their interaction. With respect to the above example the following question should be answered: what should be the meaning of an elementary statement of which someone believes that someone else assumes that its negation is true. Here, an "elementary" statement can be defined as a statement that has a predicate that does not modulate the truth value of the subject or object (like *is* in the above example). This may suffice to show how the meaning of statements generally depend on their *position*. From the intention-equivalence of extended and RDF model shown above, it follows that this is also true for RDF models – note, that the "position" of a statement in a complex structure is given by its occurence in reifications/lists, for example, the following set of statements flatens the above expression:

[Reinhold believes r_3]
[r_3 subject Ecki] [r_3 predicate assumes] [r_3 object r_2] [r_3 type Statement]
[r_2 subject r_1] [r_2 predicate hasTruthValue] [r_2 object FALSE] [r_2 type Statement]
[r_1 subject Wolfram] [r_1 predicate is] [r_1 object nice] [r_1 type Statement]

Note that there is no need (or better: no use) to "materialize" the intermediate "propositions" like [Ecki assumes r_2], for, if this would be done, an intention-equivalent extended model would contain two statements:

[Reinhold believes [Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]]
[Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]

which is somewhat different from having only the first statement, because now, [Ecki ...] has become a factual statement. It should also be clear from this example that, within the scope of different statements, literally identic subexpressions can have different meaning. This also demonstrates that it is not necessary to give every occurence of literally identic subexpressions an unique identity, because the actual meaning of each occurence depends on its context, which is captured by the position of the subexpression within other expressions.

5.2 Abbreviating Expressions with Structural Transformations

It is possible to provide some kind of syntactic sugar with the help of structural transformations that map a "sugarized" notation to the regular extended model. Some possible transformations are discussed in [4]. This creates the possibility to specify for a predicate which type

of transformation should be performed prior to interpreting the predicate. The transformations can also be applied recursively and can make use of indirection (see example below). This touches upon basic layers of semantics for which an extensive discussion is beyond the scope of this paper. We will therefore only give two brief examples.

The predicate *likes* is defined to be of the transformation type $n \times m$, that is a statement of the form [$(n_1, \ldots, n_k)$ likes $(m_1, \ldots, m_l)$] will be *expanded* to the list of statements ([n_1 likes m_1] ... [n_1 likes m_l] [n_2 likes m_1] ... [n_k likes m_l]). So,

[Wolli likes (Reinhold Eckhart)]

is transformed to

([Wolli likes Reinhold] [Wolli likes Eckhart])

Further assume that a specific predicate, *representedBy*, can be used to give *names* to lists (a similar predicate will exist for statements), like in

[(Reinhold, Eckhart) representedBy Friends]

Now, the type of the predicate*likes* can be adapted to the possibilities of *indirection*, that is if a *name* is encountered in subject or object position, the predicate will not be applied to the name (which is a resource itself) but to resource or list of resources that is represented by that name. We will denote the transformation type of *likes* accordingly as $in \times im$. Now, the following becomes possible:

[Wolli likes Friends]

which will result in the same list of statements as above. Note that this or similar kinds of indirection can be used to cleanly seperate between relations to a resource and to the resources (lists/statements) that may be represented by resources.

This technique of basic transformation that may be applied prior to assessing the complete semantics of predicates, may easily be used to answer the above question:

Assume that *describedBy* is a predicate of the transformation type *descriptive* which takes a list with an even number of elements in object position as an input to the transformation which performs the following:

[Reinhold describedBy (hasName Klapsing hasAddress Essen)]

which will be transformed to

([Reinhold hasName Klapsing] [Reinhold hasAdress Essen])

Whether this kind of transformations should be part of a basic layer of (pre-)semantics certainly remains to be discussed.

6 Discussion

Let us briefly discuss one of the potential problems of upgrading from the RDF model to the extended model: the typing of containers. The issue is that two containers with definitions that refer to the same sequence of elements but with different types (e.g., one Bag, one Seq) become indistinguishable with the above mapping into the extended model. Allow two brief remarks: (1) one solution is to relegate this kind of typing of containers to a schema level.

Together with, for example, the *representedBy* property described in 5.2, names for lists can be introduce and types for the lists represented by the names can be attached to the names etc. (2) Another solution is to drop the typing of containers and to simply regard them all as sequences – and to attach the information how a lists should be treated to the properties that make use of the list (each property can interpret a sequence as a Bag or an Alt construct if this suits the definition of the semantics of this property). More alternatives exist and a solution (an adaptation of the mapping) should be provided when defining a schema level for the extended model

Certainly, more details could be explored and more questions should be asked and answered[14]. However, this may suffice to demonstrate that the nesting of statements and the use of list of statements and resources may allow for a natural representation of useful structures that are cumbersome to model and difficult to use in RDF. Based on the interpretation of reification and containers given above, the RDF model (or, intention-equivalently, the extended model) can be seen as providing a (relatively rich) abstract syntax to build rather complex expressions. This may ease modeling with RDF (respectively with the extended model) and may also provide a more clear-cut syntactic layer for the schema layer(s) to be put on top of this model.

References

[1] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[2] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[3] Sergey Melnik. Slim RDF. Email to the RDF-IG, 12. February 2001. http://lists.w3.org/Archives/Public/www-rdf-interest/2001Feb/0090.html

[4] W. Conen, R. Klapsing and E. Koeppen. XRDF - an eXtensible Resouce Description Framework Discussion Paper, November 2000. http://nestroy.wi-inf.uni-essen.de/rdf/xrdf/

[14] Don't hesitate to contact us if you found your questions unanswered, want to contribute ideas, or simply want to share your opinion on RDF/extended models with us. We thank Graham Klyne and an anonymous reviewer for comments on a previous version of the paper

Toward Semantic Interoperability in Agent-Based Coalition Command Systems

David N. Allsopp, Patrick Beautement, John Carson and Michael Kirton
{d.allsopp, j.carson, m.kirton}@signal.QinetiQ.com, pbeautement@QinetiQ.com

QinetiQ Ltd
Malvern Technology Park
St Andrews Road, Malvern, WR14 3PS
United Kingdom

Abstract

The Coalition Agents Experiment (CoAX) is an international collaboration carried out under the auspices of DARPA's Control of Agent-Based Systems (CoABS) programme. The overall aim of CoAX is to demonstrate how an agent-enabled infrastructure, based upon the CoABS Grid, can enhance interoperability between heterogeneous components, including actual military systems, in a realistic scenario. The scenario is based on a peace-enforcement operation in the year 2012 in 'Binni' (a mythical state in Africa) which requires a mix of Coalition forces to work together.

The agent infrastructure allows the construction of a coalition command support system, with agents grouped into domains to reflect real-world organisational and national boundaries. Each domain is a community of agents that has its own secure communications, capabilities and information spaces, and is governed by policies at the domain, host, virtual machine and agent levels.

Communications between agents, whilst given some minimal semantic grounding via the use of conversation policies, should in future utilise emerging Semantic Web technology. In particular, the Resource Description Framework (RDF) and the DARPA Agent Mark-up Language (DAML) promise to deliver a greater degree of semantic interoperability. This paper describes practical experiences taking initial steps towards this goal, implementing agents that use a query language to exchange data between RDF models and implementing a prototype RDF browser. We discuss the specific requirements for querying, merging and attributing of RDF data by agents in a coalition environment, and the particular restrictions affecting agents on controlled networks, rather than on the Web. Finally, some challenges and requirements for the future are outlined.

1. Introduction

In order to demonstrate how planning, visualisation and execution activities in coalition operations can be augmented by agent technology, a collaborative programme of work has been put together under DARPA's Control of Agent-Based Systems programme (CoABS). This work involves 16 partner organisations, and is entitled the Coalition Agents Experiment (CoAX) [1,2].

This paper begins with an introduction to the challenges of coalition operations and our technical approach to solving them, and gives an overview of the CoAX demonstrations. It then outlines the infrastructure and systems in the current demonstration and the techniques used to integrate them. In evaluating this demonstration, areas that would benefit from Semantic Web technology are noted.

Our initial work on implementing agents that use RDF, including a basic query language and a RDF browser, is then described. This is followed by discussion of the benefits gained from this approach and the limitations of the current technology, in particular the querying and merging of RDF models and the attribution of data via the reification mechanism. The importance of software tools for rapid integration is emphasised. Finally, we note the potential advantages of the emerging DAML language [3] over RDF, and discuss future work.

2. Background

The year is 2012, and climate change in the Sudanese plain of East Africa has enabled the production and export of wheat in large quantities. The only way to transport this increasing volume of food to the European market is by sea. Competition over sea port access has led the government of Gao to launch a pre-emptive strike to open a corridor to the sea, declaring the annexed area to be the independent country of Binni. This action has incensed the government of neighbouring Agadez, who have launched repeated guerrilla activities to dislodge the Gao forces. Because of this dangerously unstable situation in Binni, the UN has passed a Resolution to create and deploy a UN War Avoidance Force for Binni (UNWAFB).

"Binni - Gateway to the Golden Bowl of Africa" [4] is a hypothetical scenario based on the Sudanese Plain. The countries of Gao, Agadez and Binni are fictitious, as are the events, organisations and personalities that lead to the crisis requiring UN intervention.

The vignette used for the CoAX experiment concerns a specific operation in which the UN forces are attempting to keep mutually hostile forces apart long enough to enable peace negotiations (figure 1). Incomplete, changing information, as well as deliberate misinformation from some parties hampers the UN efforts.

2.1 Coalition operations

Coalition military operations will become an increasingly important feature in future years. In any military operation, enabling commanders to have access to timely and relevant information is crucially important to a successful outcome. The difficulties are compounded in the virtual organisation of the coalition since there will be a mixture of equipment, operational procedures, languages, etc. Moreover, there is a pressing need to set up such organisations rapidly in order to respond decisively to emerging crises.

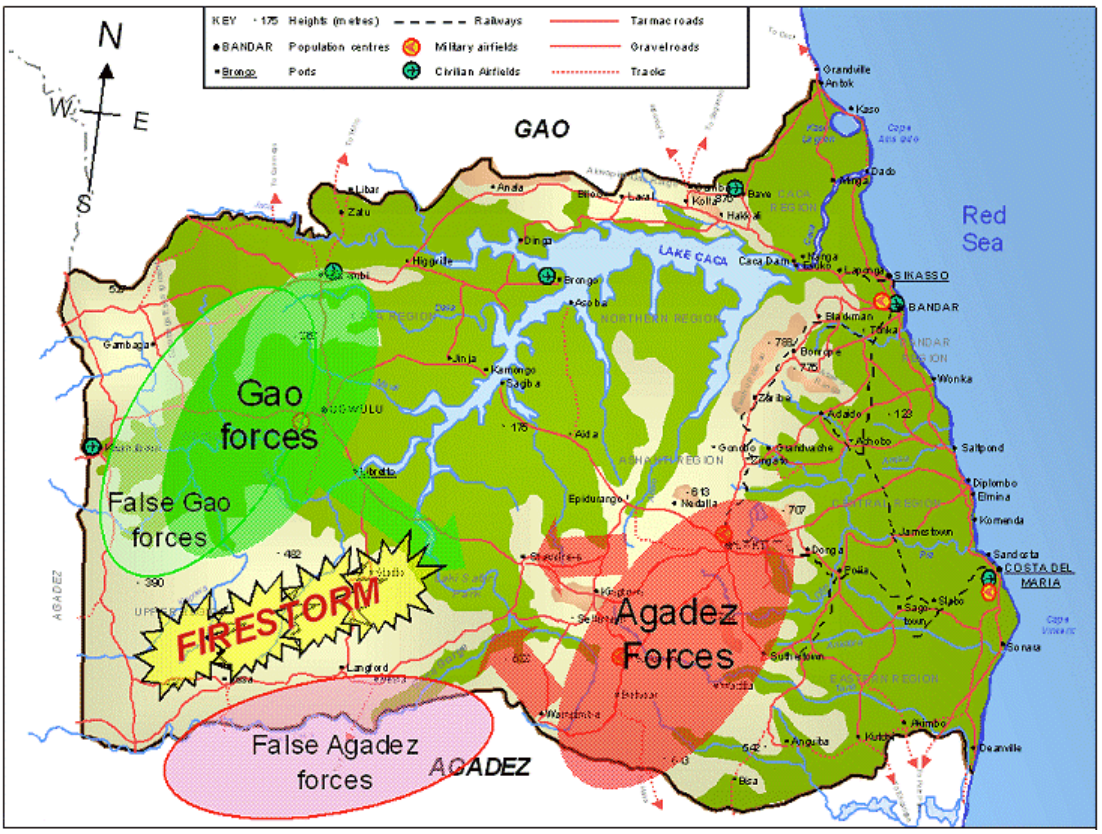


Figure 1. The fictional nation of Binni. The UN forces are considering a controversial firestorm to separate the warring forces of Gao and Agadez. Misinformation from Gao agents initially leads the UN to believe that the forces are further to the west than is in fact the case.

2.2 Technical approach

From a technical perspective, coping with the inherent heterogeneity and tight time-scales are major challenges. Traditional approaches to software integration are too inflexible to share information between such disparate command systems at short notice. Agent-oriented approaches are believed to offer advantages in such environments [5].

The principal motivation of the CoAX experiment is to investigate the conjecture that software agent technology can provide an advanced infrastructure able to support the demanding information, Command and Control requirements of a coalition force [2].

For the purposes of this research, an agent is defined as a software entity acting on behalf of, or mediating the actions of, a human user and having the ability to autonomously carry out tasks to achieve goals or support the activities of the user. Here, agents are supporting a community of human experts; they must not 'take over' or become obtrusive or obstructive. Their purpose is to help people cope with the complexities of working collaboratively in a distributed information environment. Agents operate mostly behind the scenes, integrated into familiar tools and methods of working, linking and fusing disparate sources of information as available — tasks well-matched with Semantic Web technologies. The CoAX project is producing a series of staged demonstrations of increasing complexity, showing agents and agent-

wrapped legacy systems communicating over the CoABS Grid [6], developed at Global InfoTek, Inc (GITI). The Grid is a framework for federating heterogeneous systems. Although the Grid is being developed with a military application in mind, it is a general-purpose agent framework with potential use by a variety of applications.

In recent demonstrations, agents are grouped into domains, using the Knowledgeable Agent-oriented System (KAoS) from Boeing and the University of West Florida's Institute for Human and Machine Cognition (IHMC) [7]. This system enables domain policies to be changed at runtime; for example, to cut off communications with a domain containing hostile agents or to drastically reduce CPU and network resources to agents launching a denial-of-service attack.

The current demonstration at the time of writing has grown to 25 agents grouped into 6 domains; elements of this demonstration are outlined below. The full demonstration (documents, images and video) may be seen in detail on the CoAX web-site [1]. The demonstration is genuinely heterogeneous, comprising systems and agents from at least 6 different organisations (including two real military systems) with more to be added in future demonstrations for 2001 and beyond.

3. Current demonstration

3.1 CoABS Grid Infrastructure

At the most basic level, the agents and systems to be integrated require infrastructure for discovery of other agents, and messaging between agents. This is provided by the CoABS Grid. Based on Sun's Jini services [8], the Grid allows registration and advertisement of agent capabilities, and communication by message passing. Agents can be added or removed, or their advertisements updated, without reconfiguration of the network. Agents are automatically purged from the registry after a short time if they fail. Multiple lookup services may be used, located by multicast or unicast protocols.

In addition, the Grid provides functionality such as logging, visualisation, encryption and authentication.

3.2 Knowledgeable Agent-oriented System (KAoS)

At a higher level, the KAoS framework is used to group agents into domains, to facilitate policy administration. Agents in a domain are subject to the policies of that domain. A given domain can extend across host boundaries and, conversely, multiple domains can exist concurrently on the same host. Policies can be scoped variously to individual agent instances, agents of a given class, agents running on a given host or instance of a platform (e.g., a single Java VM), or agents in a given domain or sub-domain [7]. The KAoS Policy Administration Tool (KPAT), a graphical user interface to domain management functionality, has been developed to make policy specification, revision, and application easier for administrators without specialized training.

The concept of policy-based management necessarily extends beyond typical security concerns. For example, KAoS policies will ultimately be used to represent not only *authorization, encryption, access* and *resource control* policies, but also *conversation policies, mobility policies, domain registration policies*, and various forms of *obligation policies*. The

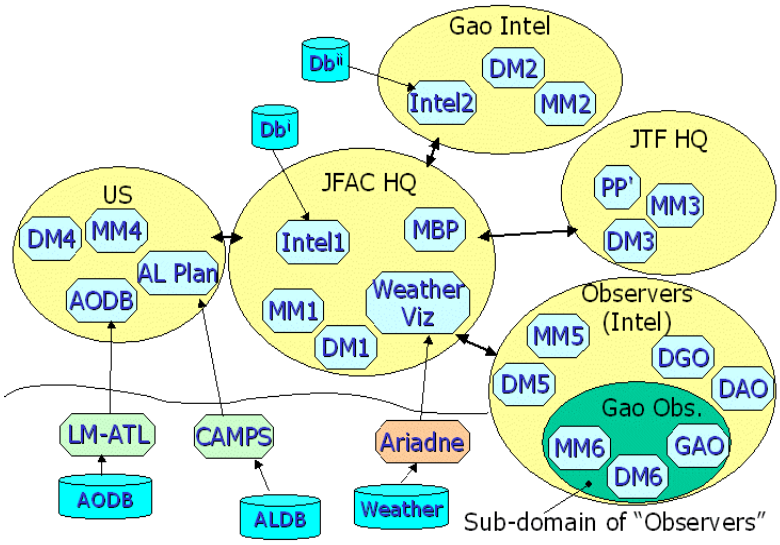


Figure 2. Agents are shown grouped into domains. Each domain contains two specialised agents: the Domain Manager (DM) and Matchmaker (MM). There are domains for countries (the US, and the fictional country of Gao), for coalition structures (such as the Joint Forces Air Component HQ), and for functional groups (Observers). Some agents (shown below the line) are not domain-aware, but have proxies within the domains. Various agents are associated with databases (DB). Other agents (MBP, Intel, CAMPS, WeatherViz, PP) are described below.

KAoS policy representation is currently very simple but an implementation of a more sophisticated DAML-based policy representation will be available later this year.
The domain structure of the current demonstration is shown in figure 2. Each domain contains two specialised agents, the Domain Manager, which enforces (directly or indirectly) the domain policies, and the Matchmaker, which provides functionality similar to the Grid registry.

KAoS also provides support for defining and using *conversation policies*: the structuring of messages for particular purposes or speech acts, such as Inform, Query, Request and others. Each basic policy forms a finite state machine, where each message, labelled with a verb identifying its purpose, represents a transition between states [9]. More general constraint-based DAML policy representations and mechanisms incorporating additional communications aspects, such as time limits, and the ability to compose policies from smaller fragments, are under development [10].

3.3 Systems integrated

The demonstration, at the time of writing, includes 25 agents from about six different organisations, grouped into six domains, written using three different programming languages. Some of the main elements are described briefly below.

3.3.1 Master Battle Planner (MBP)

A core agent in the demonstration is MBP – a highly effective visual planning-tool for air operations. MBP assists air planners by providing them with an intuitive visualisation on which they can manipulate the air intelligence information, assets, targets and missions, using a map-based graphical user interface (figure 3).

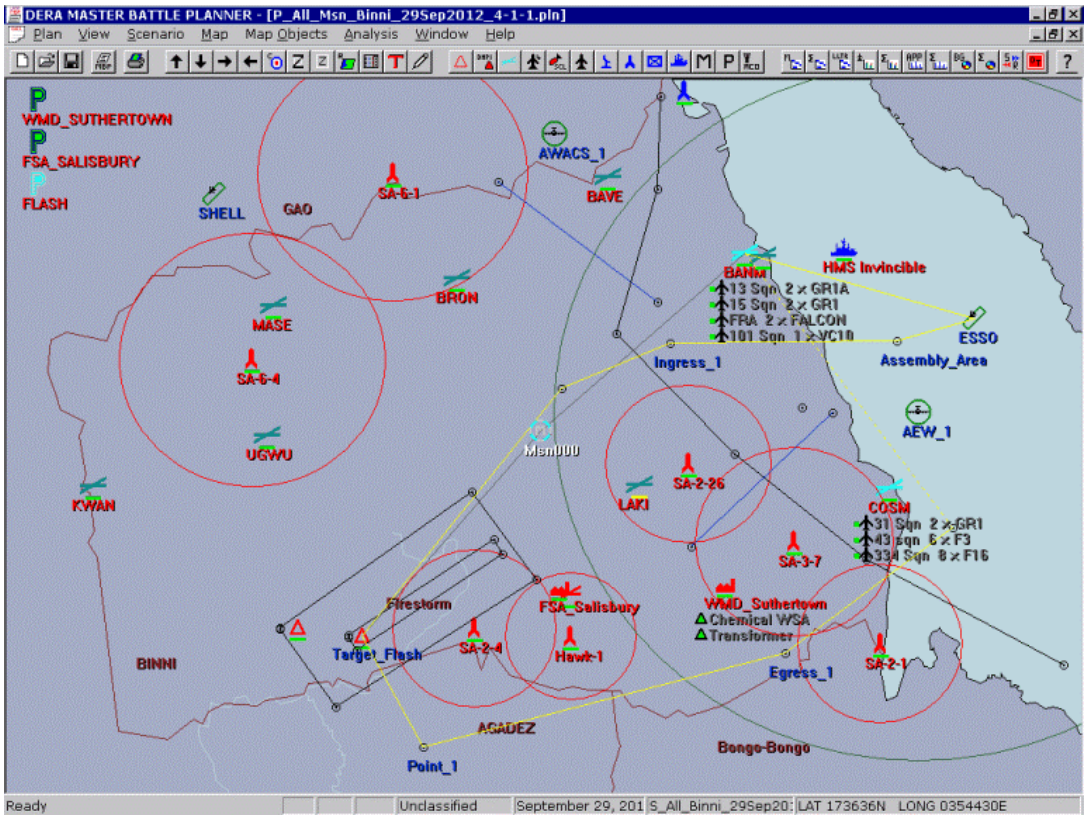


Figure 3. Master Battle Planner map display of the fictional countries of Binni, Gao and Agadez. A selected mission is highlighted in yellow; proceeding from an airbase, to refuelling tanker, via waypoints and airspaces to the target, and back to base by a different route.

The operator can interact with these operational entities and can plan individual air missions (or complex packages of missions) by dragging and dropping offensive units onto targets on the map. Supporting / defensive elements are added in the same way. The system provides the operator with analytical tools to assess the planned air operations.

MBP is a monolithic C++ application, which has been agent-enabled by wrapping it in Java code, using the Java Native Interface, and providing a proxy agent which communicates with the wrapper using the JavaSpaces API [11]. The agent enabling of MBP allows it to receive scenario data (targets, assets, airspaces etc) from other agents (Intel, figure 2), and update this information continuously. Information concerning other air missions can be accepted and merged with missions planned within MBP; export of mission data to other agents is under development.

3.3.2 Consolidated Air Mobility Planning System (CAMPS)

The second real military system integrated into the demonstration is Air Force Research Laboratories' CAMPS Mission Planner. CAMPS develops notional schedules for aircraft to pick up and deliver cargo within specified time windows. It takes into account numerous constraints on aircraft capabilities, port capabilities, etc. [12–14].

In the demonstration scenario, CAMPS schedules airlifts of cargo into Binni. These airlift flights could potentially conflict with offensive air missions, so the scheduled flights are requested from the CAMPS agent, translated by another agent, and sent to MBP, forming part of

the normal MBP air visualisation.

This is an interesting example, as only partial translation is possible; CAMPS and MBP differ fundamentally in their definition of air missions. A CAMPS mission consists of an arbitrary collection of flights, where a flight is a single journey from A to B by a single aircraft. However, an MBP mission consists of a starting point and a route, *which must return to the starting point* (perhaps by a different path), and may consist of multiple aircraft. CAMPS can therefore produce routes that have no fully valid representation in MBP, although they could be partially represented or indicated graphically. This is a fundamental limit on the achievable degree of interoperability.

3.3.3 Ariadne

In a similar manner, weather information extracted from web-sites by the Ariadne system from USC/ISI is gathered by the WeatherViz agent (figure 2). It is translated and forwarded to MBP, again forming part of the normal picture of the air situation. Ariadne facilitates access to web-based data sources via a wrapper/mediator approach [15,16]. Wrappers that make Web sources look like databases can be rapidly constructed; these interpret a request (expressed in SQL or some other structured language) against a Web source and return a structured reply. The mediator software answers queries efficiently using these sources as if they formed a single database. Translation of the XML from Ariadne into the XML expected by MBP was initially handled by custom code, but can now be performed more easily using XSL Transformations [17].

3.3.4 Process Panel

The Artificial Intelligence Applications Institute (AIAI) of the University of Edinburgh has provided its IP^2 Process Panel agent (PP, figure 2), which provides user level, configurable task and process management aids for inter-agent co-operation [18]. Each user may have their own panel to reflect their role in a co-operative process. The Process Panel keeps track of the air planning process through inter-agent messages.

3.3.5 NOMADS agents

The NOMADS mobile agent system from IHMC is used in the demonstration to allow untrusted agents (Gao Observer) to run in the Observer domain alongside trusted agents (DGO, DAO, figure 2). The Aroma virtual machine provides dynamic resource control mechanisms, protecting the host from a malicious or buggy agent [19]. When a denial-of-service attack is mounted by an agent from Gao, the excessive usage of CPU, disk and network is detected and a change of policy is automatically executed, in concert with the KAoS domain management mechanisms. A human operator can then choose to lower the resource limits even further using KPAT.

3.3.6 Future additions

Future demonstrations for 2001 and beyond have introduced further agent systems and capabilities. These include a multi-level co-ordination agent from the University of Michigan [20]; a field observation system using Dartmouth College's D'Agents technology [21]; and eGents (agents communicating over e-mail) from Object Services and Consulting, Inc [22]. This demonstration extends into the execution phase of coalition operations, showing near real-time

visualisation of air operations based upon data from agents.

4. Evaluation of demonstration

4.1 Aims and achievements

The aim of this demonstration was to investigate how software agent technology can provide an advanced coalition infrastructure. We were successful in integrating a variety of real military systems and agents, and were able to demonstrate increased functionality and interoperability between previously stand-alone systems. The systems were written separately by different organisations, in different languages, under different operating systems. Dynamic data sources were an important feature, removing the reliance on static data files. Agents also integrated data from disparate systems seamlessly as far as the user was concerned; weather and airlift missions were merged into the MBP view of air operations, for example.

Agent enabling of real legacy systems via wrappers was found viable, even where the original system made no allowance for integration with other systems.

Using KAoS, the ability to group agents into domains was demonstrated, and to change the domain policies dynamically; for example, to cut off communications with a domain containing hostile agents. Using KAoS domain management tools in conjunction with the Aroma virtual machine also allowed dynamic resource limits to be applied to individual agents to prevent denial-of-service attacks.

4.2 What is missing?

However, the vision of the CoAX project is to achieve rapid, dynamic coalition formation, in which agent domains are created and removed on-the-fly, and agents come and go. Interoperability between agents should be achieved very rapidly, with as little human intervention as possible. Agent interoperability *was* achieved relatively easily, due to the discovery and messaging facilities provided by the CoABS Grid, but message structures, primarily in XML, were pre-agreed, or translators were hand-coded. This process is time-consuming. Tools such as XSLT can accelerate the process (and for rapid integration, graphical tools based on languages such as XSLT may still be valuable) but are still human driven. A fundamental problem for such agents is that there is no mechanism for sharing terms and relations (via shared or partially shared ontologies). Consequently, messages have no unambiguous meaning (even to humans) outside of the agent that generates them. XML provides shared syntax, but not shared meaning.

4.3 How can Semantic Web technology help?

Berners-Lee *et al* write: [23] "Some low-level service-discovery schemes are currently available, such as Microsoft's Universal Plug and Play, which focuses on connecting different types of devices, and Sun Microsystems's Jini, which aims to connect services. These initiatives, however, attack the problem at a structural or syntactic level and rely heavily on standardisation of a predetermined set of functionality descriptions. Standardisation can only go so far, because we can't anticipate all possible future needs. The Semantic Web, in contrast, is more flexible."

The CoABS Grid infrastructure used in this work is based upon Sun’s Jini, but

requires only *one* standard interface – the 'AgentRep' which provides inter-agent messaging functionality, with arbitrary message content. It is therefore possible to combine the discovery services of Jini and the messaging, security, logging and other services of the CoABS Grid with machine-understandable semantics by writing messages in emerging Semantic Web languages such as the Resource Description Framework (RDF) and the DARPA Agent Mark-up Language (DAML).

5. Initial work

The aim of this initial work was to implement simple agents using a Semantic Web language to exchange data about the demonstration scenario. Practical experience with RDF and RDF Schema will assist in understanding the issues involved and assessing the current technology.

5.1 Special features of the Coalition domain

There are some differences between the Web and the networks expected in a command information system.

Hendler [24] predicts that on the Web we will not see large complex consistent ontologies, carefully constructed by expert AI researchers, and shared by a great number of users. Rather, we will see a great number of small ontological components largely created of pointers to each other and developed by Web users in much the same way that Web content is currently created. There has been little work so far on developing explicit ontologies for coalition operations; we expect that the situation will improve as benefits from the initial Semantic Web technology are realised.

A coalition network could perhaps be regarded as being 'on the edge of the Web', consisting of multiple fire-walled networks with guarded portals between them, and between them and the Web itself. A greater degree of control will be present. Medium size ontologies would be expected, constructed with care by individual coalition members and groups of members, but not all directly interoperable or consistent with each other. The challenge is to map between them at short notice. In a best-case scenario, many of these ontologies would use standard ontology libraries developed over time for common domains. Complete mapping is not always possible, as noted in the description earlier (section 3.3.2) of mapping data between MBP and CAMPS. There is a need for techniques and tools to handle this, perhaps detecting and flagging the problematic elements for human attention. Hendler [24] notes that there are many possible techniques to map between ontologies, and that this is an interesting challenge for the future.

Some of the initial interest in Semantic Web technology has focussed on the mark-up of conventional web pages with semantic metadata for improved search engines and web-crawling agents. In a coalition system, some data may be utilised in this way, but the main focus is on inter-agent messages expressed directly in RDF or another Semantic Web language. It is important to note that not all agents in a command system will have direct access to the WWW, for obvious security reasons.

5.2 Resource Description Framework (RDF)

Our initial work was based on RDF and RDF Schema (RDFS), following the initial W3C

Recommendation and the release of a number of parsers, APIs and frameworks for RDF. Due to the time scales of this project, it was essential to investigate technologies available now, as well as potential for the future. We regard RDF as an initial test-bed for investigating Semantic Web issues, and a stepping-stone to future technologies.

DAML promises a far greater level of power but is still at an early stage of development as far as tools are concerned. For rapid coalition integration, effective tools are the essential requirement. A powerful and expressive language is not useful in practice until it can be written, modified and applied rapidly, by those without expertise in logic: "A crucial aspect of creating the Semantic Web is to enable users who are not logic experts to create machine-readable Web content" [24].

There is also an interesting correspondence between RDF and DCADM (Defence Command and Army Data Model), the UK Ministry of Defence's preferred – and indeed in many cases, mandated – solution [25]. DCADM is a combination of two technologies. The first is an innovative immutable datastore. In most datastores when a record is modified the old version is overwritten by the new version. In the DCADM immutable datastore, both versions are preserved. This has considerable potential advantages in the sort of Command and Control systems that are envisaged as the primary applications of DCADM. The datastore can support multiple competing values, reflecting the uncertainty factor present in the 'fog of war', and it maintains a complete audit trail of who changed what values and when.

The second component of DCADM is a metadata model that can be used to describe the data models that form the basis for interoperability. In this respect, DCADM and RDF provide essentially equivalent facilities. Data models developed in DCADM are readily translated into RDF, and vice versa.

We have implemented simple agents, running on the CoABS Grid inside KAoS domains, which can store and query RDF databases. They make use of basic RDF Schema ontologies for defining the class and property hierarchy and the range and domain constraints on properties and their values. The entities in the scenario fall into a number of superclasses, such as mobile or fixed objects, natural features or man-made installations, locations, airspaces, or activities. Basic constraints apply: physical objects possess a location; vehicles can have a speed; airfields have runways. RDF Schema cannot however express many other features of the domain; even the fact that friends are disjoint from enemies.

The agents need to acquire information from other agents, and from their own databases. We have therefore implemented a simple RDF query language and query engine with an SQL-like syntax, similar to other recent query languages [26,27]. Most RDF query language proposals appear to be client-server oriented, assuming fast access by the client to a local database in-memory or on disk. However, with a mainly peer-to-peer model, where access via messages over a network may be slow, special features may be required.

As an alternative to returning individual values of properties matched by a query, the query engine can return the sub-graph of triples matched by the query, as a complete RDF document, using a query of the form:

```
select triples where <constraint list>
```

This allows the returned data to be directly parsed and merged into an agent's database. Returning complete RDF documents in this way supplies the context necessary when using asynchronous messaging; if individual context-free values were returned they would have to be somehow matched up with large numbers of outstanding queries.

Queries may be formed which return the sub-graph accessible from a specified resource, so an agent can ask for everything known about a resource in a single query, using the form:

```
select reachable where <constraint list>
```

Without this feature, a potentially very large number of queries would be required.

Reading or editing raw RDF syntax is difficult, and rapidly becomes impractical as the number of triples increases. In the process of developing the experimental schemas, a prototype RDF browser has been implemented. This allows the user to seamlessly navigate through both the schema and data, searching for resources either by type or via the query language. All resources are clickable hotlinks, and a history is kept, allowing navigation of an RDF graph in the now-familiar style of a Web browser. Namespaces are automatically abbreviated to namespace prefixes, e.g. 'rdf:type'. In figure 4, scenario data from the CoAX demonstration is being queried for resources of a specified type; the properties of a selected resource can then be viewed. In figure 5, the classes in the corresponding RDF Schema are being examined; super-classes and properties related to the selected class are shown.

This browser was not intended to replace dedicated ontology editors, but to provide tighter support for the specific features of RDF, to handle data and schema seamlessly, and handle large numbers of instances. Some ontology editors do not support features of RDF such as the sub-property hierarchy, global properties, and multiple domains and ranges for properties; in general they cannot handle arbitrary RDF. The aim is to facilitate the creation and examination of models without having to hand-code RDF in XML; there is a lack of mature tools for this at present.

Desirable enhancements to the browser include:

- support for reification and containers (creation, and checking of imported data for all 4 properties of a reification quad; checking and repairing collection properties (_1, _2,...) for correctness; filtering data by origin and timestamp;
- enhanced namespaces support including filtering by namespace;
- explicit support for properties (*subPropertyOf* relationship, range and domain);
- editing support (selection of sub-graphs and multiple nodes; clean deletion of entire objects, collections, and reified statements);
- graphical display of portions of an RDF graph.

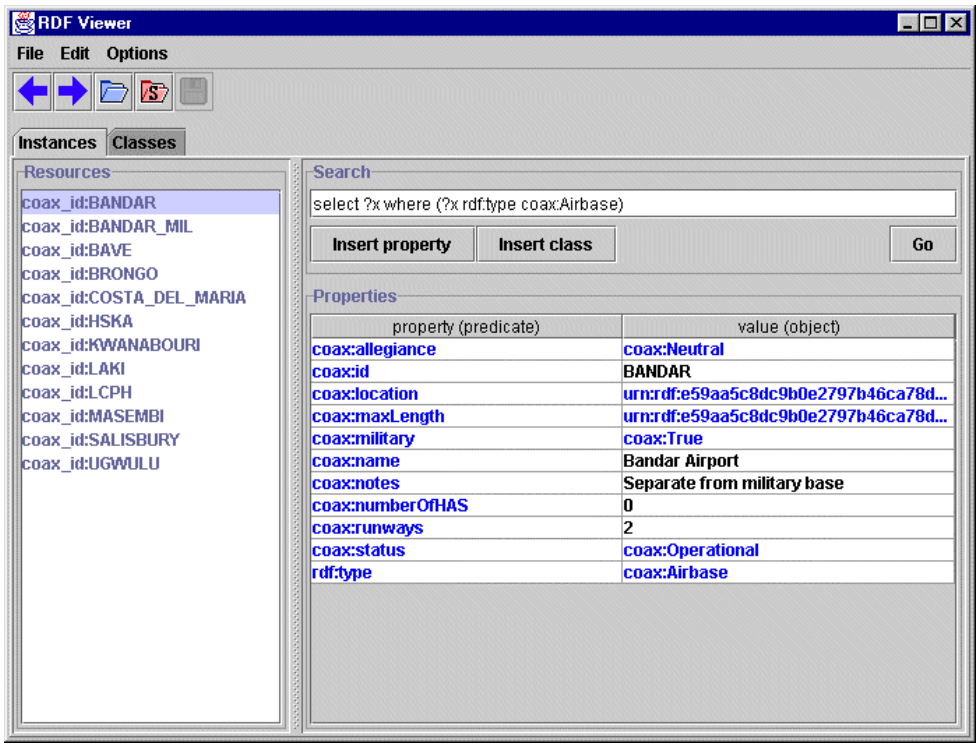


Figure 4. RDF browser. A search for resources of a specified type has been performed; these resources are listed on the left. All the properties of the selected resource (Bandar Airbase) are shown – these form hotlinks that can be explored further by clicking on them. Literals are not hotlinks, of course; these are shown in black.

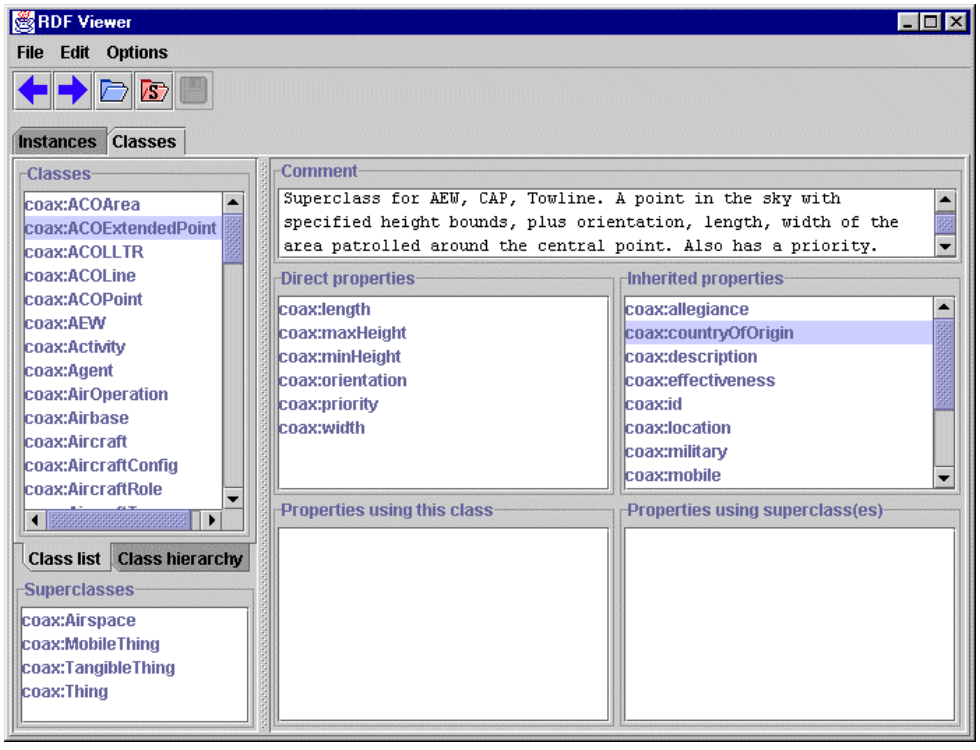


Figure 5. RDF Browser class display, showing superclasses, properties of the class (both direct and inherited) and usage of this class by other properties.

6. Discussion

6.1 Advantages of RDF

RDF provides a formal data model for representing entities (resources) and the relations between them, and a syntax for XML serialisation of data models. RDF describes directed labelled graphs, rather than just labelled trees as in XML. The use of RDF is a move away from inflexible XML data structures and DTDs, allowing more natural handling of partial data, which inevitably occur in uncertain military environments. For example, a system that represents an entity such as a radar installation may store many attributes such as location, allegiance, range, type and so on; yet if a radar is detected electronically at a great distance, we know very little about it and cannot 'fill in all the blanks' of a database form or XML DTD. Rather, data arrive gradually from disparate sources and must be merged together. Partial data records or changes in document structure can be problematic when handling XML, although more recent XML technologies such as XPath and XQuery have increased flexibility to some extent. These advances also counter the size and complexity of code previously required to handle XML trees.

RDF Schema adds a very basic ontological framework, although it appears from the many discussions on the RDF mailing lists that the semantics of RDF and RDFS are unclear in the current specifications [28]. This may be hindering the development of systems with formal semantics on the top of RDF, such as DAML. Very basic inference is possible using the RDFS class and property hierarchy definitions, and property restrictions [24]. This is already used to some extent by current query engines.

Although this seems trivial, it offers a substantial improvement over the keyword searches that Web users suffer at present, and a qualitatively different capability from semantics-free XML data. In many cases, the level of power needed to achieve significant benefits in interoperability is quite small, and so RDF and RDF Schema are useful in themselves. For example, in this coalition scenario, it is easy to search for data about any damaged friendly ground units, without needing to know their exact type. One could also learn their exact types (subclasses of the class *GroundUnit*) in the same query. Simple constraints on properties would also allow queries to specify particular regions in space or time. A flexible query interface to agents creates possibilities for all kinds of useful (and perhaps unforeseen) interactions between systems, in contrast with rigid one-to-one message links.

6.2 Current limitations

The growing industry support for XML points to an XML-based solution for semantic mark-up, yet RDF is verbose and not easily readable or writeable by humans. ‘Notation 3’ [29] is an interesting experiment with a more concise syntax (with additional logic elements). Authoring and processing tools would of course make the syntax less of an issue.

For rapid integration of agents, tools are the emphasis, not languages. A more powerful language is of no use in this domain if it cannot be deployed easily. Tools and APIs are needed throughout the lifecycle of the data – creation, checking, storing, querying and inference, mapping and converting.

Judging from the large volume of debate on the RDF mailing lists, the development of such tools appears to be suffering delays due to the substantial number of unresolved issues concerning the RDF and RDF Schema standards, and their interpretation [28].

A practical problem encountered is that there is an assumption in some APIs and parsers that schemas will always be loaded over the Web from their home site (rather than being cached) which is not realistic on a coalition network.

6.3 Querying requirements

A query language allows agents to seek and exchange information, especially when enhanced by a schema/ontology, but of course we rely on agents understanding the same query language, and there is no standard query language for RDF.

A standard language is of less importance when querying local databases or Web documents; any suitable language could be used. In a community of heterogeneous agents there is a need to talk to other agents in a common tongue, or at least translate down to some 'lowest common denominator'.

A variety of languages have been developed [26, 27, 30–33] with differing capabilities. Many do not yet deal with some of the more controversial aspects of RDF such as reification and collections. Kokkelink [34] has suggested work on standard RDF querying by attempting to re-use previous experience with XML in the development of XPath, XQuery and XSL Transformations.

In the absence of a standard query language, a simple technique for gaining rapid interoperability at the expense of increased messaging load is the use of a remote model interface. The majority of current RDF frameworks allow an RDF graph (model) to be queried using a single triple where each of the subject, predicate and/or object can be a wildcard. This can be regarded as the lowest common denominator for querying. Instead of sending complex queries for a remote agent to process, the work could be done locally, sending multiple messages invoking only the basic query method of the remote agent. As far as the initiating agent is concerned, the process is the same as querying a local (albeit very slow) model database (figure 6). Primitive agents can be made to support complex queries, and agents using differing query languages can be rapidly integrated, although the load on the network will greatly increase.

One could also send a mobile agent implementing complex queries to a remote host, interrogate agents there using low-level messages, and return the results. This could reduce the network load but requires more sophisticated infrastructure, especially for security issues.

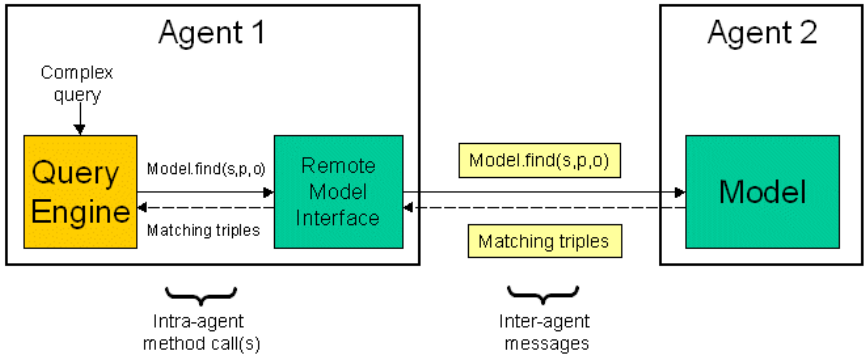


Figure 6. Use of a remote model interface, allowing Agent 1 to perform complex querying of the more primitive Agent 2, which only supports matching of single wild-carded triples.

6.4 Merging models

A fundamental process in a command information system is the collection and fusion of data from many sources. Merging data from multiple RDF sources (agents) into a single model raises several issues. Firstly, the identification of redundant anonymous resources. Anonymous resources are usually assigned locally unique identifiers by the parser; if the data in an RDF model originates from multiple documents, the same (anonymous) resource in different documents will normally receive two different identifiers. For example, figure 7 shows data obtained from three separate sources, which require merging.
Unfortunately, it does not seem possible to determine in general whether the duplication is intended, and therefore whether the redundancy can safely be eliminated. Cardinality restrictions, available in DAML, could help to resolve this issue. For example, in RDF, either of the graphs in figure 8 might be chosen. Can John Smith only have one weight? It might initially appear so, but suppose the data track his weight during a diet! Each weight would then be valid for a particular point in time, and could possess equivalent values in various units.

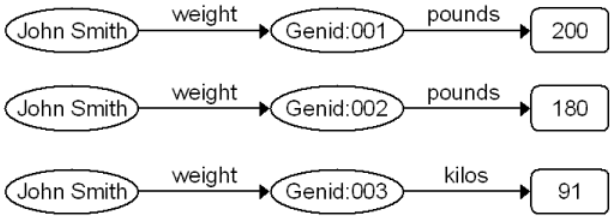


Figure 7. Three RDF data models which we are attempting to merge. The anonymous middle nodes are assigned a locally unique identifier by the parser, but we cannot determine unambiguously whether they are in fact the same resource.

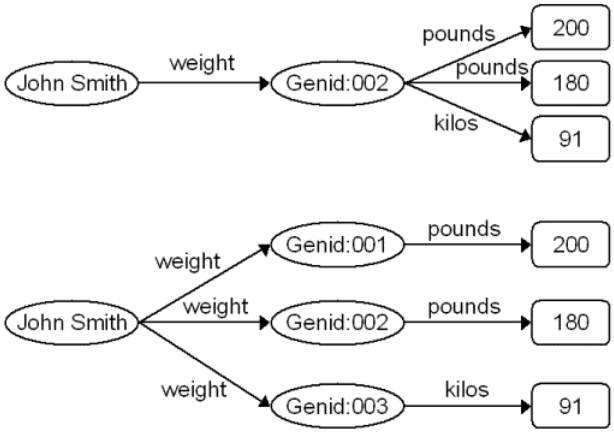


Figure 8. Two possible results of merging the models in figure 7.

Secondly, merging of RDF *Bags* or *Alternatives* cannot be achieved without renumbering all the member attributes (*rdf:_n*) of one of the collections. Merging of *Sequences* does not necessarily make sense in general. Other problems with the RDF containers have been identified [28]; for example, adding an element to an RDF *Bag* requires one to know all the existing elements.

6.5 Attribution

As mentioned earlier, it is valuable for an agent operating in an uncertain environment to store multiple competing values, and maintain an audit trail of changes. As levels of trust and other factors change, the most reliable or up-to-date data can be selected. In a military coalition context this ability is essential.

Reification provides part of the solution; we can reify statements and attach additional data (timestamp, origin, trust values) to them. In the example above, it would then be possible to determine that John Smith weighed 200 pounds (91 Kg) last year but only 180 pounds last week, although the initial value was measured by his doctor and the recent value is his own optimistic claim! However, once the criteria for selecting competing data values have been chosen, we still need to query the data as easily as if it were 'flat'. Most current APIs and query languages do not appear to provide a solution yet. Reification is a contentious issue in RDF with a number of problems identified. The concept of contexts has been introduced as an improvement [28,35].

In practical cases, it may be that no decision can be made automatically; to assist human decisions, there is a need for ways of visualising the changes in data according to time, or source.

6.6 DAML versus RDF(S)

The case for DAML is strongly made in several recent papers [24,36], and centres on DAML's greater expressive power and formal semantics. RDF(S) is not considered to provide sufficient expressive power for many applications.

DAML+OIL, an ontology language based upon description logics and encoded in RDF, provides a rich set of constructs not available in RDF(S). In addition to defining classes and properties, one can express the disjointness or equivalence of classes and a variety of restrictions on the usage of classes and properties, such as cardinality. Additional information can be expressed about properties, for example stating that they are transitive. New classes can be constructed by taking the complement of another class or the intersection of other classes. A Property can be declared as the inverse of other properties. In recent versions, DAML+OIL has also been integrated with the XML Schema datatypes. A logical language, DAML-L, is under development.

There are some incompatibilities between RDF(S) and DAML. Firstly, the semantics of *domain* and *range* are different from those in RDF Schema. Secondly, the RDF Schema specification demands that the subclass relation between classes must be acyclic. DAML+OIL deliberately has no such restriction. Finally, parsers based on the current RDF specification will not support the daml:collection parse type, although a pre-processing stage can overcome this problem. The DAML collections address the previously mentioned problems with RDF Bags (section 6.4).

6.7 What next ?

Initial proof-of-concept contributions to the CoAX experiment will involve replacing some of the XML communication links with RDF to compare the relative complexity and robustness of the code. The simple agents used for this will provide a test-bed for showing the increased flexibility of RDF(S), for example assembling partial data from several sources to form a complete description, and queries assisted by the class hierarchy defined in the schema. Agents supporting a common query language, or a remote model interface, are far more accessible to other agents, greatly increasing the interaction possibilities.

We also wish to explore the use of RDF Reification or Contexts [35] for tagging data with their origin, reliability, timestamp etc. This is obviously of crucial importance in a command information system where it is essential to be able to roll back or exclude false or inaccurate data. There is much scope here for work on tools to visualise the data, providing different views of the data filtered by time, origin, etc.

A further step would be the interoperation of agents without fully shared terminology, via translator agents. There are interesting practical issues in handling the agent conversation policies with interruptions to request translation of terms.

Further steps, such as more expressive ontologies and the use of inference rules, move into the territory of DAML; we expect that emerging DAML tools and the DAML-L logic language will provide us with an opportunity to explore these capabilities. As previously noted, we are collaborating with IHMC on specification and implementation of a DAML-based policy representation (KAoS Policy Language, or KPL), which will used to represent both simple atomic policies (e.g., Java permissions) and complex constructions. Representation of both authorizations (i.e., permitting, permitting with qualification, or forbidding some action) and obligations (i.e., requiring some action to be performed) will be possible. We expect that an initial specification of KPL will be available later this year.

7. Conclusions

Although RDF and RDF Schema have a number of flaws, they are useable for a variety of applications and for agent experimentation. In the context of CoAX, for example, they allow richer interactions between agents, and more useful and complex queries. Thus, RDF and RDF Schema still offer interesting possibilities and can contribute significantly to agent interoperability in a coalition setting. Alongside the development of languages, tools and agents, expertise in constructing ontologies for coalition operations will be needed. Some of the flaws will be addressed by the emerging DAML family of languages; others relate to the RDF data model and syntax; these need to be addressed. There is also a profusion of RDF query languages, although work toward a standard has been proposed.

DAML adds the formal semantics and expressiveness demanded by the logic community, but is incomplete as yet; we look forward to future developments, particularly in the area of tools for creating, parsing, checking, querying and reasoning with the languages.

One of the greatest challenges ahead lies in the development of mechanisms for merging or mapping between ontologies automatically, as far as possible, and fusing data from disparate sources, enabling heterogeneous agent systems to interoperate rapidly and effectively. It is important to acknowledge that any fundamental differences between systems (such as described in section 3.2.2 will always be a barrier to complete interoperability.

Acknowledgements

QinetiQ work was carried out as part of the Technology Group 10 of the UK Ministry of Defence Corporate Research Programme. We acknowledge the contributions of the following CoAX partners:

- Ariadne web wrapper/mediator (University of Southern California, Information Sciences Institute)
- Consolidated Air Mobility Planning System (CAMPS) (Global InfoTek, Inc / BBN Technologies / Air Force Research Laboratories).
- Knowledgeable Agent-oriented System (KAoS) Policy-Based Tools and Framework (University of West Florida, Institute for Human and Machine Cognition / Boeing).
- EMAA / CAST agents (Lockheed Martin Advanced Technology Laboratories).
- NOMADS Mobile Agent System (University of West Florida, Institute for Human and Machine Cognition).
- Process and Task Management tools (University of Edinburgh, Artificial Intelligence Applications Institute).
- The DARPA CoABS Grid (Global InfoTek Inc, ISX Corporation).

We would like to thank our QinetiQ colleagues Chris Walker and Don Brealey for provision of the Master Battle Planner (MBP) tool for Air Battle Planning, and Sergey Melnik of Stanford University for provision of RDF API software.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of the UK MoD, DARPA, the Air Force Research Laboratory, the US Government, the University of Edinburgh or the University of West Florida.

References

[1] Coalition Agents eXperiment: http://www.aiai.ed.ac.uk/project/coax/

[2] David Allsopp, Patrick Beautement, Jeffrey M. Bradshaw, John Carson, Michael Kirton, Niranjan Suri and Austin Tate, *Software agents as facilitators of coherent coalition operations*, 6th International Command and Control Research and Technology Symposium, 19-21 June 2001, US Naval Academy, Annapolis, MD, USA.

[3] DARPA Agent Mark-up Language: http://www.daml.org/

[4] R. A. Rathmell, *A Coalition Force Scenario 'Binni – Gateway to the Golden Bowl of Africa*', Proceedings of the International Workshop on Knowledge-Bases Planning for Coalition Forces, (ed. A. Tate) pp. 115-125, Edinburgh, Scotland, 10-11 May 1999.

[5] Nicholas R. Jennings, *An agent-based approach for building complex software systems*, Communications of the ACM, April 2001/Vol. 44, No. 4, pp 35-41.

[6] CoABS Grid: http://coabs.globalinfotek.com/

[7] Jeffrey M. Bradshaw, Niranjan Suri, Martha Kahn, Phil Sage, Doyle Weishar and Renia Jeffers, *Terraforming Cyberspace: Toward a policy-based grid infrastructure for secure, scalable, and robust execution of Java-based multi-agent systems*, Proceedings of the Workshop on Agent-based Cluster and Grid Computing, IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia, 14-18 May, 2001. (Enlarged version in IEEE Computer, July 2001, pp 48-56).

[8] Jini: http://www.sun.com/jini/

[9] Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit and John D. Woolley, *KAoS: Toward an industrial-strength generic agent architecture*, In J. M. Bradshaw (Ed.), *Software Agents*, 1997, pp. 375-418, Cambridge, MA: AAAI Press/The MIT Press.

[10] Jeffrey M. Bradshaw, Mark Greaves, Heather Holmback, Wayne Jansen, Tom Karygiannis, Barry Silverman, Niranjan Suri, and Alex Wong, *Agents for the masses?* In J. Hendler (Ed.) Special issue on agent technology, IEEE Intelligent Systems, March/April 1999, 53-63.

[11] JavaSpaces: http://www.sun.com/jini/specs/

[12] Mark Burstein, and Douglas Smith, *A Portable, Interactive Transportation Scheduling Tool Using a Search Engine Generated from Formal Specifications*, Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, B. Drabble (ed.), The AAAI Press, Menlo Park, CA, May, 1996 ISBN 0-929280-97-0.

[13] Thomas Emerson and Mark Burstein, *Development of a Constraint-based Airlift Scheduler by Program Synthesis from Formal Specifications*, Proceedings of the 1999 Conference on Automated Software Engineering, Orlando, FL, September, 1999.

[14] Mark Burstein, George Ferguson, and James Allen, *Integrating Agent-Based Mixed-Initiative Control with an Existing Multi-Agent Planning System*, Proceedings of the Fourth International Conference on MultiAgent Systems, Boston, MA, 2000.

[15] Craig A. Knoblock and Steven Minton, *The ariadne approach to web-based information integration*, IEEE Intelligent Systems, 13(5), September/October 1998.

[16] Ariadne: http://www.isi.edu/info-agents/ariadne/

[17] XSL Transformations: http://www.w3.org/Style/XSL/

[18] AIAI's IX Technology: http://www.aiai.ed.ac.uk/project/ix/

[19] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill and R. Jeffers, *Strong mobility and fine-grained resource control in NOMADS*, Proceedings of the 2nd International Symposium on Agents Systems and Applications and the 4th International Symposium on Mobile Agents (ASA/MA 2000), Zurich, Switzerland; Berlin: Springer-Verlag

[20] Michigan MCA: http://ai.eecs.umich.edu/people/durfee/COABS/

[21] Dartmouth College: http://actcomm.dartmouth.edu/

[22] OBJS eGents: http://www.objs.com/agility/

[23] Tim Berners-Lee, James Hendler and Ora Lassila, *The Semantic Web*, Scientific American, May 2001 http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

[24] James Hendler, *Agents and the Semantic Web*, IEEE Intelligent Systems, vol. 16, no. 2, March/April 2001, pp. 30-37. http://www.cs.umd.edu/~hendler/AgentWeb.html

[25] Robert Andrews, *Data Interoperability with DCADM and XML*, DERA report DERA/CIS/CIS3/TR000265/1.0, March 2000, Defence Evaluation & Research Agency (now QinetiQ Ltd), St. Andrews Road, Malvern, UK.

[26] RDFDB: http://web1.guha.com/rdfdb/query.html#query

[27] Squish: http://swordfish.rdfweb.org/rdfquery/

[28] RDF issue tracking: http://www.w3.org/2000/03/rdf-tracking

[29] Notation 3: http://www.w3.org/DesignIssues/Notation3.html

[30] RQL: http://139.91.183.30:9090/RDF/

[31] RDFQL: http://www.intellidimension.com/RDFQLmanual.html

[32] RDFQuery: http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html

[33] XWMF: http://nestroy.wi-inf.uni-essen.de/xwmf/

[34] RDFPath: http://zoe.mathematik.Uni-Osnabrueck.DE/QAT/

[35] RDF Contexts: http://public.research.mimesweeper.com/RDF/RDFContexts.html

[36] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng, *Semantic Web Services,* IEEE Intelligent Systems, vol. 16, no. 2, March/April 2001, pp. 46-53.

Object Interoperability for Geospatial Applications*

Paul W. Calnan† and Isabel F. Cruz
Department of Computer Science
University of Illinois
Chicago, IL 60607-7053, USA
{paulc|ifc}@cs.uic.edu

Abstract. In this paper, we analyze a geospatial application for visualizing U.S. election results in order to show the problems that need to be solved in the mapping between different XML representations and their conceptual models. We propose a framework that provides a number of core classes that allow applications to treat XML documents as graphs and to evaluate XPath expressions against such document graphs. We also propose a mechanism that allows information to be exchanged between document types. Our goal is to ultimately attain an overall framework for interoperation where maintenance problems are minimized. To achieve this, we anticipate the need for introducing metadata in the semantic layer that will guide the translation process between document types.

Keywords: Geospatial applications, user interface backend, object interoperability, technologies for interoperability.

1 Introduction

In statewide geospatial applications, hundreds of systems need to be integrated. In these applications, challenges in achieving interoperability are at the semantic level (e.g., different classification schemes) and at the data structure level (e.g., different XML DTDs). Current approaches that deal with this problem require the intervention of a human expert to perform the mapping between XML representations or to map between the conceptual models for those representations. Also, changes in the representations will typically entail specific code changes by a human expert. A recently proposed layered model [11] partitions this problem by considering the requirements of different layers: syntax, object, semantic, and application.

This paper discusses our approach to interoperability in the object layer using XML [2], developed as a part of a framework for geospatial data visualization applications. Applications using our framework should be able to seamlessly pull data from multiple XML data

*Research supported in part by the Advanced Research and Development Activity (ARDA) and the National Imagery and Mapping Agency (NIMA) under Award Number NMA201-01-1-2001, and by the National Science Foundation under CAREER Award IRI-9896052.
The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Imagery and Mapping Agency or the U.S. Government.

†Worcester Polytechnic Institute, ADVIS Research Group

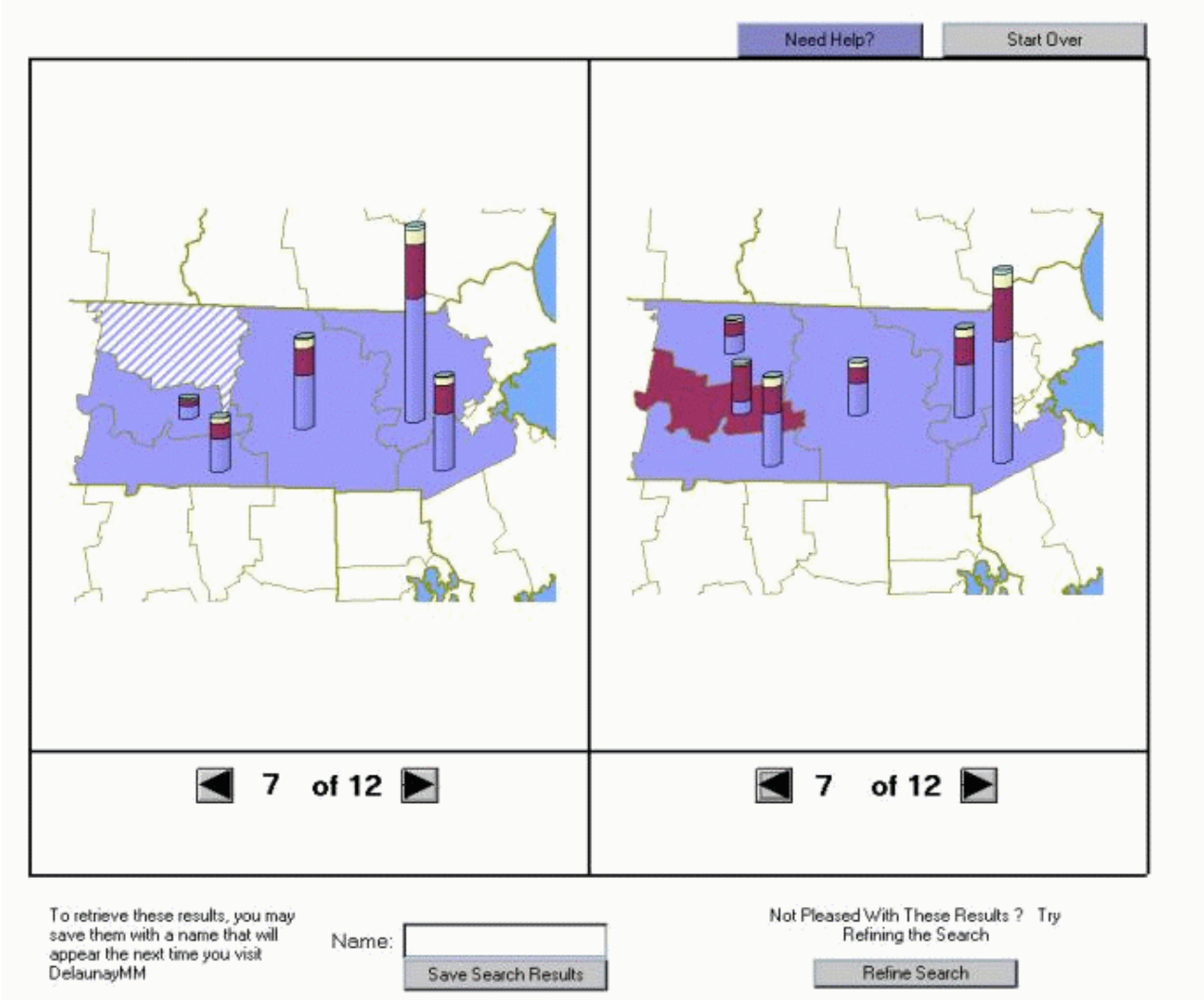


Figure 1: User interface for the election scenario

sources. Thus, we focus on the backend portions that allow for querying and integrating data from multiple XML sources.

In developing our framework, we have focused on a number of application scenarios. In this paper, we examine our election results scenario. In this scenario, we create an application that will visualize results from the 2000 U.S. Presidential Election, superimposing vote totals and campaign donations over a given region on a map. Users will be able to search for a geographic area in the United States, choose which data to display (e.g., demographic information, votes cast per candidate, campaign donations, etc.), and then view that data on a map of the region. Data can be displayed at multiple granularities, depending on what the user wants to see (e.g., the user can display a map of Massachusetts with county boundaries showing election results for each county and then zoom in on a map of Worcester County showing election results and campaign donations for each municipality). A screen shot from this scenario is shown in Figure 1.

XML was chosen as our data representation language because it is quickly becoming the new standard for information interchange. The main appeal of XML is that it allows users

to define their own document types, and then store and exchange documents conforming to that document type. A document type declaration is used to accomplish this. Document type declarations serve to identify the root element of a document, but can also contain a document type definition (DTD) [1]. This provides users a way of declaring the markup, syntax, and structure of a given document type. This also provides the parser a way of determining whether a document is valid (i.e., conforms to a DTD), rather than just well formed (i.e., uses correct XML syntax).

Allowing everyone to create their own document types makes data exchange easier, but it also makes data interoperability more difficult. Different organizations could model the same information with different document types [1]. It is trivial to load and view documents with different document types, but integrating the data between them is potentially quite difficult.

In this paper, we concentrate on providing a general-purpose application framework that allows data to be interchanged between various XML document types. We utilize XPath to concisely describe a path through the document graph and to select a number of nodes whose values can be aggregated. Finally, we introduce the concept of a *geospatial authority*. Authoritative geospatial information about the region being covered by a given application is collected in an XML document. This document serves as the source for geospatial relationships at the core of the application. For instance, in our election scenario, we maintain a geospatial authority containing the names of all of the states, the counties contained in those states, and the municipalities contained in those counties. This provides us with a means of query refinement by providing context for a given geospatial query (e.g., a search for the string "Worcester" would yield matches in New York, Vermont, Massachusetts, Wisconsin, Missouri, and Pennsylvania—the authority allows us to ask the user for clarification in which "Worcester" is desired). This also provides us with context and a starting point when looking up data in other documents, as seen in the examples given below.

Our framework is implemented in Java 1.3 and uses Apache's Xalan-Java 2 API for XML, XPath, and XSLT processing.

The paper is organized as follows: In Section 2, we examine treating an XML document as a graph, using the Document Object Model (DOM) and XPath as a means of traversing the graph. In Section 3, we present a number of classes that form the core of the backend of our application framework. In Section 4, we introduce our lookup mechanism that allows users to interchange data from one document type to another. Finally, in Section 5, we examine issues that remain open or that need to be addressed to complete our framework.

2 XML As a Graph

XML is actually a collection of W3C recommendations that define the syntax and semantics of XML and its related technologies [1]. The core XML recommendation is the XML Information Set (or Infoset) [5], which models the core abstractions of XML as a set of information items [1]. Information items represent the pieces of an XML document, what is required of them, and how they behave. The items modeled in the Infoset are reflected in the Document Object Model (DOM) [9]. The DOM allows programmers to access XML documents uniformly, regardless of the underlying implementation.

When an XML document is loaded using a DOM compliant parser, a DOM tree is returned. The DOM provides interfaces to the following information items: documents, document fragments, document types, entities, entity references, elements, attributes, processing

instructions, comments, text, CDATA sections, and notations. While this is useful, for certain purposes it would be more useful to deal with XML documents as a graph of elements.[1] Each element would have a type corresponding to its attributes. It should be possible to retrieve an element's children, parent, siblings, etc.

Also worth noting is the XPath [4] engine in the Xalan-Java API. XPath provides a simple way of expressing a path through a document tree to select a set of nodes. When a path expression is evaluated, the XPath facilities select a set of nodes relative to a context node. Any information item represented in the DOM (and consequently any information item in any document) can be selected by an XPath expression. Using XPath expressions to traverse an XML document is much more concise and easier to understand than using the DOM. Also, after analyzing a DTD, it is possible to generate XPath expressions for the body of the accessor methods necessary to perform the graph operations described above.

3 Framework Core Classes

Our framework provides a number of core classes, shown below in Figure 2, that allow applications to treat XML documents as graphs, to perform XSL transformations on an XML document, to evaluate XPath expressions against a document, and to perform inter-document lookups.

The base class that we use in the framework is the `XMLObject` class. In our Java implementation, the `XMLObject` class contains a reference to a node in the DOM tree. Through this node reference, it is possible to retrieve any data contained in the node, as well as any nodes linked to that node in the DOM tree. This class also provides an interface to the XPath facilities in the Xalan-Java API. The node referenced by an `XMLObject` object is used as the context node when evaluating XPath expressions. When an XPath expression is evaluated, a set of DOM nodes matching that expression are selected and placed in a DOM `NodeList`. The `NodeList` interface is wrapped by the `XPathResult` class, which allows users to cast the i^{th} node in the list to a `Boolean`, `Double`, `Integer`, `Node`, or `String`. Coupled with prior knowledge of attribute types (determined by a domain expert, or in the future by analyzing an XML Schema for a given document type), this allows typed access to data in the document.

The `XPathResult` class also provides aggregate operations for the set of nodes that match a given XPath expression. Once a set of nodes is selected, it is often necessary to traverse the set, accumulating data. Rather than have the application designer write code to perform these aggregate operations, we provide such code in our framework. The aggregate operations that are provided are listed in Figure 3.

There are three subclasses of the `XMLObject` class: `XMLData`, `XMLSource`, and `LookupResult`. The `XMLData` class represents elements in the DOM tree and is the basis for treating a document as a homogeneous graph. It encapsulates many of the DOM methods that relate only to elements (i.e., retrieving attribute names, retrieving attribute values by attribute name, retrieving children, ancestors, and siblings). It also translates the name and namespace

[1] Documents whose elements contain references to other elements (e.g., IDREFs) can be considered to be a graph, rather than a tree. In the DOM, references are not considered to be edges in the document tree, thus maintaining the tree structure. However, in certain circumstances, it may be necessary to treat references as edges, and thus treat the document as a graph. In any case, an XML document is primarily tree-structured, so we can use terms like sibling, parent, ancestor, and child.

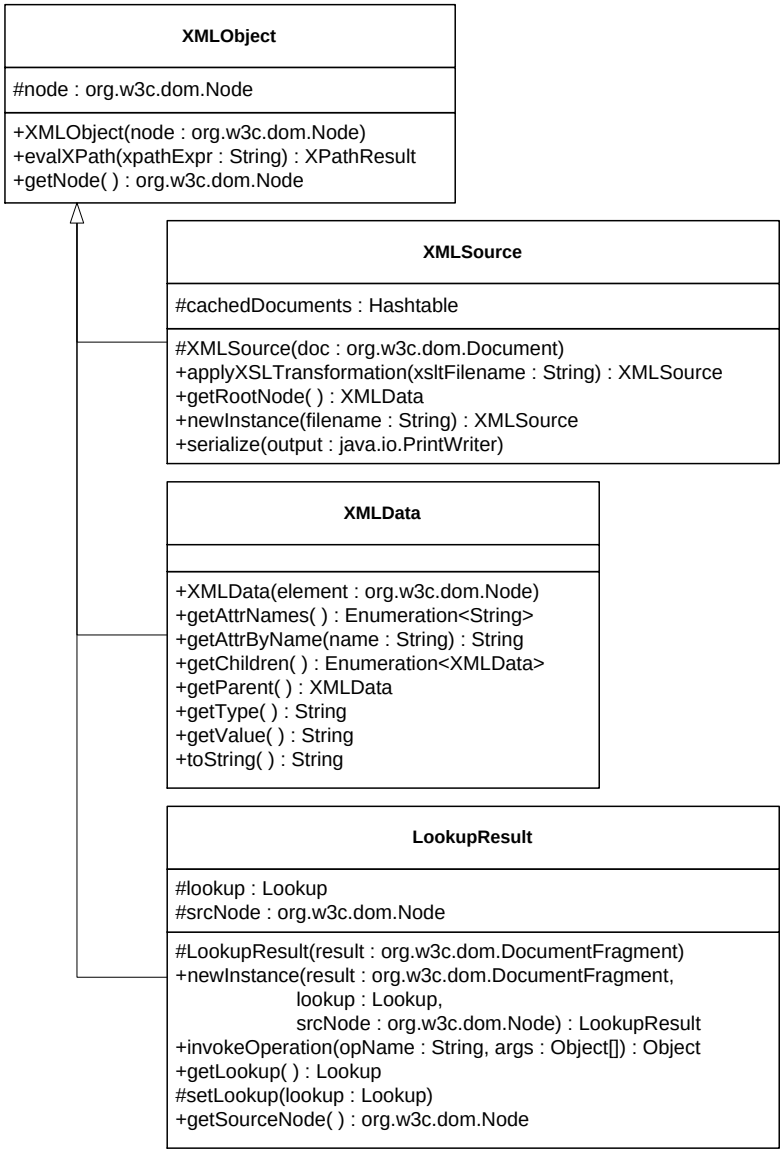


Figure 2: UML class diagram of core framework classes

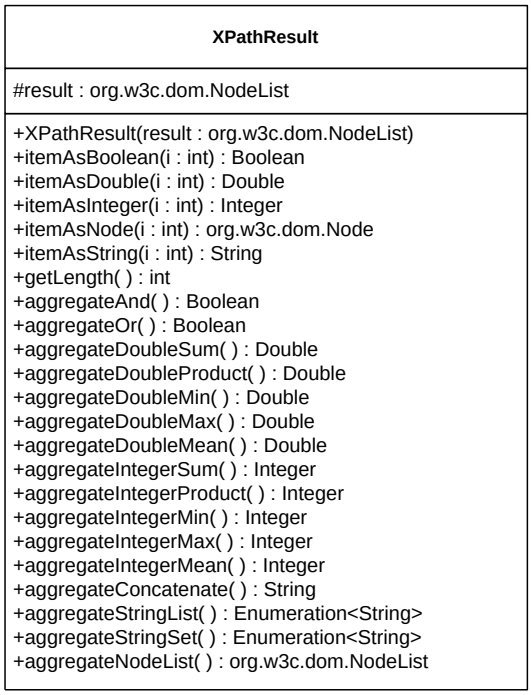


Figure 3: UML class diagram for the `XPathResult` class

of an element into the type of the graph node. `XMLData` objects are instantiated by the `XMLSource` class. `XMLSource` objects represent XML documents and allow users to load and parse documents, to serialize documents out to a stream, and to retrieve the root node of a document as an `XMLData` object. The `XMLSource` class also provides access to the XSLT [3] facilities (discussed below) provided in the Xalan-Java API. The `LookupResult` class encapsulates the resulting data from an inter-document lookup (discussed below).

3.1 Example using the Framework Core Classes

In our election scenario, we start with an XML document that we call the *geospatial authority*, which contains geographic information about the area being covered by the application (in this case, the United States). For simplicity, we consider an authority document that contains state elements, which in turn contain county elements, which in turn contain municipality elements. The DTD for the geospatial authority is shown below:

```
<!ELEMENT state        (county+)>
<!ELEMENT county       (municipality+)>
<!ELEMENT municipality EMPTY>

<!ATTLIST state        name CDATA #REQUIRED>
<!ATTLIST county       name CDATA #REQUIRED>
<!ATTLIST municipality name CDATA #REQUIRED>
```

This would suggest that subclasses of `XMLData` are needed for states, counties, and municipalities. Each of these classes would have an accessor method that would allow the user

to retrieve the name, as well as the region(s) contained by that region (e.g., retrieve all counties for a given state, or retrieve the municipality named "Worcester" from Worcester county in Massachusetts). There are also documents for each state containing election results. Potentially, each state can have a different DTD, and consequently, a different structure for its results document. For simplicity, we will examine the DTDs for election results from only two states, Massachusetts and Maine:

MA_results.dtd

```
<!ELEMENT state        (county+)>
<!ELEMENT county       (municipality+)>
<!ELEMENT municipality (candidate+)>
<!ELEMENT candidate    EMPTY>

<!ATTLIST state        name CDATA #REQUIRED>
<!ATTLIST county       name CDATA #REQUIRED>
<!ATTLIST municipality name CDATA #REQUIRED>
<!ATTLIST candidate    name CDATA #REQUIRED
                       votes CDATA #REQUIRED>
```

ME_results.dtd

```
<!ELEMENT state        (county+)>
<!ELEMENT county       (municipality+)>
<!ELEMENT municipality (ward+)>
<!ELEMENT ward         (precinct+)>
<!ELEMENT precinct     (candidate+)>
<!ELEMENT candidate    EMPTY>

<!ATTLIST state        name  CDATA #REQUIRED>
<!ATTLIST county       name  CDATA #REQUIRED>
<!ATTLIST municipality name  CDATA #REQUIRED>
<!ATTLIST ward         id    CDATA #REQUIRED>
<!ATTLIST precinct     id    CDATA #REQUIRED>
<!ATTLIST candidate    name  CDATA #REQUIRED
                       votes CDATA #REQUIRED>
```

This represents the general structure of the election result data as presented at the Massachusetts and Maine state websites. In order to better understand these DTDs, Figure 4 shows the E-R diagrams of these document types.

Note that in the diagram, the cardinality of all relationships is one-to-many. This is due to the fact that, in the DTD, all subelements have the "+" qualifier, meaning that one or more instances of that subelement can appear. While this is fine for the state-to-county relationship and the county-to-municipality relationship, it does not tell the whole story for candidates. The municipality- or precinct-to-candidate relationship should have a cardinality of many-to-many, since there are many municipalities or precincts and many candidates. However, the DTD does not reflect this—it only states that each municipality or precinct can have more than one candidate. To determine a many-to-many relationship, it would be necessary to examine the actual data.

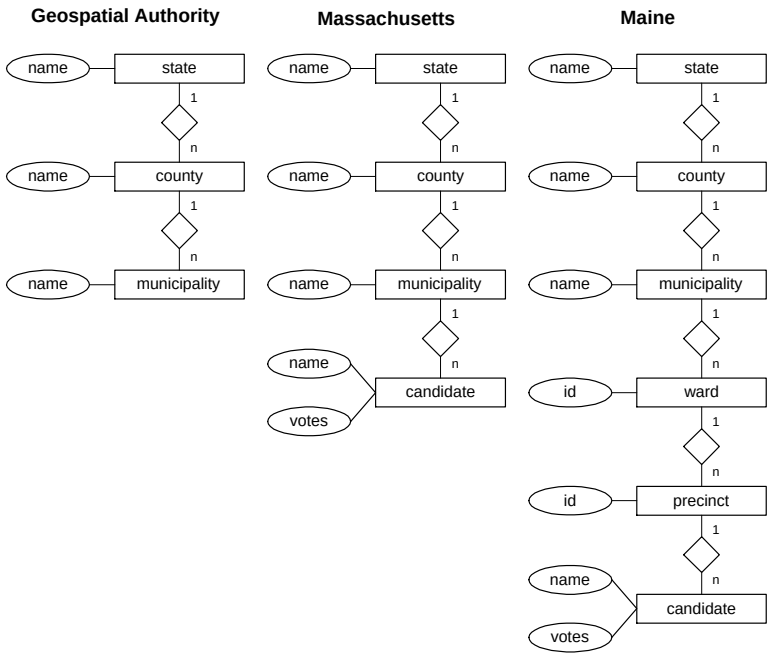


Figure 4: DTD E-R Diagrams

It was stated above when discussing the geospatial authority document that classes for states, counties, and municipalities would be desirable. However, since each state's results document could potentially have a different structure, it would be unfeasible to define a separate state, county, municipality, and candidate class for each state's election results. It would make more sense to have just one state, county, and municipality class for the entire application and let the framework classes handle the necessary translation between the underlying XML formats of the various election result documents. To accomplish this, we introduce our lookup mechanism.

4 Lookups

Interoperability hinges upon a system that is able to seamlessly interchange data between various document types. Data from one document needs to be selected, possibly restructured, and then linked to data in another document. Our framework provides a way of linking data in this manner using what we call *lookups*. This section discusses the features of our lookup mechanism and some of the design rationale behind those features. We will illustrate this using a running example from our election scenario: defining a lookup for Massachusetts election results for municipalities.

Lookups in our system are specified declaratively in a lookup specification document. This document, stored in XML, allows application designers to specify all lookup types for an application. Each lookup is given a name, a description, and an identifier, as shown below:

```
<?xml version='1.0' standalone='no'?>
<!DOCTYPE lookup-spec SYSTEM 'lookup-spec.dtd'>

<lookup-spec>
```

```
<lookup name='MA Municipal Results'
        description='Massachusetts election results per
                     municipality'
        id='1'>
```

4.1 Predicates

Any node in an XML document used by our system can potentially have any number of lookups associated with it. Therefore, we first need a way in which we can define predicates that determine whether a lookup exists from a given source node. These predicates are expressed as XPath expressions. When checking if a node is associated with a lookup, the predicate XPath expressions are evaluated using this source node as the context node. If all of the predicates evaluate to non-null results, we can consider this node associated with this lookup and can proceed to execute the lookup. In our example, we only want to link municipalities in the geospatial authority to municipalities in the Massachusetts election results document. Consequently, we define the following predicates:

```
<predicate>
<!-- ensure that the source node is a municipality -->
self::municipality
</predicate>

<predicate>
<!-- ensure that the source node is in Massachusetts -->
ancestor::state[attribute::name='Massachusetts']
</predicate>
```

4.2 Arguments

Lookups actually link *entities* (that is, entities in the database sense; these entities are expressed in XML as elements) in different documents, not actual instances of the entities. In other words, lookups act as a link between different data types, not their objects. It may only be possible to state a lookup in terms of variables whose values depend on the context of the lookup being performed (i.e., the source node's context). For instance, in our running example linking a municipality in Massachusetts with a municipality in the election results, one lookup can serve all municipalities in the state. Contextual data, such as the county and municipality name, is necessary for the lookup to be performed, but cannot be determined until the lookup is about to be executed. Therefore, we provide a mechanism for defining arguments whose values are computed before executing the lookup and then substituted into the body of the lookup. These arguments are associated with an XPath expression. When the lookup is about to be executed from a given source node, the XPath expression is evaluated using the source node as the context node, and the resulting value is substituted for the argument name in the body of the lookup. For example, we would need to define arguments for county and municipality names in order to properly link a municipality to the election results:

```
<argument>
  <name>$county_name</name>
```

```
  <value>parent::county/attribute::name</value>
</argument>

<argument>
  <name>$municipality_name</name>
  <value>attribute::name</value>
</argument>
```

4.3 Lookups: XPath or XSLT

The next step is to define the actual body of the lookup. There are two types of lookups supported by our framework. The first type of lookup involves simply selecting a set of nodes in a document using an XPath expression. To specify an XPath lookup, the actual XPath expression can be stated directly in the lookup specification document. The second type of lookup involves restructuring a subset of the data in a document using XSLT. To specify an XSLT lookup, the name of the XSLT file can be stated in the lookup specification document. When the time comes to execute a given lookup, the XPath expression or XSLT file is loaded, argument values are evaluated and substituted, and the lookup is performed. It is also necessary to state the file name of the target document. In our example, we want to find the municipality element in the target document and to select all children of that municipality. The XPath lookup to accomplish this, written in terms of the variables given above, is shown here:

```
<target-document>
MA_election_results.xml
</target-document>

<!--
  Start at the root, trace through the tree to find the
  municipality, and select all children of the municipality.
 -->
<xpath-expr>
/child::state[attribute::name='Massachusetts'] \
  /child::county[attribute::name='$county_name']  \
  /child::municipality[attribute::name='$municipality_name'] \
  /child::*
</xpath-expr>
```

4.4 Linking the Results

When an XPath expression is evaluated, a `NodeList` or a `NodeIterator` containing the selected nodes is returned. When an XSLT file is evaluated, the resulting DOM tree structure is returned. While DOM trees that result from XSL transformations can have XPath expressions evaluated against them, `NodeList`s and `NodeIterator`s that result from XPath expressions cannot. It would be useful to organize these results in such a manner that XPath expressions can be evaluated against them. Similarly, we would like to be able to cache and possibly serialize these results. It would be nice if we could simply add these nodes as children of the lookup's source node, but this is not allowed by the DOM because it could

potentially make the containing document invalid. However, the DOM provides the `DocumentFragment` interface for representing lightweight collections of nodes. The `DocumentFragment` interface is intended to support movement of nodes for operations such as "cut" and "paste" [9]. Also, since `DocumentFragment`s are nodes, XPath expressions can be evaluated against them.

We can specify that the XSLT part of the Xalan-Java API produce a `DocumentFragment` with the results of the transformation. For XPath lookups, we can simply take each node in the resulting `NodeList` or `NodeIterator` and add that node as a child of a `DocumentFragment`.

Our framework provides the `LookupResult` class, a subclass of `XMLObject`, to store the resulting `DocumentFragment`. The details of this class are discussed later on, but it is worth noting here that in the lookup specification, the application designer can specify which result class to instantiate with the lookup results. Either `LookupResult`, or a subclass of it, can be specified to take the lookup results. In our running example, we simply use the `LookupResult` class:

```
<result-class>LookupResult</result-class>
```

4.5 Operations: Accessing the Results

Once a lookup is performed, code is needed to access the data contained in the nodes that result from the lookup. Potentially, each lookup can select nodes from different documents containing different structures. Therefore, each lookup would require its own class to be instantiated with the appropriate code to access its data. However, this can quickly become cumbersome as the number of lookups increase. Also, if the structure of the various documents change, those classes would need to be rewritten. In our election scenario, each state could potentially require a separate class for lookups for its election results. While these classes could be arranged into an inheritance hierarchy, having 50 election result classes would be a little excessive. It would be more useful if accessor methods could be defined at the lookup level.

Accessor methods can be defined using XPath expressions that will be evaluated using the `DocumentFragment` as the context node. The return values for such accessor methods can be computed using our aggregate operations discussed above. The application designer can therefore define accessor operations in the lookup specification using a name, aggregation type, and an XPath expression for the body of the operation. Parameters can also be defined when extra context is needed for the operation. Default values for those parameters can also be given, where necessary.

In our running example, we would define the following two operations:

```
<!-- select all candidate names, return a set of strings -->
<operation name='getCandidateNames' aggregation='sset'>
  <body>
    child::candidate/attribute::name
  </body>
</operation>

<!-- select vote total for a candidate, return an integer -->
```

```
<operation name='getVotesByCandidate' aggregation='isum'>
  <parameter>
    <name>$candidate_name</name>
  </parameter>

  <body>
    child::candidate[attribute::name='$candidate_name'] \
    /attribute::votes
  </body>
</operation>
</lookup>
</lookup-spec>
```

If all election result lookups define operations with the same name, multiple classes to represent the election results would no longer be needed. Rather, we can use one subclass of `LookupResult`. Operations defined in the lookup specification can be invoked using the `invokeOperation()` method provided by the `LookupResult` class. In our subclass, we can provide methods (e.g., `getCandidateNames()` and `getVotesByCandidate()`), which simply call `invokeOperation()` with the appropriate parameters.

4.6 The Lookup Specification

As stated above, all lookups are specified in an XML document. These documents must conform to the lookup specification DTD (`lookup-spec.dtd`), shown below:

```
<!ELEMENT lookup-spec     (lookup+)>
<!ELEMENT lookup          (predicate+,
                           argument*,
                           target-document,
                           (xpath-expr | xslt-file),
                           result-class,
                           operation*)>
<!ATTLIST lookup name        CDATA #REQUIRED
                 description CDATA #REQUIRED
                 id          ID    #REQUIRED>
<!ELEMENT predicate       (#PCDATA)>
<!ELEMENT argument        (name,
                           value)>
<!ELEMENT name            (#PCDATA)>
<!ELEMENT value           (#PCDATA)>

<!ELEMENT target-document (#PCDATA)>
<!ELEMENT xpath-expr      (#PCDATA)>
<!ELEMENT xslt-file       (#PCDATA)>
<!ELEMENT result-class    (#PCDATA)>

<!ELEMENT operation       (parameter*,
```

```
                              body)>
<!ATTLIST operation name         CDATA #REQUIRED
                    aggregation (nodelist |
                         and | or |
                         dsum | dprod | dmin | dmax | dmean |
                         isum | iprod | imin | imax | imean |
                         concat | slist | sset) #REQUIRED>
<!ELEMENT parameter       (name, default-value?)>
<!ELEMENT default-value   (#PCDATA)>
<!ELEMENT body            (#PCDATA)>
```

5 Conclusion and Open Issues

In this paper, we analyzed a geospatial application for the U.S. elections as a means of illustrating the problems that need to be solved in the mapping between different XML representations and their conceptual models. The framework that we presented allows application designers to treat XML documents as homogeneous graphs and to evaluate XPath expressions and XSL transformations against a document. Our framework also allows application designers to define a set of lookups that integrate data from multiple documents. Data retrieved using a lookup can also have operations defined for it declaratively in a lookup specification. Coupled with our aggregation operations, this provides uniform access to non-uniformly structured data.

We anticipate the need for introducing metadata in the semantic layer to guide the translation process between document types. In this section, we discuss metadata issues in the semantic layer and a number of other issues that still need to be investigated.

5.1 Metadata, Semantics, and Ontologies

A certain amount of metadata is necessary for a domain expert or application designer to appropriately identify the entities that can be linked and how the lookups should be performed. In some cases, this metadata is available from the DTD or from another external source that describes that document type. In other cases, this metadata may be implied. For example, one can reasonably expect measurements in documents from Europe to be in metric units, while measurements in documents from the United States to be in feet and inches, even though this information is not explicitly stated anywhere in the document. Some conversions may be handled by XSLT, but some may require more complicated computations and consequently must be performed by another mechanism.

There are also naming issues that need to be taken into consideration. For instance, it is possible for a geographic entity to have multiple accepted spellings (e.g., Foxboro, MA vs. Foxborough, MA; or Mt. Washington vs. Mount Washington). Often times, foreign place names have multiple accepted English spellings, and this too must be taken into consideration. In some cases, this can be handled using a crosswalk table or some other form of dictionary data structure. In other cases, the accepted spellings of a place name may depend on the current context or the location of that place.

Further research is also needed to see how to use ontologies, especially as presented in [7] and [8], in our system.

5.2 Technical Issues

We need an engine capable of executing graph queries. With such an engine, it may be possible to take a generic graph query language that supports aggregation, such as the G+ language presented in [6], and compile queries in that language into XPath expressions or another XML based query language (e.g., XQL, or whichever XML query language reaches W3C recommendation status).

We can also look at performance issues, especially in a distributed environment. Each XML document that we are interoperating between can potentially be stored on a different server. We need to look at different ways of caching and processing data. We also are examining different security schemes for accessing the various XML data sources.

At this point in the development of our framework, much of the code necessary to perform the tasks described above must be written by hand. This may become too complex to be practical. In the future, we plan to analyze all DTDs that will be supported by and application and generate the necessary code to perform many of the tasks described above. It would still be necessary for the application designer or a domain expert to determine the links between the different document types. It may be possible, however, to automatically or semi-automatically generate the lookup specifications necessary to execute the lookups, but more research is needed in this area. In order to generate any code based on a DTD, it is first necessary to parse the DTD to determine its structure. This can be done by creating a DTD graph, using algorithms described in [10] and [12]. From there, we can begin to examine code generation for XPath-based accessor methods in the lookup specification. It may also be possible to automatically or semi-automatically generate the lookup code (using either XPath or XSLT). Code generation can be supplemented by the input of the application designer or domain expert via a visual tool that displays E-R diagrams based on a given DTD.

5.3 Document Structure

The election scenario example shown throughout this paper is somewhat contrived (the actual data was originally retrieved in HTML format from a website and then converted to XML) as we have created all of the documents and defined their structure. Obviously, in a real-world application, the document structures could potentially be quite different, requiring more restructuring of the data.

We also need to examine the effects that changing a document type would have on code that has already been written or generated. Ideally, our final framework would be sufficiently powerful to handle such changes.

6 Acknowledgements

We would like to thank Nancy Wiegand and Steve Ventura for discussions on the subject of statewide geospatial applications.

References

[1] D. Box, A. Skonnard, and J. Lam. *Essential XML Beyond Markup*. Addison-Wesley, 2000.

[2] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 2000.

[3] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 1999.

[4] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, 1999.

[5] J. Cowan and R. Tobin. XML Information Set. W3C Candidate Recommendation, 2001.

[6] I. Cruz and T. Norvell. Aggregative Closure: An Extension of Transitive Closure. In *IEEE International Conference on Data Engineering*, pages 384–390, 1989.

[7] M. Erdmann and R. Studer. How to Structure and Access XML Documents with Ontologies, 2001. DKE 36(3): 317–335.

[8] F. Fonseca. GIS_ontology.com. GIScience 2000. http://www.giscience.org/GIScience2000/papers/218-Fonseca.pdf.

[9] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 2 Core Specification Version 1.0. W3C Recommendation, 2001.

[10] D. Lee and W. Chu. Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. In *Proc. 19th Int'l Conf. on Conceptual Modeling*. Springer-Verlag, October 2000.

[11] S. Melnik and S. Decker. A Layered Approach to Information Modeling and Interoperability on the Web. In *ECDL 2000 Workshop on the Semantic Web*, Lisbon, Portugal, September 2000.

[12] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the* 25^{th} *VLDB Conference*, Edinburgh, Scotland, 1999.

Brokerage of Intellectual Property Rights in the Semantic Web

Roberto García, Jaime Delgado
Distributed Multimedia Applications Group (DMAG)
Technology Department, Universitat Pompeu Fabra (UPF)
Pg. Circumval·lació 8, E-08003 Barcelona, Spain
{roberto.garcia, jaime.delgado}@tecn.upf.es

Abstract. New approaches in the Web environment are underway. These new methodologies try to leverage it from an information medium to a knowledgeable level, from a machine point of view. This upgrade, mainly focused on improving Web automation capabilities, can solve some of the problems derived from its widespread adoption. Among them, the necessity of a framework to manage the enormous market of digitalised multimedia and to ensure that all the intervening actors get a satisfactory experience from the Internet adventure. An application under development is described and future plans in this direction are presented. A broker component has already been implemented applying a Semantic Web layered architecture. Its mission is to mediate in a restricted community of digital video providers and distributors, benefiting also final purchasers. The intention is to use it as a test bed for this promising initiative and its application in the Intellectual Property Rights (IPR) domain.

1 Introduction

The Internet, and more concretely Web technologies, has matured, passed the promises phase and is currently firmly established in our society. Now it takes part of our daily life, it is in the appropriation phase [1]. We are trying to profit, economically or not, from it, as we did with other revolutionary ideas that arose before. Now, we can observe with greater perspective and calm what has been achieved and what can be done further. It seems like new phases must be engaged, solving the new problems that came up and, why not, making new promises.

One of the biggest problems, strongly founded on the consequences of the overall Digital Era, is the easy copy and ulterior uncontrolled distribution of multimedia creations. This motivates new approaches to intellectual property management, mechanisms by which the authors of these materials and all other implicated actors get fair revenues from their efforts. However, the management of such features in the current Web environment, decentralised and enormously dynamic, becomes almost impossible if these mechanisms are not largely automated. Therefore, the real problem is the lack of an easy automation framework in the current Web, as it also happens to other initiatives in the field, for instance thus centred on resource location.

This fundamental problem is the focus of the new Web environment initiative, the Semantic Web. The migration of the interoperability layer from the syntactic to the semantic

level is the base of this initiative. This new approach, despite less suited a priori to computer abilities, provides a more capable base for the automation of complex processes in a highly heterogeneous environment like the Web.

The Semantic Web is the core of all the work presented in this paper. It has been applied to its main objective, the use of semantics in the Web for IPR conceptual modelling. In addition the implemented business model and architecture will be shown, since there is already a developed part and in order to provide a practical context.

Work already done has been co-financed by the Catalonian Government initiative to establish a pilot infrastructure of the future Internet 2. Concretely, inside the multimedia cluster that explores the multimedia capabilities of this evolution of the current Internet, especially for video distribution. The developed application, called MARS (Multimedia Advanced brokerage and Redistribution Surveillance), deals with video IPR management for a group of video producers that also participate in the project. However, it is important to remark that despite this initial focus, it can easily upgrade to any type of intellectual property resources and to wider environments, mainly thanks to the semantic approximation choosen.

2 Approach

In this section, we depict the different facets that explain the way the described project has been considered. They have marked the development until now but have even greater influence in the future work.

2.1 Intellectual Property Rights and the Semantic Web

Since now the Web is more and more business oriented, organisations in all sectors are trying to automate its processes and relations to improve services, reduce costs and attain global markets. However, these efforts are finding great difficulties, especially when considering its implementation in the wide and open environment that the Web provides. Moreover, the multimedia creations sector is not one of the easiest to deal with.

There are many problematic issues. The products in this market are not clearly defined ones. They can have multiple independent components that involve multiple actors with different rights over this material. Consider, for instance, a sophisticated Web advertisement, with a soundtrack, some good quality photos, a synthetic animation of the product to clearly show its functionality, etc.… Related to this, there is the identification problem of all these creations and the actors involved.

Finally, there are the interoperability problems that an automated approximation to this problem will find. There exist many different vocabularies to deal with intellectual property, derived from diverse cultures, legislations, communities… This is the key issue to extract full potential from an automated platform for IPR management can provide. Understanding between parties using different vocabularies is fundamental. Therefore, there must be a supporting layer enabling such mappings, since no communication is possible if there is not a common base.

All these requirements fit pretty well in the features Semantic Web is promising. Indeed, we can now observe some initiatives in this direction, as it could be observed in the last W3C Workshop on Digital Rights Management [2]. Therefore, our intention in the underway

MARS project [3] is to develop an IPR management system that profits from Semantic Web features. It should appear completely integrated in the Web, facilitating interoperability and allowing advanced processes automation by extracting full potential from the provided semantic layer.

As an overview, detailed further, our plans involve the following Semantic Web building blocks:

- URIs as identifiers. This includes URNs when persistence is needed. They are used to identify creations and to reference digital certificates. The latter, in conjunction with digital signatures, will allow actor identification and validation of the statements they made.
- Ontologies. They define the different vocabularies used during statements construction. They are not limited to the intellectual property domain, some model more abstract levels or describe concrete multimedia types, for instance videos. Ontologies are interconnected, directly or by common upper levels, so interoperability through mappings between them is feasible.
- Metadata carrying semantic annotations. It is the framework that merges the previous pieces. Metadata fragments will talk about a resource through an URI and use ontology words and the semantics they define.

2.2 Semantic ExtraWeb

Although Semantic Web initiative has great potential, it is still at its beginnings. For the moment, as it happened to the initial Web, it is being applied to very limited and more or less closed environments, sometimes called Community Web Portals. We can also use the more general term Semantic Extrawebs, since they apply to the same scope than the well-known extranets. However, Semantic Web has openness in its foundations, inherited from its Web origins, and this establishes a great difference with other similar initiatives. In the future, it may upgrade easily to the global domain aggregating independently developed initiatives.

Therefore, for the moment, the developed system is intended to cope with the necessities of a small community of users in the multimedia environment. This is not a handicap for future wider extensions or interactions with other previously endemic communities. As has been said, interoperability is one of the main features of the Semantic Web, as can be read for instance in [4].

2.3 Business model and brokerage

Figure 1 shows the general IPR model that has been considered. It is inspired by the one defined by the Imprimatur project [5].

However, we have simplified this model considering the necessities of the community participating in our project. This allows us to facilitate implementation without losing any characteristic because the application environment does not have the entire requirements covered by the complete model. First, the number of actors has been reduced because the main participants in our project cope with the three top roles of the value chain, shown in the centre of *figure 1*.

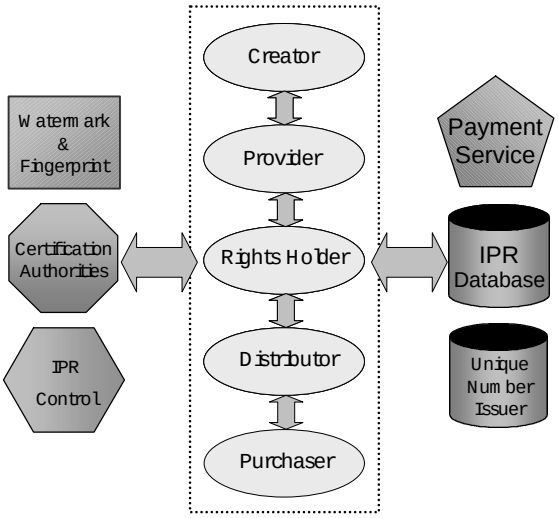


Figure 1: general IPR model

A further modification is the introduction of a new entity that will coordinate all the services provided to the value chain. Brokerage is a main issue in e-commerce environments [6][7]. Therefore, a broker will be the main interacting party for the other actors, hiding them the complexities of the whole model. At the same time, it presents a unified view to all the available services. The broker can also coordinate actions in an efficient and coherent form because it has a central and complete view of all that occurs.

The final considered business model is presented in *figure 2*.

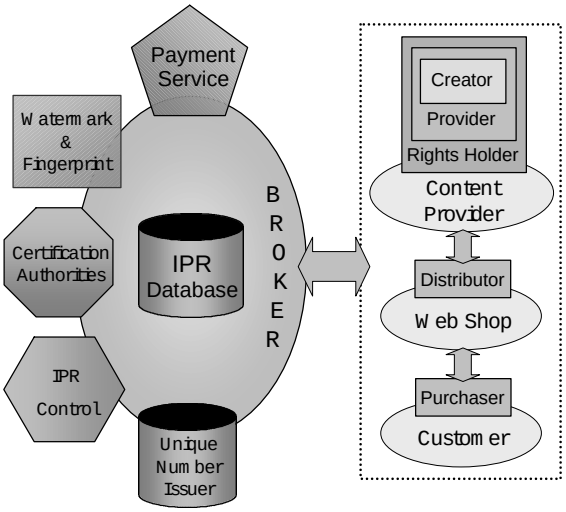


Figure 2: Considered business model

Two of the provided services must be highlighted because they are tightly connected to the undertaken approximation. They are the *Watermark and Fingerprint* facility and the *IPR Control* service. They appear because IPR control, and not IPR enforcement, is the method we have implemented to ensure that all the actors get the expected revenues. There have been many attempts to implement methods that restrict uncontrolled multimedia contents

distribution. However, until now, they have not had the expected results. They have not succeeded because they impose proprietary software, or even hardware, solutions and contradict the current openness and multi-device tendencies.

Therefore, our approach comes from a different point of view. Digital multimedia materials treated by the application do not carry any mechanism to avoid its copy and redistribution. Instead, they are watermarked, they contain an invisible digital mark that contains an identification number. It is inserted with a password that must be provided later to retrieve it.

On the other hand, modular chunks of metadata are also associated to multimedia resources. They explain, in a machine understandable way, the rights situation of these multimedia streams of bytes and other characteristics. No need to say that such streams, with so complex senses for us, are hardly understandable for computers. Therefore, metadata must have the necessary expressive power to deal with the complexities of the intellectual property and multimedia field.

These metadata annotations, in conjunction with the possibilities of digital signatures and certificates, can be used as proofs on transmission and licensing of rights on intellectual property, i.e. digital contracts. For instance, we can get a system mainly based on licensing, where this semantically modelled licenses and other statements allow a great level of automation of their processing [8].

Among this processes there is *IPR Control*. Its commitment is the continuous monitoring of the Web, focusing in those services providing multimedia material online. When a suspicious digital creation is found it tries to identify it. There are three kinds of identification:

- If the creation is not watermarked, or the mark is not accessible, identification could be done by contextual information or some unique characteristics.
- If a watermark is retrieved, automatic identification can be very reliable. There is also the possibility that more than one watermark is found, identifying different component creations.
- There is also the possibility that the creation has a fingerprint. This special watermark, not only identifies a creation, but also determines a concrete transaction.

Results could not be accurate enough in the first case. For instance, from the video title appearing in the controlled Web page a controlled video copy corresponding to it can be located. Digital videos can be compared, e.g. total length or some key frames. Finally, if it is not possible to determine identity, pre-established with a similarity threshold, identification may require human intervention. Therefore, the watermarks and fingerprints present in the other two cases are very useful.

Finally, after identification, the relevant IPR statements are retrieved. Thus, we can check if the intellectual property is available in a situation under its rights statements. In case the identification has been done thanks to a fingerprint, we can even track to the last controlled transaction on this creation.

If any problem is encountered then we enter the legal scenario. Now, we have some clues pointing to a party that may have bypassed the rights it has acquired and, at least, one actor using a creation under unlawful conditions, from the IPR statements perspective. Here we must return from the world of bits back to the world of atoms. Affected parties must be

notified and then laws have the floor. At this point, our system has nothing to do and we must rely on laws application. However, some help can be provided, digital signatures have legal support in some countries and signed contracts and notarisation of the IPR database can be considered in the legal scenario. We hope, in the future, laws and technology may evolve more aware from each other and mutual support would be easier.

To conclude, we must say that, although the business model points to IPR control, the application per se is not limited technologically to this methodology. The components focusing on this control solution are only Watermarking and IPR Control. However, semantic IPR statements can be used also for rights enforcement.

3 Architecture

This section will describe the architecture of the application we are developing. It is composed of the broker and some related tools, shown in *figure 2*. The provided view is structured in layers, see *figure 3*, starting at the final user side with the presentation layer and then going deeper in. Application logic layer provides the high level services available for the intervening actors of the multimedia value chain. Knowledge layer is the application core setting the vertebral column for the whole initiative, which is made persistent using the storage layer at the bottom.

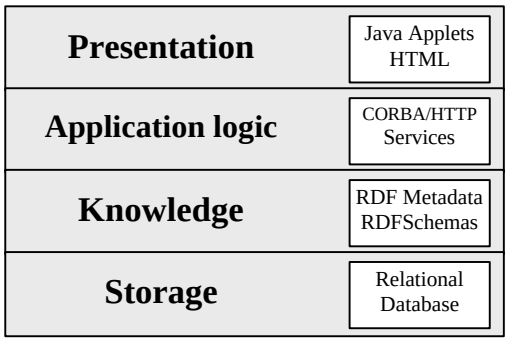


Figure 3: Layered application architecture

However, we will explain the different layers in an order more coherent with the followed development process. We started conceptualising the domain, i.e. establishing the knowledge layer. Then, over the previous structure, the services required by the actors were defined inside the application logic. Finally, the persistence and the user interface were completed.

3.1 Knowledge layer

This layer has guided the application development from the beginning. Its mission is to create a well-established conceptualisation of the domain, in our particular case a digitalised multimedia market. More specifically, it is an environment for video commercialisation structured following the previous business model, centred in supporting IPR management of these digital assets.

The fundamental objective is producing a machine interpretable model and thus to make the conceptual level the automation support point. We have followed Semantic Web guidance

and, in this initial phase of the project, only the more basic tools it provides have been used. More sophisticated ones were not available when the project started or just announced.

Therefore, we use RDF [9] and RDFSchema [10] as the building blocks of the knowledge layer. RDF uses URIs as resource identifiers and becames the language to express the metadata that describes video semantics. Nevertheless, the real semantics carrier resides in the structure of the vocabularies used by the metadata language. These vocabularies are implemented using RDFSchema.

These key issues of the knowledge level (identification, vocabulary and metadata) are detailed in the next sections.

3.1.1 Identification

URLs are the way things are named in the Web but URLs have a serious problem, its persistence. We can get one URL to some Web resource but it may be unavailable in the future or point to a completely different thing.

To avoid this problem and enlarge the set of things that RDF can refer to it uses URIs [11]. These are very similar to URL, indeed URIs include URLs, but they are an identification system, not only locators. Formally, any identifiable object, real or virtual, can have an URI and, what is more important, URIs also include URNs. These are especial ones because they have an institutional commitment to persistence. This can be reduced to a kind of mapping service from the URN that continuously identifies it to the location where it can be actually found, which is an URL. It can also provide more information but, at least, we have the institution compromise this identifier is stable.

Therefore, URIs give us a very flexible identification system. Commonly we will use directly URLs in the IPR statements we construct with RDF. This is not a problem if we have a clear idea about what the URL is pointing to. Nevertheless, in some cases it would be preferable a more Web-independent and deeply founded identifier.

For instance, if we are talking about a book it is preferable to use the ISBN that actually is identifying it, in URN form *urn:isbn:84-85081-95-1*.

In our case, we will use URN when talking about creations in their abstract form, this is a video as an intellectual creation. The mapping schemes, or institutions that make the persistence commitment, can be specialised ones like in the books case or other oriented to the digital environment, like the digital object identifiers provided by DOI [12].

For the moment, because the system is pretty closed and for testing purposes, the broker is acting as an identifiers issuer in the form of URNs, like *urn:mars:687455*. Nevertheless, the broker can be completely transparent to this matter. Creations can be previously identified and this identifier provided to the broker, although always in URI form. In the other hand, when talking about a concrete copy of a video available in a defined location, an URL will be used in the common way. For example *http://video.provider.org/687455.mpg*.

Finally, all the intervening actors also need identification. The intention stated in the approach was to identify them by using digital certificates. Profiting from this public key infrastructure digital signatures would also be used to track responsibility in the transactions actors realise. However, this module has not been adapted yet. Now, we use the traditional user-identifier and password method, so this issue is moved to the future work section. Currently, the broker generates these user identifiers, like the previous video creations identifiers in URN form.

3.1.2 Vocabularies

Ontologies, formalised vocabularies, are the building blocks of the Semantic Web. Indeed, the Semantic Web tools provide only the grounding over which different vocabularies can be developed and interconnected, facilitating its reuse and refinement.

In the MARS project, some of these vocabularies have been used to construct the conceptual model of the application domain. Some are reused and some are RDFSchema implementations of previously defined conceptualisations. Finally, the uncovered aspects have been attained developing the application specific ones.

Any RDFSchema can be reused. Actually, the application is reusing the Dublin Core [13] RDFSchema. This provides a very generic set of properties for basic creations descriptions. However, other possibilities exist, the most remarkable could be future schemas for MPEG7 [14] that would allow very detailed video descriptions.

The basic schema, that provides the groundings of the application domain model, is a RDFSchema implementation of a predefined model. Work started from the results of the INDECS [15] project, mainly a structured dictionary of terms for metadata related to the e-commerce of intellectual property. Its focus is providing an interoperability framework for this sector so it was a good starting point for the implementation of a schema for IPR management. First, a deep study of the INDECS documentation resulted on a clearer hierarchical view. The upper levels and the connection we made to RDF for its posterior RDFSchema implementation are shown in *figure 4.*

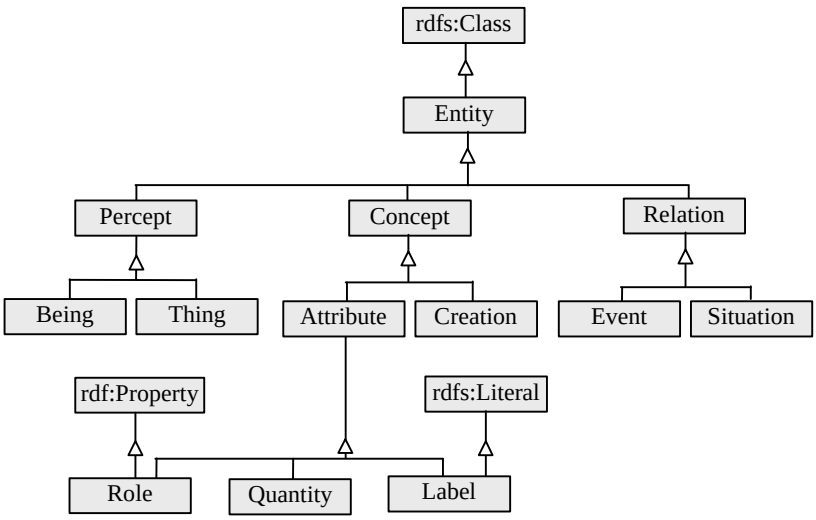


Figure 4: MARS INDECS-based upper ontology levels

After that, a minimal implementation of the whole model was made resulting in a RDF schema. We focused on the indispensable parts for our concrete application. The basic upper levels and those necessary for IPR modelling were the main interesting parts.

On top of that, the more concrete part dealing with videos, and other features specific to our approach, were constructed. This new schema is not an isolated part, it is grounded on the INDECS based schema by refining the appropriate concepts it defines. For instance, we added some concrete intellectual property rights so real IPR statements can be done. Some of

these extensions are presented in *figure 5*, where original INDECS concepts are preceded by the *indecs* alias.

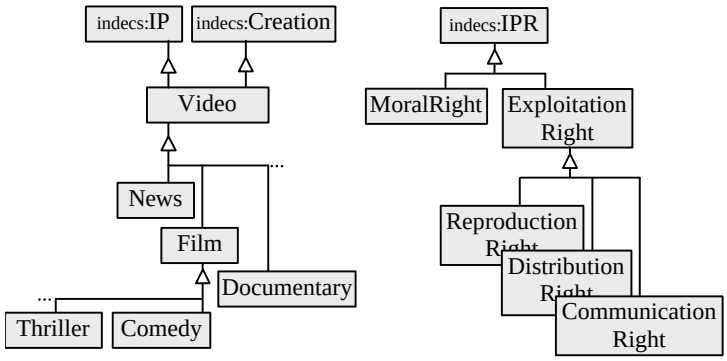


Figure 5: Some concepts defined in mars-schema

3.1.3 Metadata

Now we have the methods to refer to resources and some words to say things about them. There is an example of metadata in *table 1*, a set of RDF statements serialized in XML form about a video documentary. This is only a video description.

Table 1: Video resource RDF annotation

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1#"
          xmlns:mars="http://dmag.upf.i2/schemas/mars/video#"
>
<mars:Documentary rdf:ID="urn:mars:3928172">
    <dc:title>Climate change</dc:title>
    <dc:description>
        Some reseach groups. . .of the planet.
    </dc:description>
    <dc:language>en</dc:language>
    <dc:date>1999-06-27</dc:date>
    <mars:markedCopy
        rdf:resource="ftp://mmm.upc.i2/30m-climatics.mpg"/>
    <mars:preview
        rdf:resource="vstcp://192.168.48.3:5000/climatics.mpi/>
</mars:Documentary>
</rdf:RDF>
```

A more complex example dealing with IPR statements is shown in *figure 6*. It presents a graph view of a set of statements modelling an IPR agreement between two hypothetic parties, *Gamma Productions* and *Wide Distributions*. The former is transferring the *Reproduction Right* for a limited time to the latter. Furthermore, it is licensing the dissemination of a concrete number of copies worldwide for the same period using the Internet. *Gamma Productions* revenue is a fixed payment of 1000€ plus a 5% royalty rate. This agreement can be easily expressed using the RDF Data Model and consequently serialised in its XML form.

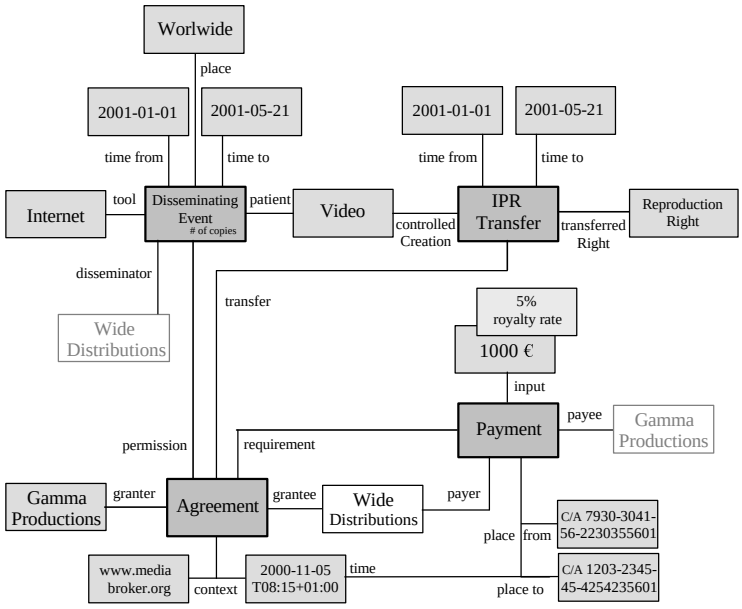


Figure 6: Graph view of an IPR agreement

It can serve as an example of the great expressive power that can be achieved using the basic tools of the Semantic Web, RDF and RDFSchema.

The conclusion of this part is that, although not visible, the important thing behind these chunks of metadata is their hidden semantics that emerge from the structure and interrelations of the used RDFSchemas. Now, we will move to the application logic level, where all this knowledge is really used. There, in an automated environment, they will be more significant.

3.2 Application logic layer

The broker provides a CORBA [16] interface to the entities willing to use their services. The actors in the considered model, especially content providers and distributors, can use this interface or http encapsulations of it with Java Servlets [17]. The remaining actors, the purchasers, will interact with the system through more user-friendly interfaces, distributors will provide them with web video-shops.

Available services for the business model roles are shown in *table 2*.

Table 2: Available services for the business model roles

Affiliated (any registered user)
- affiliate(string rdfUserDescription)
- login(string affiliatedId, string password)
- getRightsHolderRole()
- getDistributorRole()

RightsHolder (content provider)
- register(string rdfVideoDescription)
- offer(string videoId, string markedProductLoc, string markKey)

Distributor (or a purchaser through distributor's web shop interface)
- getSchemas()
- search(string xmlQuery)
- buy(string markedVideoId)

For the moment, the logical layer focuses on facilitating storage and retrieval of video descriptions and IPR statements about these videos. Therefore, the real work carried out in this layer reduces to profit from semantic annotations to improve searches for registered videos.

Queries are sent using the *search* method of the *Distributor* role. Actually, a XML formatted string is fetched containing a multiple property-value query with logic connectives. This allows easy integration with common web shop's forms without restricting the available properties. The used schemas determine the available properties and they can be listed with the *getSchemas* method. The broker contains an independent module to map XML to internal queries so new and more expressive query formats can be plugged in.

Future work, presented in section 4, will try to extract its full potential from the knowledge layer, which will also be improved. These enhancements would allow more sophisticated services, some of them sketched then.

3.3 Storage layer

Supporting all the system there is the storage layer, it provides the necessary persistence. A RDF oriented mechanism has been adopted, the RDF Data Model is directly stored in a database in its triple form. This allows a Relational Database implementation using only a table for triples. We will refer to it as the monolithic approach.

However, to avoid redundant storage of entities with many properties, or entities or literals referred as values of diverse properties, we have taken them out of the main table. To improve efficiency, digests of all entity and property identifiers, i.e. URIs, are used as the real table identifiers.

Consequently, entities and literals were stored in different tables, but this introduced a lot of problems when translating user queries to SQL ones, they became really complicated. We opted to unify both tables and to add a new attribute, *isLiteral*, to easily differentiate literals from entities. This increases searches efficiency; when we find a literal, as there cannot be properties applying to it, we can stop searching for them.

The database design can be seen in *figure 7*.

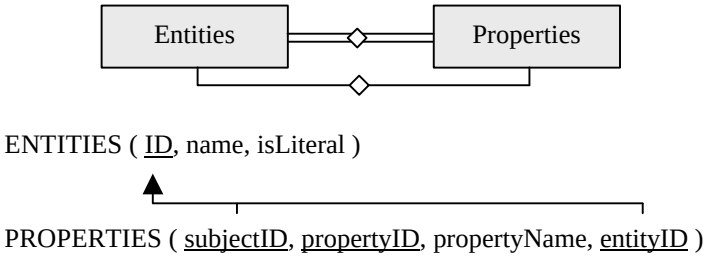


Figure 7: EER diagram and table intensions

A completely knowledge-independent storage medium has been obtained. The database does not suffer any change when new schemas are developed or reused. Even the previous and new schemas are stored in the same medium, because they are expressed using RDF and, consequently, they have a triple form that allows their transparent storage in the database.

3.4 Presentation layer

Finally, this layer contains all that purchasers will see when interacting with MARS services. As mentioned before, the user will interact through the Web pages of the distributors' video shops. These pages are totally out of the main MARS development process, though we have implemented a minimal web shop for testing purposes. In addition, we have developed a tool to facilitate that distributors use broker's search capabilities without great effort.

The tool takes the form of an easily integrable Java applet. It provides a palette of properties and classes for each of the available schemas and, when a concrete property is selected, help on possible values. *Figure 8* shows a capture of it.

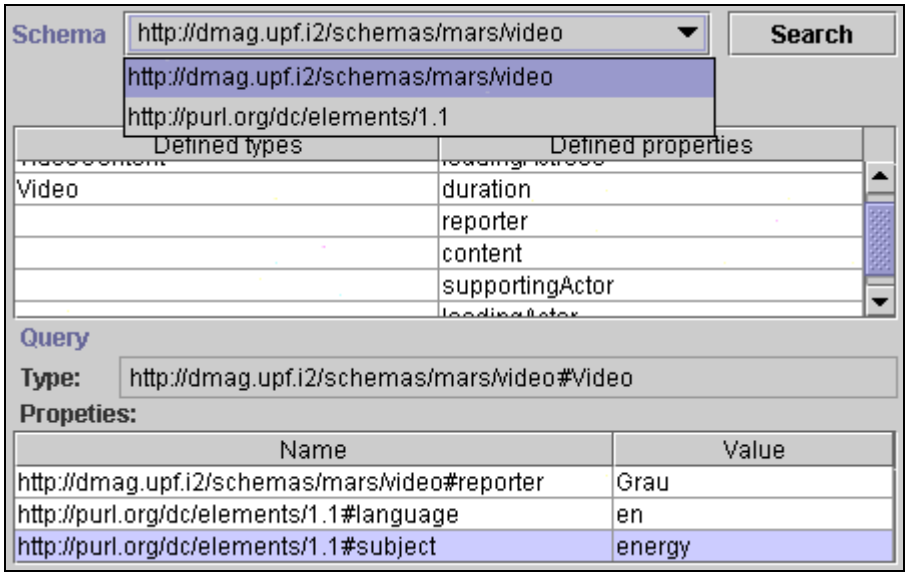


Figure 8: Query construction applet.

4 Future work

Not all the planned work has been finished and even the completed part needs an update, since new advances in the field are continuous. Therefore, there is still a lot of work to be done. Some of these future intentions are depicted in this section. They are presented following the layered structure of the architecture, each improvement is described in the section where it takes place.

4.1 Knowledge layer

This layer is one of the more affected by the future advances of the Semantic Web, consequently it is going to cope with a great part of the future work.

4.1.1 Ontologies

In the future, we are going to stop talking about schemas. New languages on top of RDFSchema have been developed and they allow expressing what now can be considered as

real ontologies. Therefore, future plans in the knowledge layer will centre in upgrading the developed schemas to the possibilities offered by ontology capable languages, like DAML+OIL [18].

Moreover, we are going to substitute the current upper levels inherited from the INDECS model with those from another ontology really focused on this matter, like SUO [19]. Nevertheless, the more concrete levels of INDECS, mainly those concerned with IPR, will be maintained but adapted to be founded in the new conceptual base level. It will not be really a replacement process, it will be implemented, if possible, by cross mapping the old and the new upper levels. Acting this way will allow greater interoperability and backward compatibility between the different versions of MARS. Therefore, even INDECS unaware application would communicate, or at least partially understand all the metadata generated by MARS.

Finally, one of the problems that have come out is the sparse diffusion of metadata in the Web. To reduce its impact, a lexical layer will be put under the main part of the application ontology. This will allow ontology driven information extraction from natural language. Metadata will be easier to produce or will be automatically retrieved, for instance from Web pages. This can be faced using a lexicon like Wordnet [20], although it is limited to English. However, recently the EuroWordnet [21] project has extended it to other European languages.

4.1.2 Trust

As commented in section 3.1, the use of public key infrastructures has been moved to future work. The primary intention is requiring that each actor taking part in the systems have its own digital certificate. This certificate, with its corresponding private key, will be used for digitally signing all the statements done by this actor (agreements, offers, assertions…) so responsibility can be tracked later and even produce contracts.

We are planning to apply digital signatures at the RDF Data Model level [22], not to its serialised form. This approach avoids ambiguity problems while maintaining the flexibility RDF annotations provide.

4.2 Application logic layer

This layer may also receive a lot of work. However, unlike the previous one, it will not have the priority. This is because this layer feeds on the semantics generated by the knowledge layer. Therefore, the new extensions will be undertaken when all profit from a well-developed knowledge level can be taken.

However, a research line classifiable under this section is already been designed and first implementations are been produced. It is a mobile agent platform related to the IPR control service. It would consist of a brigade of mobile agents patrolling the Web randomly or focusing on suspicious user constrained zones. Their mission will be, when possible, to automatically test the IPR situation of the found multimedia items against available IPR databases [23]. In addition, we are considering automatic, or partially assisted, negotiation on multimedia assets carried out by agents [24].

However, to enable sophisticated and highly automated implementations, more powerful tools are necessary. The Semantic Web has been until now very focused on representation issues, and less effort has been made to exploit these representations. That is reasonable

because the latter needs that the former is well established before advances can be done. Following these trends, we are now considering Conceptual Graphs [25] as on of the formalisms to apply to the logical level.

Conceptual Graphs, due to their graph oriented philosophy, seem well suited as RDF/RDFSchema extensions. Indeed, some proposals to map between Conceptual Graphs and RDF/XML have been done [26]. Conceptual Graphs can also be translated to and from predicate logics, so integration with other initiatives in the field is possible, like DAML-L [27].

4.3 Storage layer

The intention is to continue using common relational databases, they are widely used and this facilitates implementation and deployment. However, some improvements can be done, for instance using Object Relational Databases. They provide *subtable* relations between tables that simplify queries on subtype hierarchies, this allows simplifying the current complex and inefficient SQL queries.

We are also considering using pre-build RDF storage packages, like the ICS-FORTH [28] RDFSuite. This package provides advanced storage of RDF metadata and enhanced mechanisms for retrieval, a SQL like language and RDF oriented called RQL.

4.4 Presentation layer

Finally, we are also planning to use Conceptual Graphs in the user. We can profit from its graphical orientation to develop more intuitive interfaces, allowing a more sophisticated level of interaction. For instance, graphical conceptual graphs editors could guide user interaction, during resource description or query formulation, profiting from domain knowledge, i.e. ontologies. Therefore, the editor can recommend best-suited choices or even warn user about inconsistent constructions. *Figure 9* shows its possible appearance.

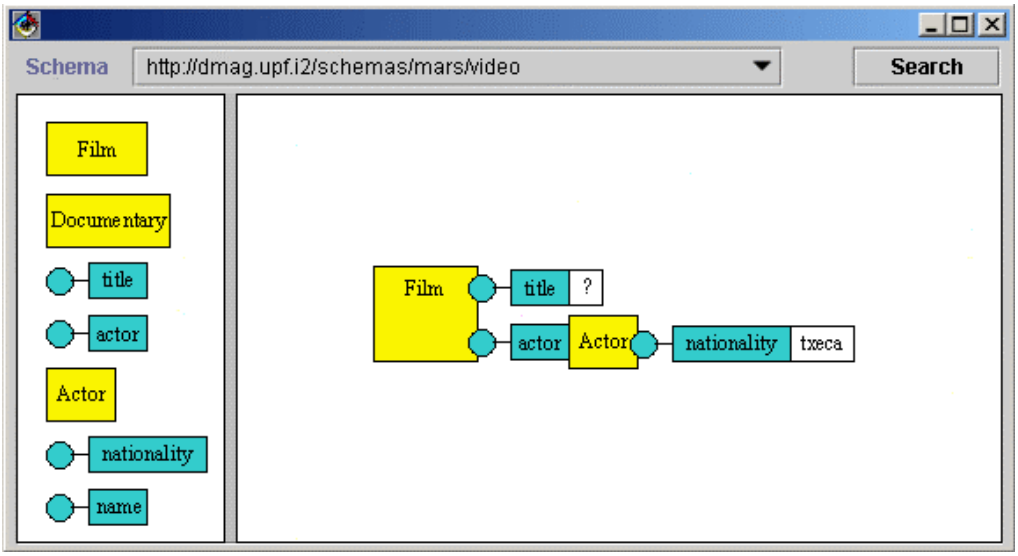


Figure 9: Drawing of a feasible query interface using graphs and ontologies palette.

5 Conclusions

Although the Semantic Web is at its beginnings, in our experience in the MARS project it has demonstrated its high potential. The promise of interoperability at the semantic level seems very appropriate in a heterogeneous domain with great expressiveness requirements, like IPR management.

Most significant improvements must be done in the business models, and it is going to be difficult to satisfy all the participants, from customers to authors. Notwithstanding, technology can provide a very valuable help in this process. Moreover, there are many other environments where this approach could give great results. Those tightly related to the Web approach (openness, decentralisation, universality…) could be specially benefited.

From a more general point of view, it has been interesting to see the benefits that a knowledge driven approximation to software development can provide. Traditionally, in the software development environment, there is a knowledge level phase were a model of the domain of application is done. Nevertheless, the produced model is only used for human consumption. When the project enters the machine aware part most of the produced conceptual value is lost. For instance, we can think about relational database models, the SQL implementation is the only view that the application has, while the Extended Entity Relationship diagram is exclusively used for documentation purposes.

Therefore, it would be interesting to profit from this initial effort, making the conceptual model machine available. The Semantic Web faces this problem using ontologies. Nevertheless, this is not revolutionary, the new value it adds is inherited from the previous Web. The Web provided the rhizome[1] approach to the information level, where the rhizome approach stands for a hierarchy less, open and decentralised way of organisation [29]. This approach, applied to information, has showed as the best suited in an Internet-connected world. Therefore, the novelty, and the challenge, is to apply it to the knowledge level, i.e. constructing a Web of interrelated ontologies.

---

[1] The rhizome serves as a metaphor for the multiplicity and infinite interconnectedness of all thought, life, culture, and language. Developed by French theorists Gilles Deleuze and Felix Guattari in their book "A Thousand Plateau's" [29], from which there is an interesting quote:

"A rhizome ceaselessly establishes connections between semiotic chains, organizations of power, and circumstances relative to the arts, sciences, and social struggles. A semiotic chain is like a tuber agglomerating very diverse acts, not only linguistic, but also perceptive, mimetic, gestural, and cognitive: there is no language in itself, nor are there any linguistic universals, only a throng of dialects, patois, slangs, and specialized languages. There is no ideal speaker-listener, any more than there is a homogeneous linguistic community.... There is no mother tongue, only a power takeover by a dominant language within a political multiplicity. Language stabilizes around a parish, a bishopric, a capital. It forms a bulb. It evolves by subterranean stems and flows, along river valleys or train tracks; it spreads like a patch of oil. It is always possible to break a language down into internal structural elements, an undertaking not fundamentally different from a search for roots. There is always something genealogical about a tree. It is not a method for the people. A method of the rhizome type, on the contrary, can analyse language only by decentering it onto other dimensions and other registers. A language is never closed upon itself, except as a function of impotence."

References

[1] Flichy, P. (translation Libbrecht, L.) “Dynamics of Modern Communication : The Shaping & Impact of New Communications Technologies”. ISBN: 0 803 97851 0. Sage Publications, 1995.

[2] W3C Workshop on Digital Rights Management for the Web, 2001. http://www.w3.org/2000/12/drm-ws

[3] MARS project, http://www.upf.es/esup/dmag

[4] The evolution of a specification (Evolvability), Tim Berners-Lee, 1998. http://www.w3.org/designissues/evolution.html

[5] IMPRIMATUR Project, http://www.imprimatur.net

[6] Gallego, I., Delgado, J., Acebron, J.J. “Distributed models for brokerage on electronic commerce”, in “Trends in distributed systems for electronic commerce”. ISBN: 3 540 64564 0. Springer, Germany, 1998, pp. 129-140.

[7] Delgado, J., Gallego, I., Polo, J. "Electronic commerce of multimedia services", in “MultiMedia Modeling” (invited paper). ISBN: 981 02 4146 1. World Scientific Publishing, Singapur, 1999, pp. 97-110.

[8] Manasse, M.S. “Why Rights Management is wrong (and what to do instead)”. W3C Workshop on Digital Rights Management for the Web, 2001. http://www.w3.org/2000/12/drm-ws/compaq.html

[9] Lassila, O., Swick, R.R. (editors). “Resource Description Framework (RDF) Model and Syntax Specification”. W3C Recommendation 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax

[10] Brickley, D., Guha, R.V. (editors). “Resource Description Framework (RDF) Schema Specification 1.0”. W3C Candidate Recommendation 27 March 2000. http://www.w3.org/TR/rdf-schema

[11] Naming and Addressing: URIs, URLs,... http://www.w3.org/Addressing

[12] The Digital Object Identifier, http://www.doi.org

[13] Dublin Core Element Set, http://dublincore.org/documents/dces

[14] MPEG7, http://www.cselt.it/mpeg

[15] INteroperability of Data in E-Commerce Systems (INDECS) Framework Ltd, http://www.indecs.org

[16] Common Object Request Broker Architecture, http://www.corba.org

[17] Java Servlets, http://java.sun.com/products/servlet

[18] DAML+OIL (DARPA Agent Markup Language + Ontology Interchange Language), http://www.daml.org/2001/03/daml+oil-index.html

[19] IEEE Standard Upper Ontology (SUO), http://ltsc.ieee.org/suo

[20] Wordnet, http://www.cogsci.princeton.edu/~wn

[21] Eurowordnet, http://www.hum.uva.nl/~ewn

[22] RDF API Draft, Cryptographic digests of RDF models and statements. http://www-db.stanford.edu/~melnik/rdf/api.html#digest

[23] Gallego, I., Delgado, J., García, R. “Use of Mobile Agents for IPR Management and Negotiation”, in “Mobile Agents for telecommunication applications”. Lecture Notes in Computer Science 1931, MATA 2000. ISBN: 3 540 41069 4. Springer, Germany, 2000, pp. 205-213.

[24] Delgado, J., Gallego, I. “Negotiation of Copyright in E-Commerce of Multimedia Publishing Material”, in “Electronic Publishing’01. 2001 in the Digital Publishing Odyssey”. ISBN 1 58603 191 0. IOS Press, 2001, pp. 298-306.

[25] Conceptual Graphs Standard, http://www.cs.uah.edu/~delugach/CG

[26] Corby, O., Dieng, R., Hébert, C A. “Conceptual Graph Model for W3C Resource Description Framework”. In Proceedings of ICCS 2000, LNAI 1867, 2000.

[27] Lassila, O., van Harmelen, F., Horrocks, I., Hendler, J., McGuinness, D.L. “The semantic Web and its languages”. IEEE Intelligent Systems, Volume: 15 Issue: 6, 2000.

[28] ICS-FORTH RDFSuite, http://www.ics.forth.gr

[29] Deleuze, G., Guattari, F. “Mil plateaux (capitalisme et schizophrénie)”, 1980. Ed. de Minuit.

Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology

Jane Hunter
DSTC Pty Ltd
University of Qld, Australia
jane@dstc.edu.au

Abstract. For the past two years the Moving Pictures Expert Group (MPEG), a working group of ISO/IEC, have been developing MPEG-7 [1], the "Multimedia Content Description Interface", a standard for describing multimedia content. The goal of this standard is to develop a rich set of standardized tools to enable both humans and machines to generate and understand audiovisual descriptions which can be used to enable fast efficient retrieval from digital archives (pull applications) as well as filtering of streamed audiovisual broadcasts on the Internet (push applications). MPEG-7 is intended to describe audiovisual information regardless of storage, coding, display, transmission, medium, or technology. It will address a wide variety of media types including: still pictures, graphics, 3D models, audio, speech, video, and combinations of these (e.g., multimedia presentations). MPEG-7 is due for completion in October 2001. At this stage MPEG-7 definitions (description schemes and descriptors) are expressed solely in XML Schema [2-4]. XML Schema has been ideal for expressing the syntax, structural, cardinality and datatyping constraints required by MPEG-7. However it has become increasingly clear that in order to make MPEG-7 accessible, re-usable and interoperable with other domains then the semantics of the MPEG-7 metadata terms also need to be expressed in an ontology using a machine-understandable language. This paper describes the trials and tribulations of building such an ontology represented in RDF Schema [5] and demonstrates how this ontology can be exploited and reused by other communities on the semantic web (such as TV-Anytime [6], MPEG-21 [7], NewsML [8], museum, educational and geospatial domains) to enable the inclusion and exchange of multimedia content through a common understanding of the associated MPEG-7 multimedia content descriptions.

1. Introduction

Audiovisual resources in the form of still pictures, graphics, 3D models, audio, speech, video will play an increasingly pervasive role in our lives, and there will be a growing need to enable computational interpretation and processing of such resources. Forms of representation that will allow some degree of machine interpretation of audiovisual information's meaning will be necessary [27]. The goal of MPEG-7 [1] is to support such requirements by providing a rich set of standardized tools to enable the generation of audiovisual descriptions which can be understood by machines as well as humans and to enable fast efficient retrieval from digital archives (pull applications) as well as filtering of streamed audiovisual broadcasts on the Internet (push applications).

The main elements of the MPEG-7 standard are:

- Descriptors (D), representations of Features, that define the syntax and the semantics of each feature representation;
- Description Schemes (DS) that specify the structure and semantics of the relationships between their components. These components may be both Descriptors and Description Schemes;

- A Description Definition Language (DDL) to allow the creation of new Description Schemes and, possibly, Descriptors and to allows the extension and modification of existing Description Schemes;
- System tools, to support multiplexing of descriptions, synchronization of descriptions with content, transmission mechanisms and coded representations (both textual and binary formats) for efficient storage and transmission, management and protection of intellectual property in MPEG-7 descriptions.

XML Schema language has been chosen as the DDL [9] for specifying MPEG-7 descriptors and description schemes because of its ability to express the syntactic, structural, cardinality and datatyping constraints required by MPEG-7 and because it also provides the necessary mechanisms for extending and refining existing DSs and Ds. However it has recently become increasingly clear that there is also a need for a machine-understandable representation of the semantics associated with MPEG-7 DSs and Ds to enable the interoperability and integration of MPEG-7 with metadata descriptions from other domains. New metadata initiatives such as TV-Anytime [6], MPEG-21 [7], NewsML [8], and communities such as the museum, educational, medical and geospatial communities, want to combine MPEG-7 multimedia descriptions with new and existing metadata standards for simple resource discovery (Dublin Core [10]), rights management (INDECS [11]), geospatial (FGDC [12]), educational (GEM [13], IEEE LOM [14]) and museum (CIDOC CRM [15]) content, to satisfy their domain-specific requirements. In order to do this, there needs to be a common understanding of the semantic relationships between metadata terms from different domains. XML Schema provides little support for expressing semantic knowledge. RDF Schema provides us with a way to do this.

The Resource Description Framework (RDF) [16] is the accepted language of the semantic web due to its ability to express semantics and semantic relationships through class and property hierarchies. In this paper, we investigate the feasibility of expressing the semantics of MPEG-7 Descriptors (Ds) and Description Schemes (DSs) in an RDF Schema [5] ontology. An earlier paper evaluated RDF Schema for video metadata representation (prior to the development of MPEG-7) and determined a number of limitations [23]. In this paper we hope to ascertain whether those limitations still exist when representing the semantics of MPEG-7 DSs and Ds or whether they can be overcome – either by using the extra constraints provided by DAML+OIL [17] or through combining RDF Schema semantics with XML Schema encoding specifications in a complementary manner.

Whilst manually building the RDF Schema for a core subset of MPEG-7, we also hope to be able to recognize patterns and hence determine automatic mechanisms for generating compatible RDF Schema definitions corresponding to the complete set of MPEG-7 XML Schema definitions.

In Section 2 we describe the methodology, problems encountered and results of building an RDF Schema ontology for MPEG-7. In Section 3 we describe how the RDF Schema semantic definitions for MPEG-7 can be linked to their corresponding pre-existing XML Schema definitions (or recommended encodings). In Section 4 we describe how the MPEG-7 RDF Schema can be merged with RDF schemas from other domains to generate a single "super-ontology" called MetaNet. Expressed in DAML+OIL [17], MetaNet can be used to provide common semantic understanding between domains. Finally we illustrate how this super-ontology can be used to enable the co-existence of interoperability, extensibility and diversity within metadata descriptions generated by integrating metadata terms from different domains.

2. Building the Ontology

During the early development stages of MPEG-7, Unified Modelling Language (UML) [18] was used to model the entities, properties and relationships (description schemes and descriptors) which comprised MPEG-7. However the massive size of the specification (the Multimedia Description Schemes specification [19] is almost 800 pages and that is only one part out of 7 parts) combined with the belief that the UML models were a development tool only, which duplicated information in the XML schemas, led to the decision to drop them from the final specifications.

Although the lack of an existing data model hampered the development of an RDF Schema ontology, it also means that the generated RDF Schema will be even more valuable - providing both a data model as well as definitions of the semantics of the MPEG-7 terms and the relationships between them. Building the data model and schema should also highlight any inconsistencies, duplication or ambiguities which exist across the large number of MPEG-7 description schemes and descriptors.

Without a data model to build on, the class and property hierarchies and semantic definitions had to be derived through reverse-engineering of the existing XML Schema definitions together with interpretation of the english-text semantic descriptions. To simplify the process, we used a core subset of the MPEG-7 specification together with a top-down approach to generate the ontology described here. An additional very helpful mechanism for determining the data model was to generate the DOM (Document Object Model) for the XML Schema (using XML Spy). This graphical representation of the structures helped determine the class and property hierarchies.

The first step was to determine the basic multimedia entities (classes) and their hierarchies from the Multimedia Description Scheme (MDS) basic entities [19]. This process is described in Section 2.1. Next the structural hierarchies were determined from the Segment Description Schemes (Section 2.2). Section 2.3 describes the non-multimedia entities defined within MPEG-7. Section 2.4 describes the different multimedia and generic properties associated with the multimedia entities. Sections 2.5 describes the RDF Schema representations of the MPEG-7 visual and audio descriptors defined in [20] and [21] respectively.

2.1 Top-level MPEG-7 Multimedia Entities

The top-level Multimedia Content entities are described in Section 4.4 of the MDS FCD [19]. The RDF class hierarchy corresponding to these basic entities is illustrated in Figure 1 and the RDF Schema representation of these entities and relationships is shown in Appendix A. Within MPEG-7, multimedia content is classified into five types: *Image*, *Video*, *Audio*, *Audiovisual* and *Multimedia*. Each of these types have their own segment subclasses.

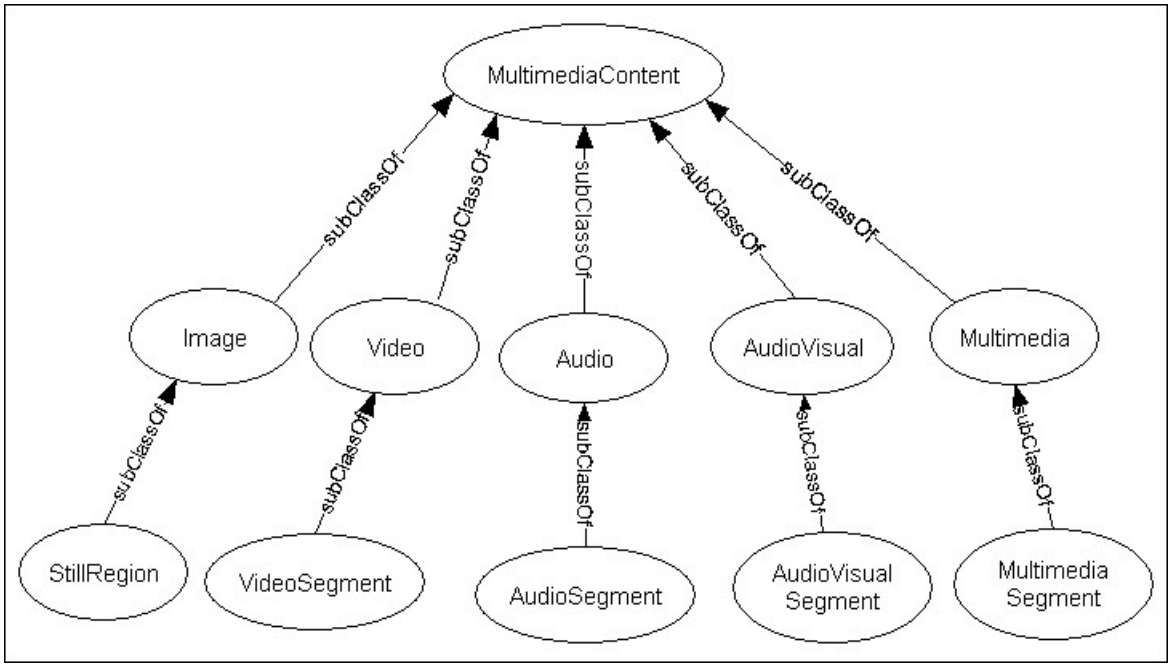


Figure 1: Class Hierarchy of MPEG-7 Top-level Multimedia Content Entities

2.2 MPEG-7 Multimedia Segments and Hierarchical Structures

MPEG-7 provides a number of tools for describing the structure of multimedia content in time and space. The Segment DS (Section 11 of [19]) describes a spatial and/or temporal fragment of multimedia content. A number of specialized subclasses are derived from the generic Segment DS. These subclasses describe the specific types of multimedia segments, such as video segments, moving regions, still regions and mosaics, which result from spatial, temporal and spatiotemporal segmentation of the different multimedia content types. Table I describes the different types of MPEG-7 segments and Figure 2 illustrates the corresponding segment class hierarchy.

Segment	Fragment or segment of multimedia content.
StillRegion	2D spatial regions of an image or video frame.
ImageText	Spatial regions of an image or video frame corresponding to text or captions.
Mosaic	Mosaics or panoramic view of a video segment.
StillRegion3D	3D spatial regions of a 3D image.
VideoSegment	Temporal intervals or segments of video data.
MovingRegion	2D spatio-temporal regions of video data.
VideoText	Spatio-temporal regions of video data that correspond to text or captions.
AudioSegment	Temporal intervals or segments of audio data.
AudioVisualSegment	Temporal intervals or segments of AV data.
AudioVisualRegion	Arbitrary spatio-temporal segments of AV data.
MultimediaSegment	Composites of segments that form a multimedia presentation.
EditedVideoSegment	Video segments that result from an editing work.

Table I: Semantic Definitions of MPEG-7 Segment Types

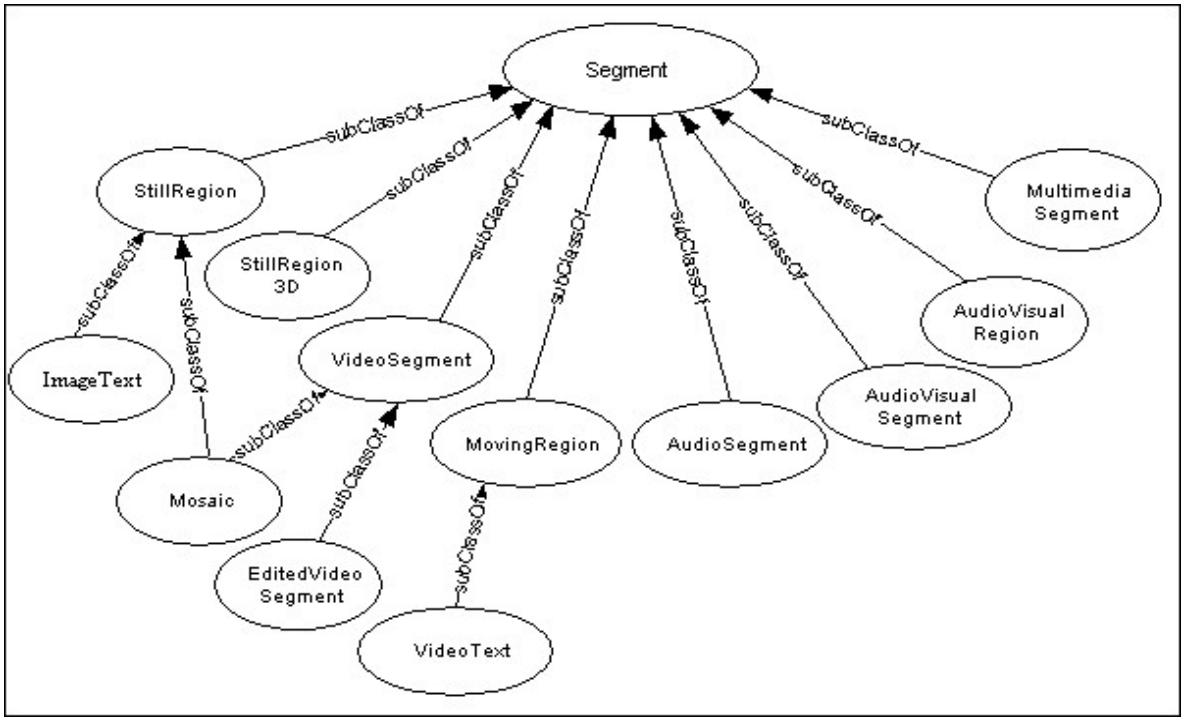


Figure 2: Class Hierarchy of MPEG-7 Segment Classes

The RDF Schema representation for the segment class hierarchy can be found in Appendix A. Certain segment entities, such as the VideoSegment, are subclasses of multiple superclasses i.e., both the Video class and the Segment class. The relationships of these segment types to the top-level multimedia entities is illustrated in Figure 3. Multimedia resources can be segmented or decomposed into sub-segments through 4 types of decomposition:

- Spatial Decomposition - e.g., spatial regions within an image;
- Temporal Decomposition - e.g., temporal video segments within a video;
- Spatiotemporal Decomposition - e.g., moving regions within a video;
- MediaSource Decomposition - e.g., the different tracks within an audio file or the different media objects within a SMIL presentation.

The different types of segment decomposition can be represented via an RDF property hierarchy. For example:

```
<rdf:Property rdf:ID="decomposition">
      <rdfs:label>decomposition of a segment</rdfs:label>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
 </rdf:Property>

 <rdf:Property rdf:ID="temporal_decomposition">
      <rdfs:label>temporal decomposition of a segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#decomposition"/>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
 </rdf:Property>
```

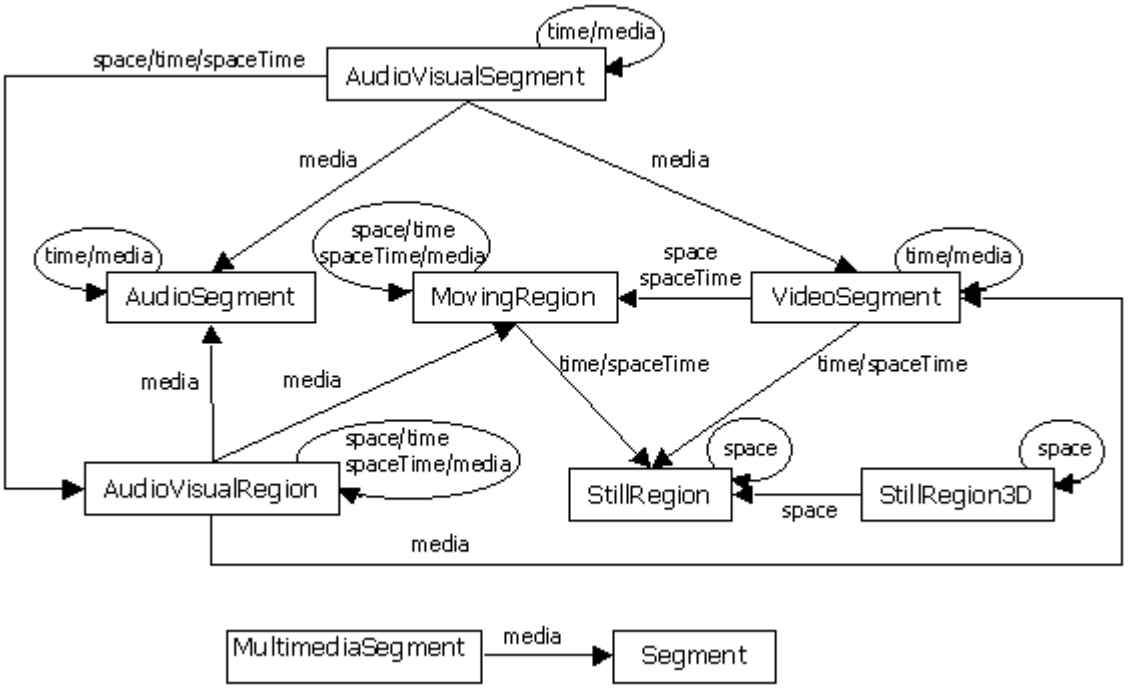


Figure 3: Valid decomposition relationships between MPEG-7 Segment Classes (from Figure 32 [19])

If we consider the decomposition of a *VideoSegment* then, we would like to constrain the temporal decomposition of *VideoSegments* into either smaller *VideoSegments* or *StillRegions*.

```
<rdf:Property rdf:ID="videoSegment_temporal_decomposition">
      <rdfs:label>temporal decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#temporal_decomposition"/>
      <rdfs:domain rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#StillRegion"/>
 </rdf:Property>
```

However this is illegal within RDF Schema because of the inability to specify multiple range constraints on a single property. This limitation was first recognized in [23] when RDF Schema was being considered as a candidate for the MPEG-7 DDL. The only way to express this within RDF Schema is to define a new superclass which merges the permissable range classes into a single common class.

DAML+OIL [17] permits multiple range statements but interprets the resulting range to be the intersection of the specified classes. In this case, we want to specify that the range will be an instance from the union of the two classes (*VideoSegment* and *StillRegion*). In order to do this we must use *daml:unionOf* to define a class which is the union of these two classes and then specify this new class as the range. For example:

```
<rdfs:Class rdf:ID="#VideoSegmentsOrStillRegions">
   <daml:unionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#VideoSegment"/>
      <rdfs:Class rdf:about="#StillRegion"/>
   </daml:unionOf>
</rdfs:Class>

<rdf:Property rdf:ID="videoSegment_temporal_decomposition">
      <rdfs:label>temporal decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#temporal_decomposition"/>
```

```
        <rdfs:domain rdf:resource="#VideoSegment"/>
        <rdfs:range rdf:resource="#VideoSegmentsOrStillRegions"/>
    </rdf:Property>
```

Also associated with the segment classes are the properties which define the location of a segment within its containing media object. These include such properties as: *mediaLocator*, *spatialLocator*, *mediaTime* (temporal locator) and *spatioTemporalLocator*. If the segment is non-continuous (i.e., the union of connected components) , then the *spatialMask*, *temporalMask*, *spatio-TemporalMask* and *mediaSpaceMask* properties may be applicable. These are sequences of spatial, temporal or spatiotemporal locators. Below we represent the temporalLocator or *mediaTime* property (which has two components, the mediaTimePoint (start of a segment) and the mediaDuration (length of the segment)):

```
<rdf:Property rdf:ID="mediaTime">
      <rdfs:label>temporal location of a video or audio segment</rdfs:label>
      <rdfs:domain rdf:resource="#Segment"/>
      <rdfs:range rdf:resource="#MediaTime"/>
 </rdf:Property>
<rdfs:Class rdf:ID="MediaTime">
     <rdfs:label>time point or interval within media</rdfs:label>
     <rdfs:subClassOf rdf:resource="#Time"/>
  </rdfs:Class>
<rdf:Property rdf:ID="mediaTimePoint">
      <rdfs:label>time point</rdfs:label>
      <rdfs:domain rdf:resource="#MediaTime"/>
      <rdfs:range rdf:resource="http://www. mpeg7.org/2001/MPEG-7_Schema# basicTimePoint"/>
 </rdf:Property>
 <rdf:Property rdf:ID="mediaDuration">
      <rdfs:label>temporal length of segment</rdfs:label>
      <rdfs:domain rdf:resource="#MediaTime"/>
      <rdfs:range rdf:resource=" http://www. mpeg7.org/2001/MPEG-7_Schema#basicDuration"/>
 </rdf:Property>
```

2.3 Basic Non-multimedia Entities within MPEG-7

As well as the multimedia entities described above, MPEG-7 defines a number of basic non-multimedia entities which are used in different contexts across MPEG-7. These include:

- Agent
    - Person
    - PersonGroup
    - Organisation
- Role
- Place
- Time
- Instrument

The RDF Schema representations of these classes can be found in Appendix A. The code below shows both the XML Schema definition for the Person *complexType.* Figure 4 shows corresponding the RDF model for the Person *Class*. This example illustrates how, in generating the RDF Schema, we have translated the children elements of the XML Schema *complexType* to properties attached to the RDF Schema *class*.

```
<complexType name="PersonType">
   <complexContent>
      <extension base="mpeg7:AgentType">
         <sequence>
            <element name="Name" type="mpeg7:PersonNameType"/>
            <element name="Affiliation" minOccurs="0" maxOccurs="unbounded">
              <complexType>
                <choice>
                  <element name="Organization" type="mpeg7:OrganizationType"/>
                  <element name="PersonGroup" type="mpeg7:PersonGroupType"/>
                </choice>
              </complexType>
            <element name="Address" type="mpeg7:PlaceType"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<complexType name="PersonNameType">
  <sequence>
    <choice minOccurs="1" maxOccurs="unbounded">
      <element name="GivenName" type="string"/>
      <element name="FamilyName" type="string"/>
    </choice>
  </sequence>
</complexType>
```

Again we have the situation where we would like to be able to say that the Affiliation property can have values which are instantiations of either the Organisation or PersonGroup class i.e., we would like to be able to define multiple possible ranges. DAML+OIL provides a way of doing this through the *unionOf* mechanism as shown below:

```
<rdfs:Class rdf:ID="Affiliation">
   <rdfs:comment>Either an Organisation or a PersonGroup</rdfs:comment>
   <daml:unionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#Organisation"/>
      <rdfs:Class rdf:about="#PersonGroup"/>
   </daml:unionOf>
</rdfs:Class>

<rdf:Property rdf:ID="affiliation">
     <rdfs:label>affiliation</rdfs:label>
     <rdfs:domain rdf:resource="#Person"/>
     <rdfs:range rdf:resource="#Affiliation"/>
</rdf:Property>
```

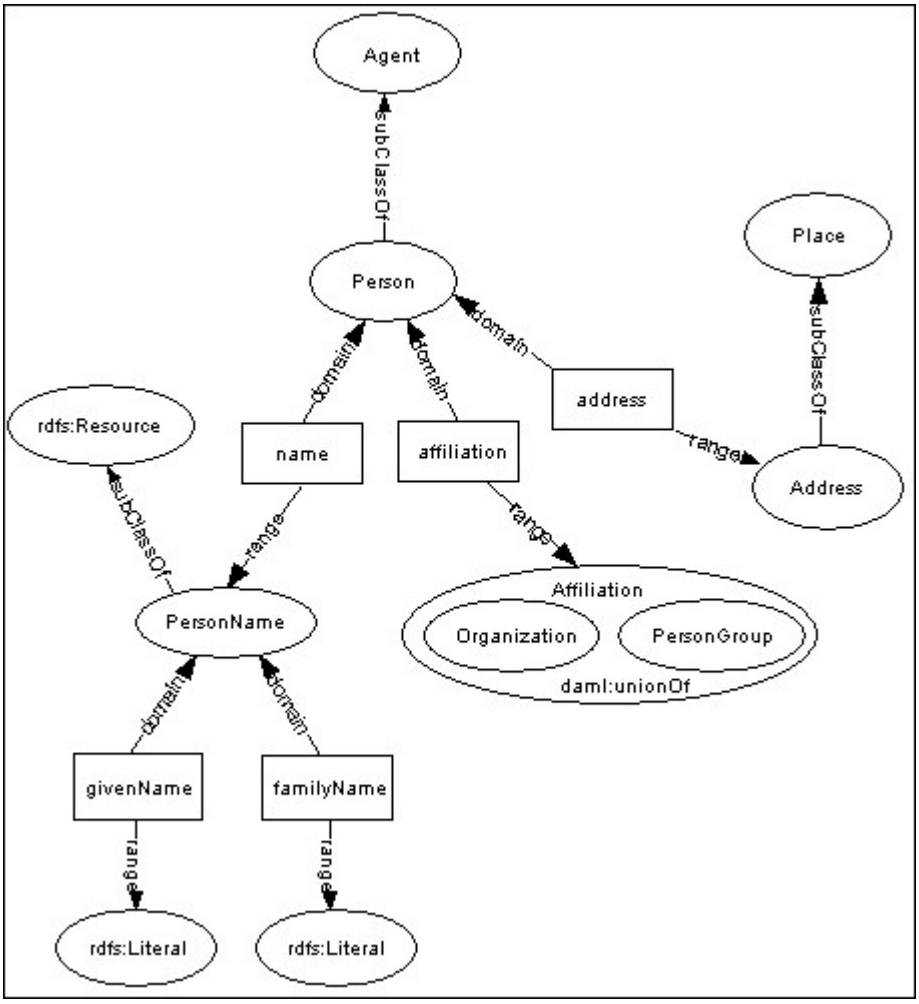


Figure 4: RDF Class and Property Representation of PersonDS

2.4 Multimedia Description Schemes

Figure 5 provides an overview of the organization of MPEG-7 Multimedia DSs into the following six categories: Basic Elements, Content Description, Content Management, Content Organization, Navigation and Access, and User Interaction. The MPEG-7 DSs in Figure 5 define descriptions which provide:

- Information describing the creation and production processes of the content (director, title, short feature movie);
- Information related to the usage of the content (copyright pointers, usage history, broadcast schedule);
- Media information of the storage features of the content (storage format, encoding);
- Structural information on spatial, temporal or spatio-temporal components of the content (scene cuts, segmentation in regions, region motion tracking);
- Information about low level features in the content (colors, textures, sound timbres, melody description);
- Conceptual, semantic information of the reality captured by the content (objects and events, interactions among objects);
- Information about how to browse the content in an efficient way (summaries, views, variations, spatial and frequency subbands);

- Organization information about collections of objects and models, which allows multimedia content to be characterized on the basis of probabilities, statistics and examples;
- Information about the interaction of the user with the content (user preferences, usage history)

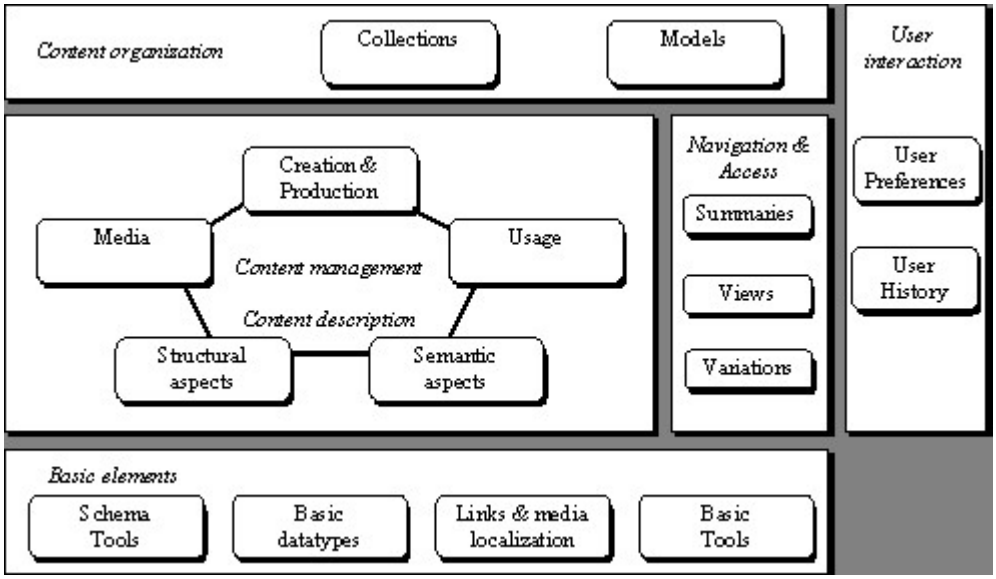


Figure 5 - Overview of MPEG-7 Multimedia DSs (from Figure 1 [19])

We will not cover all of these DSs in this paper but have chosen to represent only the CreationDS in order to demonstrate RDF Schema's ability to model a typical MPEG-7 DS. Figure 6 illustrates the RDF Schema classes and properties corresponding to the CreationDS (expressed in XML Schema) below.

```
<complexType name="CreationType">
   <complexContent>
      <extension base="mpeg7:DSType">
         <sequence>
            <element name="Title" type="mpeg7:TitleType"/>
            <element name="Abstract" type="mpeg7:TextAnnotationType"/>
            <element name="Creator">
              <complexContent> <extension base="mpeg7:AgentType">
                <complexType>
                  <sequence>
                     <element name="Role" type="mpeg7:ControlledTermType"/>
                     <element name="Instrument" type="mpeg7:CreationToolType"/>
                  </sequence>
                </complexType>
              </extension></complexContent>
            </element>
            <element name="CreationLocation" type="mpeg7:PlaceType"/>
            <element name="CreationDate" type="mpeg7:DateType"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

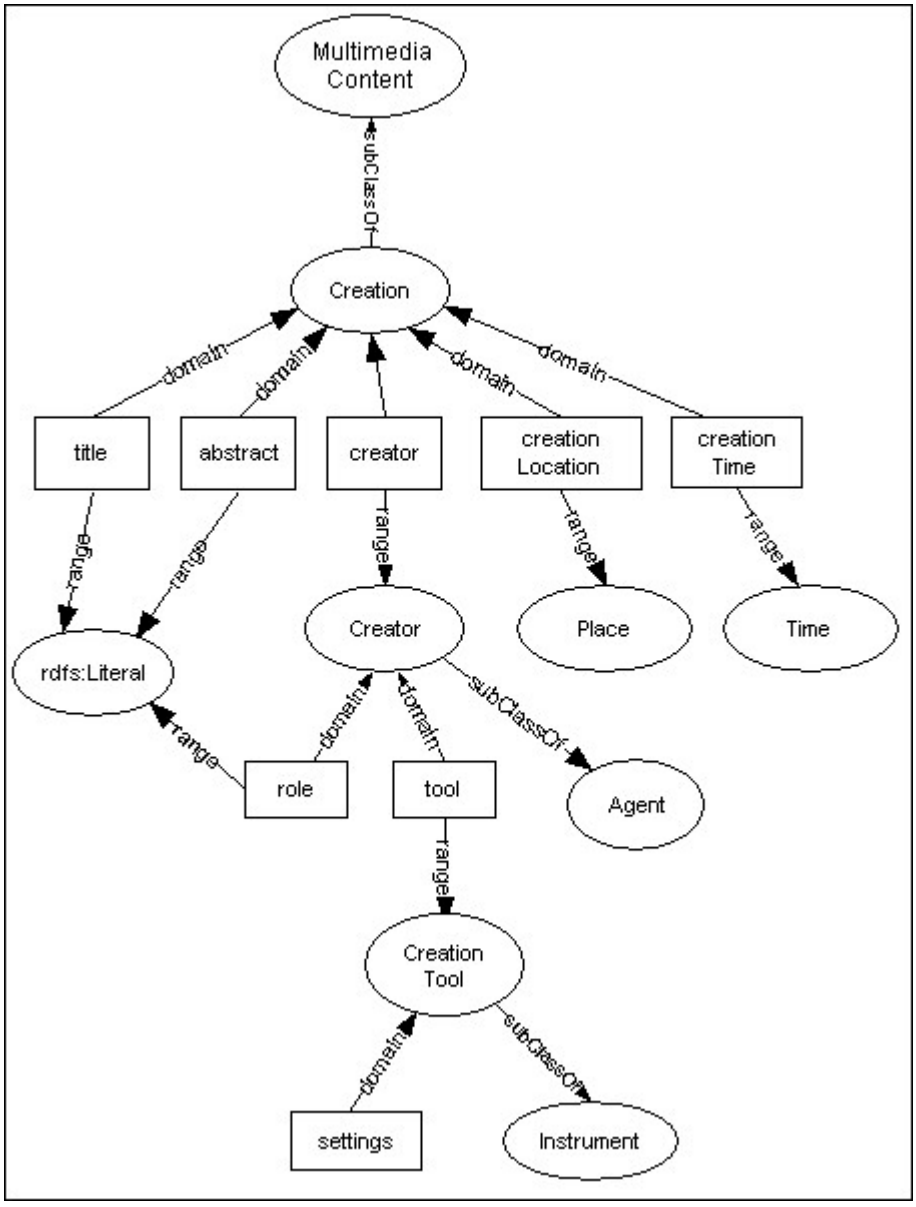


Figure 6 – RDF Class and Property Representation of MPEG-7 Creation DS

2.5 Low Level Visual and Audio Descriptors

The set of features or properties which is specific to the visual entities (Image, Video, AudioVisual, StillRegion, MovingRegion, VideoSegment) include:

- Colour
- Texture
- Motion
- Shape

Each of these features can be represented by a choice of descriptors. Table II below lists the visual features and their corresponding MPEG-7 descriptors. Precise details of the structure and semantics of these visual descriptors are provided in ISO/IEC 15938-3 FCD Multimedia Content Description Interface - Part 3 Visual [20].

Feature	Descriptors
Color	DominantColor
	ScalableColor
	ColorLayout
	ColorStructure
	GoFGoPColor
Texture	HomogeneousTexture
	TextureBrowsing
	EdgeHistogram
Shape	RegionShape
	ContourShape
	Shape3D
Motion	CameraMotion
	MotionTrajectory
	ParametricMotion
	MotionActivity

Table II: Visual features and their corresponding Descriptors

Similarly there is a set of audio features which is applicable to MPEG-7 entities containing audio (Video, AudioVisual, Audio, AudioSegment):

- Silence
- Timbre
- Speech
- Melody

ISO/IEC 15938-3 FCD Multimedia Content Description Interface - Part 4 Audio [21] describes in detail the XML Schema specifications of the audio descriptors. Each of these audio features can be represented by one or more audio descriptors. Table III below lists the audio descriptors which correspond to each audio feature.

Feature	Descriptors
Silence	Silence
Timbre	InstrumentTimbre
	HarmonicInstrumentTimbre
	PercussiveInstrumentTimbre
Speech	Phoneme
	Articulation
	Language
MusicalStructure	MelodicContour
	Rhythm
SoundEffects	Reverberation, Pitch, Contour, Noise

Table III: Audio features and their corresponding Descriptors

Only certain low-level visual and audio descriptors are applicable to each segment type. Table IV below illustrates the association of visual and audio features to different segment types. RDF Schema must be able to specify the constraints on these property-to-entity relationships.

Feature	Video Segment	Still Region	Moving Region	Audio Segment
Time	X	-	X	X
Shape	-	X	X	-
Color	X	X	X	-
Texture	-	X	-	-
Motion	X	-	X	-
Audio	X	-	-	X

Table IV: Relationships between Segment types and Visual and Audio Features

Using the color descriptor, we demonstrate in Figure 7, how RDF Schema is able express these constraints through the domain and range values in the color property definitions.

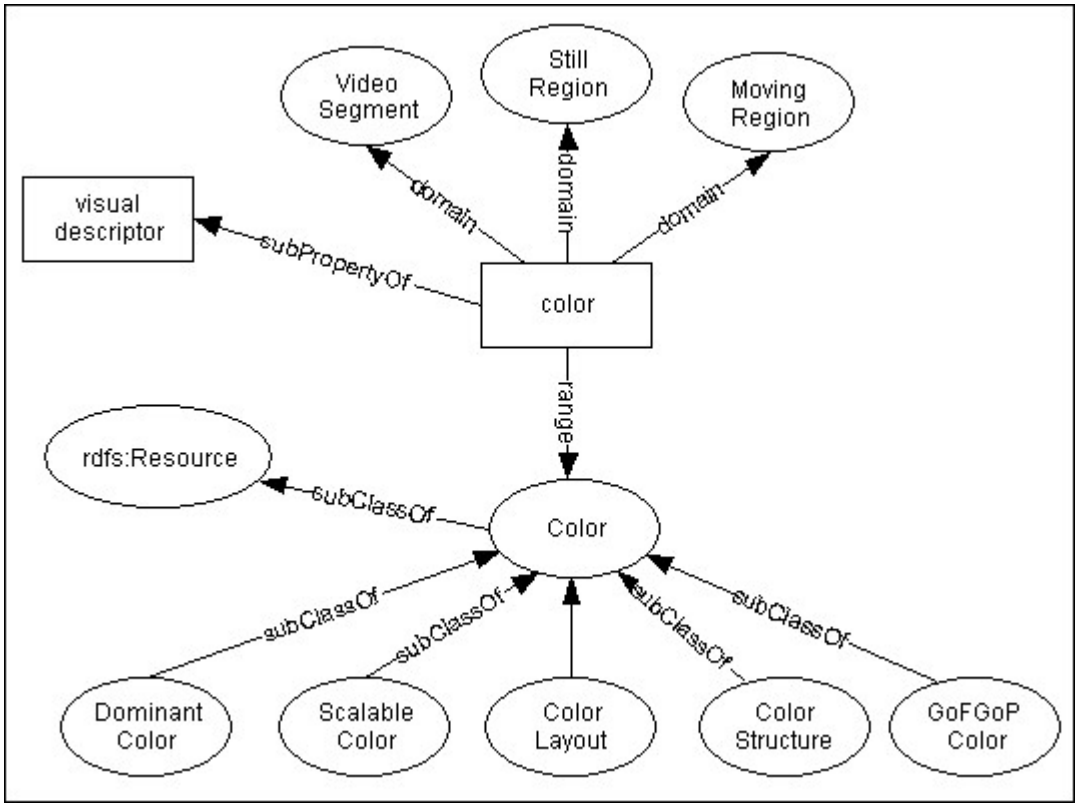


Figure 7: RDF Class and Property Representation of the MPEG-7 Color Descriptor

3. Linking the MPEG-7 XML and RDF Schemas

In a previous paper [22] we outlined the advantages of separating the semantics of domain-specific metadata terms from the recommended encodings by defining both an RDF Schema and an XML Schema in the domain's registered namespace. The RDF Schema file defines the domain-specific semantic knowledge by specifying type hierarchies and definitions - based on the ISO/IEC 11179 standard for the description of data elements. The XML Schema file specifies the recommended encodings of metadata elements and descriptions by defining types and elements, and their content models, structures, occurrence constraints and datatypes. In addition, the XML Schema file contains links to the corresponding semantic definitions in the RDF Schema file. Because the underlying semantics will remain relatively stable compared to the syntax, which will be application-dependent, we

choose to make the RDF Schema the base schema and to point to the base RDF Schema from the application-specific XML Schemas, rather than the other direction.

The most concise and flexible method for implementing the link from the XML Schema definitions to their corresponding RDF Schema definitions is to exploit the openness of XML Schema attributes. Since nearly all types are extended from the *openAttrs* type in the Schema for Schemas in [3], it is possible to extend XML Schema *simpleType* and *complexType* definitions with a "semantics" attribute defined in another namespace e.g., "XMLRDFSchemaBridge". Using this approach, the value of the "semantics" attribute is set to the RDF Property or Class which defines the semantics of the corresponding simple or complex type. We have chosen to link the semantics to XML Schema type definitions, rather than element declarations. This is because restrictions, extensions, redefinitions and elements are all built on top of XML Schema types, so the most logical and flexible approach is to attach the semantics to the type rather than the element. The XML Schema code below demonstrates an implementation of this approach.

```
<schema xmnls="http://www.w3.org/2001/10/XMLSchema"
    targetNamespace="http://www.mpeg7.org/2001/MPEG-7_Schema"
    xmlns:mpeg7="http://www.mpeg7.org/2001/MPEG-7_Schema"
    xmlns:xx="http://www.example.org/XMLRDFSchemaBridge">

    <annotation>
      <documentation>
        XML Schema for MPEG-7
      </documentation>
    </annotation>

    <simpleType name="Person"
            xx:semantics="http://www.mpeg7.org/2001/MPEG7_Schema/mpeg7.rdf#Person">
      <extension base="Agent"/>
    </simpleType>

    <simpleType name="Organisation"
          xx:semantics="http://www.mpeg7.org/2001/MPEG-7_Schema/mpeg7.rdf#Organisation>
      <extension base="Agent"/>
    </simpleType>
...

</schema>
```

4. Balancing Metadata Interoperability, Extensibility and Diversity

By making the semantic knowledge of a domain or community available in a machine-understandable RDF Schema, it becomes possible to merge separate ontologies or metadata vocabularies from different communities into a single encompassing ontology expressed using DAML+OIL. Using the ABC vocabulary ([24][28]), developed within the Harmony project [29], as the top-level or umbrella, we have manually developed a draft version of such a "super-ontology" - the MetaNet ontology [26]. MetaNet expresses the semantic relationships (e.g., equivalent, narrower, broader) between metadata terms from different domains. By linking the semantic knowledge provided by MetaNet with XSLT [25], we have been able to perform both the semantic and the structural and syntactic mapping required to map between XML-encoded metadata descriptions from different domains. The overall architecture of a system, which should enable the coexistence of metadata interoperability together with extensibility and diversity, is illustrated in Figure 8. The key components are:

- Domain-specific namespaces which express each domain's metadata model and vocabulary using both an RDF Schema and an XML Schema. Each XML Schema contains links to the corresponding RDF Schema;
- MetaNet - a single "super" metadata ontology, generated by merging the domain-specific ontologies (RDF Schemas) from different namespaces. This is expressed using DAML+OIL and will be based on a common underlying, extensible vocabulary such as the ABC vocabulary being developed within the Harmony project [24];
- XSLT - a language suitable for transforming between XML-encoded metadata descriptions. Combined with the semantic knowledge of MetaNet, XSLT [25] is capable of flexible dynamic mappings between application profile instantiations;
- Application Profiles - XML Schema definitions which combine, restrict, extend and redefine elements from multiple existing namespaces. Application profiles could also embed RDF Schema definitions of new classes or properties which are derived from classes and properties defined in the domain-specific RDF Schemas.

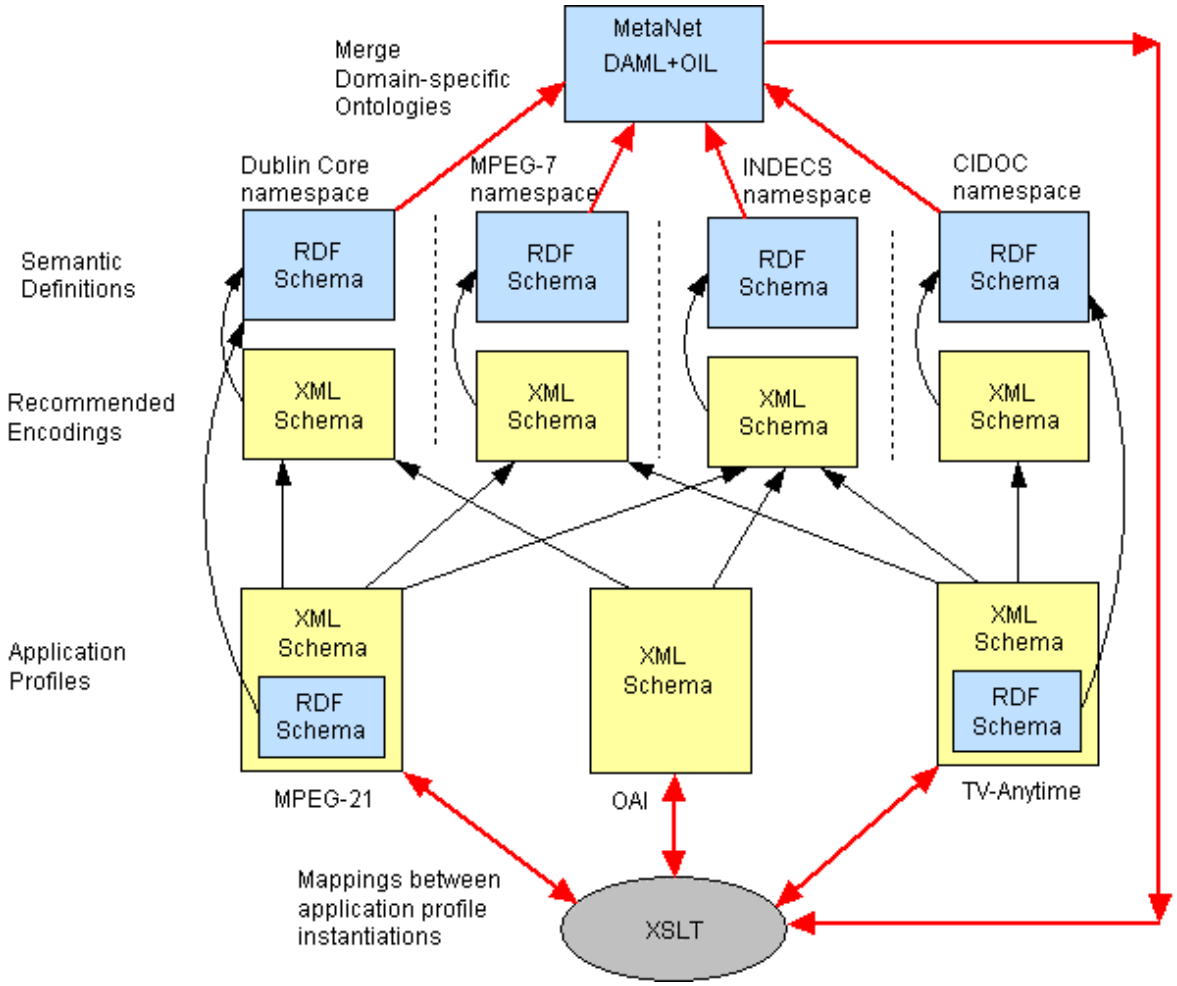


Figure 8: The Proposed Web Metadata Architecture

5. Conclusions

In this paper, we first outlined the reasons for why an RDF Schema representation of MPEG-7 is desirable. We then described the methodology, problems encountered and results of manually building an RDF Schema representation for a core subset of MPEG-7. Our

conclusion from this exercise is that, although RDF Schema is capable of expressing the semantics of MPEG-7 Description Schemes and Descriptors, it does have certain serious limitations. RDF Schema's property-centricity makes it difficult to generate property definitions and domain constraints from the class-centric XML Schema definitions. The inability to specify multiple range constraints or class-specific property constraints are other major limitations of RDF [23] within this context. However, these can be overcome through the use of DAML+OIL extensions to RDF Schema including multiple range constraints, boolean combinations of classes and class-specific constraints on properties. In addition, the lack of cardinality and datatyping constraints in RDF Schema can be overcome by maintaining the XML Schema definitions and linking them to the RDF Schema semantic definitions.

Whilst generating the RDF Schema representation of a subset of MPEG-7, we have also been able to determine certain repetitive patterns and other information which can be derived from the XML Schema definitions (baseTypes, comments, annotation, textual semantic descriptions, the DOM). We believe that by exploiting this information, it may be possible to automate the generation of an RDF Schema/DAML+OIL representation of MPEG-7 from the existing XML Schema definitions.

So our future work plan is to attempt to develop programmatic tools capable of automatically processing an MPEG-7 XML Schema document and converting this to a DAML+OIL ontology which correctly represents the semantics of MPEG-7 description schemes and descriptors and which is compatible and consistent with the corresponding XML Schema. Links to this ontology can then be added to the MPEG-7 XML Schema definitions.

Once the MPEG-7 ontology is complete, we will then investigate ways of merging this with the ABC/MetaNet ontology [28] as well as other metadata ontologies from other domains (rights management, museums (CIDOC CRM)), to enable a common understanding of descriptive terms across domains and the sharing and exchange of multimedia content over the semantic web.

Acknowledgements

The work described in this paper has been carried out as part of the Harmony Project. It has been funded by the Cooperative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science and Resources).

References

[1] J. Martinez, "Overview of the MPEG-7 Standard (version 5.0)", ISO/IEC JTC1/SC29/WG11 N4031, Singapore, March 2001. <http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm>

[2] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/xmlschema-0>

[3] XML Schema Part 1: Structures, W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/xmlschema-1/">

[4] XML Schema Part 2: Datatypes, W3C Recommendation, 2 May 2001, <http://www.w3.org/TR/xmlschema-2/>

[5] RDF Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000. <http://www.w3.org/TR/rdf-schema/>

[6] TV-Anytime Forum, <http://www.tv-anytime.org/>

[7] MPEG-21 Multimedia Framework, <http://www.cselt.it/mpeg/public/mpeg-21_pdtr.zip>

[8] NewsML <http://www.newsml.org/>

[9] ISO/IEC 15938-2 FCD Information Technology - Multimedia Content Description Interface - Part 2: Description Definition Language, March 2001, Singapore

[10] Dublin Core Metadata Element Set, Version 1.1, 2 July, 1999. <http://www.purl.org/dc/documents/rec-dces-19990702.htm>

[11] G. Rust, M. Bide, "The indecs Metadata Schema Building Blocks", Indecs Metadata Model, November, 1999. <http://www.indecs.org/results/model.htm>

[12] Content Standard for Digital Geospatial Metadata (CSDGM), <http://www.fgdc.gov/metadata/contstan.html>

[13] The Gateway to Educational Materials <http://www.the gateway.org>

[14] IEEE Learning Technology Standards Committee's Learning Object Meta-data Working Group. Version 3.5 Learning Object Meta-data Scheme.

[15] ICOM/CIDOC Documentation Standards Group, Revised Definition of the CIDOC Conceptual Reference Model, September 1999. <http://www.geneva-city.ch:80/musinfo/cidoc/oomodel>

[16] RDF Model and Syntax Specification, W3C Recommendation 22 February 1999. <http://www.w3.org/TR/REC-rdf-syntax/>

[17] DAML+OIL Revised Language Specification, March 2001. <http://www.daml.org/2001/03/daml+oil-index>

[18] UML Resource Center, <http://www.rational.com/uml/index.jsp>

[19] ISO/IEC 15938-5 FCD Information Technology - Multimedia Content Description Interface - Part 5: Multimedia Description Schemes, March 2001, Singapore

[20] ISO/IEC 15938-3 FCD Information Technology - Multimedia Content Description Interface - Part 3: Visual, March 2001, Singapore

[21] ISO/IEC 15938-4 FCD Information Technology - Multimedia Content Description Interface - Part 4: Audio, March 2001, Singapore

[22] J. Hunter, C.Lagoze, "Combining RDF and XML Schemas to Enhance Metadata Interoperability Between Application Profiles", WWW10, HongKong, May 2001. <http://www10.org/cdrom/papers/572/index.html>

[23] J. Hunter, L.Armstrong, "A Comparison of Schemas for Video Metadata Representation", WWW8, Toronto, May 1999 <http://archive.dstc.edu.au/RDU/staff/jane-hunter/www8/paper.html>

[24] C.Lagoze, J. Hunter, D. Brickley, "An Event-Aware Model for Metadata Interoperability", ECDL 2000, Lisbon, September 2000.

[25] XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999, <http://www.w3.org/TR/xslt.html>

[26] J. Hunter, "MetaNet – A Metadata Term Thesaurus to Enable Semantic Interoperability Between Metadata Domains", Journal of Digital Information, Volume 1, Issue 8, April 2001 <http://jodi.ecs.soton.ac.uk/Articles/v01/i08/Hunter/>

[27] J. van Ossenbruggen et al., "Towards Second and Third Generation Web-Based Multimedia", WWW10, HongKong, May 2001

[28] C. Lagoze, J. Hunter, "The ABC Ontology and Model", <http://metadata.net/harmony/dc_paper.pdf>

[29] The Harmony International Digital Library Project, <http://metadata.net/harmony/>

Appendix A: An MPEG-7 Ontology Expressed as a DAML+OIL Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
         xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
         xmlns:mpeg7="http://www.mpeg7.org/2001/MPEG-7_Schema#"
         xmlns="http://www.mpeg7.org/2001/MPEG-7_Schema#">

<rdfs:Class rdf:ID="MultimediaContent">
   <rdfs:label>MultimediaContent</rdfs:label>
   <rdfs:comment>The class of multimedia data</rdfs:comment>
```

```
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Image">
    <rdfs:label>Image</rdfs:label>
    <rdfs:comment>The class of images</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Video">
    <rdfs:label>Video</rdfs:label>
    <rdfs:comment>The class of videos</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Audio">
    <rdfs:label>Audio</rdfs:label>
    <rdfs:comment>The class of audio resources</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="AudioVisual">
    <rdfs:label>AudioVisual</rdfs:label>
    <rdfs:comment>The class of audiovisual resources</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Multimedia">
    <rdfs:label>Multimedia</rdfs:label>
    <rdfs:comment>The class of multimedia resources</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
  </rdfs:Class>
<rdfs:Class rdf:ID="Segment">
    <rdfs:label>Segment</rdfs:label>
    <rdfs:comment>The class of  fragments of multimedia content</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="StillRegion">
    <rdfs:label>StillRegion</rdfs:label>
    <rdfs:comment>2D spatial regions of an image or video frame</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Image"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="ImageText">
    <rdfs:label>ImageText</rdfs:label>
    <rdfs:comment>Spatial regions of an image or video frame that correspond to text or
captions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
 </rdfs:Class>
<rdfs:Class rdf:ID="Mosaic">
    <rdfs:label>Mosaic</rdfs:label>
    <rdfs:comment>Mosaic or panaoramic view of a video segment</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="StillRegion3D">
    <rdfs:label>StillRegion3D</rdfs:label>
    <rdfs:comment>3D spatial regions of a 3D image</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Image"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="VideoSegment">
    <rdfs:label>VideoSegment</rdfs:label>
```

```
    <rdfs:comment>Temporal intervals or segments of video data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Video"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="MovingRegion">
    <rdfs:label>MovingRegion</rdfs:label>
    <rdfs:comment>2D spatio-temporal regions of video data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="VideoText">
    <rdfs:label>VideoText</rdfs:label>
    <rdfs:comment>Spatio-temporal regions of video data that correspond to text or captions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MovingRegion"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="AudioSegment">
    <rdfs:label>AudioSegment</rdfs:label>
    <rdfs:comment>Temporal intervals or segments of audio data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Audio"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="AudioVisualSegment">
    <rdfs:label>AudioVisualSegment</rdfs:label>
    <rdfs:comment>Temporal intervals or segments of audiovisual data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#AudioVisual"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="AudioVisualRegion">
    <rdfs:label>AudioVisualRegion</rdfs:label>
    <rdfs:comment>Arbitrary spatio-temporal segments of AV data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="MultimediaSegment">
    <rdfs:label>MultimediaSegment</rdfs:label>
    <rdfs:comment>Segment of a composite multimedia presentation</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Multimedia"/>
    <rdfs:subClassOf rdf:resource="#Segment"/>
  </rdfs:Class>
 <rdfs:Class rdf:ID="EditedVideoSegment">
    <rdfs:label>EditedVideoSegment</rdfs:label>
    <rdfs:comment>Video segment that results from editing work</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#VideoSegment"/>
 </rdfs:Class>
 <rdf:Property rdf:ID="decomposition">
      <rdfs:label>decomposition of a segment</rdfs:label>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
  </rdf:Property>
  <rdf:Property rdf:ID="spatial_decomposition">
      <rdfs:label>spatial decomposition of a segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#decomposition"/>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
  </rdf:Property>
  <rdf:Property rdf:ID="temporal_decomposition">
      <rdfs:label>temporal decomposition of a segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#decomposition"/>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
```

```
      <rdfs:range rdf:resource="#Segment"/>
 </rdf:Property>
 <rdf:Property rdf:ID="spatio-temporal_decomposition">
      <rdfs:label>spatio-temporal decomposition of a segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#decomposition"/>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
 </rdf:Property>
 <rdf:Property rdf:ID="mediaSource_decomposition">
      <rdfs:label>media source decomposition of a segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#decomposition"/>
      <rdfs:domain rdf:resource="#MultimediaContent"/>
      <rdfs:range rdf:resource="#Segment"/>
</rdf:Property>
<rdf:Property rdf:ID="videoSegment_spatial_decomposition">
      <rdfs:label>spatial decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#spatial_decomposition"/>
      <rdfs:domain rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#MovingRegion"/>
 </rdf:Property>
<rdfs:Class rdf:ID="VideoSegmentsOrStillRegions">
   <daml:unionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#VideoSegment"/>
      <rdfs:Class rdf:about="#StillRegion"/>
   </daml:unionOf>
</rdfs:Class>
 <rdf:Property rdf:ID="videoSegment_temporal_decomposition">
      <rdfs:label>temporal decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#temporal_decomposition"/>
      <rdfs:domain rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#VideoSegmentsOrStillRegions"/>
</rdf:Property>

<rdfs:Class rdf:ID="MovingOrStillRegions">
   <daml:unionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#MovingRegion"/>
      <rdfs:Class rdf:about="#StillRegion"/>
   </daml:unionOf>
</rdfs:Class>
 <rdf:Property rdf:ID="videoSegment_spatio-temporal_decomposition">
      <rdfs:label>spatio-temporal decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#spatio-temporal_decomposition"/>
      <rdfs:domain rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#MovingOrStillRegions"/>
</rdf:Property>
 <rdf:Property rdf:ID="videoSegment_mediaSource_decomposition">
      <rdfs:label>media source decomposition of a video segment</rdfs:label>
      <rdfs:subPropertyOf rdf:resource="#mediaSource_decomposition"/>
      <rdfs:domain rdf:resource="#VideoSegment"/>
      <rdfs:range rdf:resource="#VideoSegment"/>
</rdf:Property>
<rdfs:Class rdf:ID="Agent">
     <rdfs:label>Agent</rdfs:label>
     <rdfs:comment>Agent - person, organisation or group which performs
 an act.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
```

```
 <rdfs:Class rdf:ID="Person">
     <rdfs:label>Person</rdfs:label>
     <rdfs:comment>An individual person.</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Agent"/>
  </rdfs:Class>
 <rdfs:Class rdf:ID="PersonGroup">
     <rdfs:label>PersonGroup</rdfs:label>
     <rdfs:comment>A group of persons with a collective title.</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Agent"/>
  </rdfs:Class>
 <rdfs:Class rdf:ID="Organisation">
     <rdfs:label>Organisation</rdfs:label>
     <rdfs:comment>Organisation.</rdfs:comment>
     <rdfs:subClassOf rdf:resource="#Agent"/>
  </rdfs:Class>
<rdf:Property rdf:ID="role">
      <rdfs:label>The Role played by an agent or place in an event</rdfs:label>
      <rdfs:domain rdf:resource="#Agent"/>
      <rdfs:domain rdf:resource="#Place"/>
</rdf:Property>
 <rdfs:Class rdf:ID="Place">
     <rdfs:label>Place</rdfs:label>
     <rdfs:comment>Describes real, fictional, historical locations.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Time">
     <rdfs:label>Time</rdfs:label>
     <rdfs:comment>Describes date/time points and durations</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="Instrument">
    <rdfs:label xml:lang="en">Instrument</rdfs:label>
     <rdfs:comment>Describes instrument or tool used to perform an action.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>
<rdf:Property rdf:ID="name">
      <rdfs:label>name</rdfs:label>
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#PersonName"/>
 </rdf:Property>
<rdfs:Class rdf:ID="Affiliation">
  <rdfs:comment>An affiliation is either an Organisation or a PersonGroup </rdfs:comment>
  <daml:unionOf rdf:parseType="daml:collection">
    <rdfs:Class rdf:about="#Organisation"/>
    <rdfs:Class rdf:about="#PersonGroup"/>
  </daml:unionOf>
</rdfs:Class>
 <rdf:Property rdf:ID="affiliation">
      <rdfs:label>affiliation</rdfs:label>
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Affiliation"/>
</rdf:Property>
 <rdf:Property rdf:ID="address">
      <rdfs:label>address</rdfs:label>
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Address"/>
 </rdf:Property>
```

```
<rdfs:Class rdf:ID="Address">
    <rdfs:label>Address</rdfs:label>
    <rdfs:comment>Address of person, organisation or person group.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Place"/>
 </rdfs:Class>
<rdfs:Class rdf:ID="PersonName">
    <rdfs:label>PersonName</rdfs:label>
    <rdfs:comment>Name of an individual person.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
<rdf:Property rdf:ID="givenName">
     <rdfs:label>givenName</rdfs:label>
     <rdfs:domain rdf:resource="#PersonName"/>
     <rdfs:range rdf:resource="#Literal"/>
</rdf:Property>
<rdf:Property rdf:ID="familyName">
     <rdfs:label>familyName</rdfs:label>
     <rdfs:domain rdf:resource="#PersonName"/>
     <rdfs:range rdf:resource="#Literal"/>
</rdf:Property>
<rdfs:Class rdf:ID="Creation">
    <rdfs:label>Creation</rdfs:label>
    <rdfs:comment>A multimedia content creation.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MultimediaContent"/>
 </rdfs:Class>
<rdf:Property rdf:ID="title">
     <rdfs:label>title</rdfs:label>
     <rdfs:subPropertyOf rdf:resource="#multimediaDescriptor"/>
     <rdfs:domain rdf:resource="#Creation"/>
     <rdfs:range rdf:resource="#Title"/>
</rdf:Property>
<rdf:Property rdf:ID="abstract">
     <rdfs:label>abstract</rdfs:label>
     <rdfs:subPropertyOf rdf:resource="#multimediaDescriptor"/>
     <rdfs:domain rdf:resource="#Creation"/>
     <rdfs:range rdf:resource="#TextAnnotation"/>
</rdf:Property>
<rdf:Property rdf:ID="creator">
     <rdfs:label>creator</rdfs:label>
     <rdfs:subPropertyOf rdf:resource="#multimediaDescriptor"/>
     <rdfs:domain rdf:resource="#Creation"/>
     <rdfs:range rdf:resource="#Creator"/>
</rdf:Property>
<rdf:Property rdf:ID="creationLocation">
     <rdfs:label>creationLocation</rdfs:label>
     <rdfs:subPropertyOf rdf:resource="#multimediaDescriptor"/>
     <rdfs:domain rdf:resource="#Creation"/>
     <rdfs:range rdf:resource="#Place"/>
</rdf:Property>
<rdf:Property rdf:ID="creationDate">
     <rdfs:label>creationDate</rdfs:label>
     <rdfs:subPropertyOf rdf:resource="#multimediaDescriptor"/>
     <rdfs:domain rdf:resource="#Creation"/>
     <rdfs:range rdf:resource="#Time"/>
</rdf:Property>
<rdfs:Class rdf:ID="Creator">
    <rdfs:label>Creator</rdfs:label>
```

```
    <rdfs:comment>Person, organisation or person group who created the content.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Agent"/>
 </rdfs:Class>
 <rdf:Property rdf:ID="role">
     <rdfs:label>role</rdfs:label>
     <rdfs:domain rdf:resource="#Creator"/>
     <rdfs:range rdf:resource="#ControlledTerm"/>
 </rdf:Property>
 <rdf:Property rdf:ID="creationTool">
     <rdfs:label>instrument</rdfs:label>
    <rdfs:comment>Instrument used by creator to create multimedia content.</rdfs:comment>
     <rdfs:domain rdf:resource="#Creator"/>
     <rdfs:range rdf:resource="#Instrument"/>
 </rdf:Property>
<rdfs:Class rdf:ID="Color">
    <rdfs:label>Color</rdfs:label>
    <rdfs:comment>Color of a visual resource</rdfs:comment>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="DominantColor">
    <rdfs:label>DominantColor</rdfs:label>
    <rdfs:comment>The set of dominant colors in an arbitrarily-shaped region.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Color"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="ScalableColor">
    <rdfs:label>ScalableColor</rdfs:label>
    <rdfs:comment>Color histogram in the HSV color space.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Color"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="ColorLayout">
    <rdfs:label>ColorLayout</rdfs:label>
    <rdfs:comment>Spatial distribution of colors.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Color"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="ColorStructure">
    <rdfs:label>ColorStructure</rdfs:label>
    <rdfs:comment>Describes color content and the structure of this content.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Color"/>
 </rdfs:Class>
 <rdfs:Class rdf:ID="GoFGoPColor">
    <rdfs:label>GoFGoPColor</rdfs:label>
    <rdfs:comment>Group of frames/pictures color descriptor.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#ScalableColor"/>
 </rdfs:Class>

 <rdf:Property rdf:ID="color">
     <rdfs:label>color</rdfs:label>
     <rdfs:comment>Color descriptor - applicable to video segments, still regions and moving
regions.</rdfs:comment>
     <rdfs:subPropertyOf rdf:resource="#visualDescriptor"/>
     <rdfs:domain rdf:resource="#VideoSegment"/>
     <rdfs:domain rdf:resource="#StillRegion"/>
     <rdfs:domain rdf:resource="#MovingRegion"/>
     <rdfs:range rdf:resource="#Color"/>
 </rdf:Property>
</rdf:RDF>
```

Overcoming Ontology Mismatches in Transactions with Self-Describing Service Agents

Drew McDermott
Yale University
drew.mcdermott@yale.edu

Mark Burstein
BBN Technologies
burstein@bbn.com

Douglas Smith
Kestrel Institute
smith@kestrel.edu

Abstract. One vision of the "Semantic Web" of the future is that software agents will interact with each other using formal metadata that reveal their interfaces. We examine one plausible paradigm, where agents provide *service descriptions* that tell how they can be used to accomplish other agents' *goals*. From the point of view of these other agents, the problem of deciphering a service description is quite similar to the standard AI planning problem, with some interesting twists. Two such twists are the possibility of having to reconcile contradictory *ontologies* — or conceptual frameworks — used by the agent, and having to rearrange the data structures of a message-sending agent so they match the expectations of the recipient. We argue that the former problem requires human intervention and maintenance, but that the latter can be fully automated.

1 Introduction

Suppose an agent is given the task of buying the paperback edition of "Robo Sapiens" for less than $25.

The agent must carry out several tasks:

1. Find other agents that might be able to help carry out the given action. (A *broker agent* would perform this part.)
2. For each such agent, get a description of what service it provides. This description must be expressed in a formal language, such as DAML (DARPA Agent Markup Language).

3. If the goal description and the service description do not use the same ontology, find a common framework to translate them to. An *ontology* is a "conceptual scheme," a way of talking about the world.[1]

4. Find and execute a *plan* for satisfying its goal, that is, a series of interactions with a given bookseller that result in the agent acquiring a copy of the book. The primitive actions of the plan will be actions that send and receive messages. Building and decoding these messages may require further translation, between what one agent wants to receive and what the other knows.

One of the key questions we address in this paper is how agents' goals and servers' service description can be represented, and what is necessary to make the two mesh. Many treatments of such problems assume that representations can be as simple as lists of keywords and values

```
(''Task: buy; Thing-to-buy: book; Price: (< $25); ... .'')
```

Such notations work fine as long as all tasks fit within a preimagined framework, but are unable to express anything novel.

We prefer to use notations that respect the degrees of freedom we're likely to require in the future. It seems inescapable that such notations will have the power of formal logic:

```
(do-for-some (λ(m - Merchant b - Book)
               (and (= (title b) "Robo Sapiens")
                    (sells m b)
                    (< (price m b) (* 25 $))))
             (λ(m - Merchant b - Book)
               (buy-from m 1 b)))
```

`(do-for-some` p a`)` means, "For some object(s) x satisfying predicate p, do `(`a x`)`." We use Lisp-style notation for logical constructs. Function application is written `(`*function* arg_1 ... arg_n`)`, even if the *function* is traditionally written using infix notation. So `(* (+ 3 4) 5)` is the Lisp way to write `(3+4)*5`.[2]

The notation `(λ (`*params*`)` e`)` denotes a function whose parameters are *params* and whose value is e. We use the term *body of* the λ-expression to refer to e. Although it's not our emphasis in this paper, all expressions must be *typable*, meaning that it must be possible to assign consistent types to all their subexpressions. When necessary for typability or perspicuity, parameters can have declared types, indicated using the notation `(λ (` ... *param* `-` *type* ... `)` ... `)`. λ-expressions have many purposes. The first λ-expression in our example is a predicate, because its body is of type `Proposition`. The second denotes a function from merchants and books to actions, so that applying it to a particular merchant and book yields a particular action, namely, buying one copy of that book from that merchant. The

[1] Original meaning: the philosophical study of being. As used in AI, the word "ontology" has come to mean "what is represented as existing."

[2] We depart from Lisp notation in two contexts. We represent finite sets using braces and tuples using angle brackets. Lisp purists may prefer to read `{a, b, c}` as `(set a b c)`, and `<a b c>` as `(tuple a b c)`.

combination of `do-for-some` and λ work together to define a "quantifier" for actions, analogous to the usual existential quantifier $\exists(x \in S)P(x)$ in mathematical logic. The action `(do-for-some` p a`)` is carried out whenever the agent does `(`a x`)` for some x satisfying p. There is no presupposition that it achieves this by, say, finding an x that satisfies p, then doing `(`a x`)`. In the present case, it might search for a plan for `(buy-from m96 1 b97)`, where `m96` and `b97` are placeholder constants labeled with the constraints that `b97` be *Robo Sapiens*, and that `m96` be a merchant that sells `b97` for less than $25. Or it might pursue it in some other way entirely; the logic doesn't care.

In this paper, we focus on the question how these logic-based representations can be used, and in particular what happens after brokers have done their work, so that two or more agents know of each other's existence and possible usefulness. At that point the task becomes getting the agents to talk to each other in order to solve a common problem. For clarity, we will adopt the following terminology: the *planning agent* is the one whose point of view we are taking, i.e., the buyer in our example; the *target agent(s)* are those the planning agent is trying to interact with. We assume the target agents are not under our control. They share some of the notational assumptions we make, but we must take their notations as we find them.

2 Using Self-Describing Agents

One of our notational assumptions is that each target agent will have a *service description* embedded in the interface it presents to the world, which one may visualize as a web page. This description will have an internal and an external form. The external form is "web-friendly," in the sense that it looks like XML, and, when appropriate, can be displayed and browsed through. Such a language is under development under the label "DARPA Agent Markup Language," or DAML (`http://www.daml.org`), which is an extension of RDF, the Resource Description Framework.

(See `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`.)

So what we have been writing as

```
(book-isbn book21 "0-262-13383-0")
```

might be encoded on the web more like this:

```
<rdf:Description about=''#book21''>
        <pub:book_isbn>0-262-13383-0</pub:book_isbn>
</rdf:Description>
```

However, these are simply two alternative syntaxes for the same thing, which is represented internally as an abstract syntactic object.

The first hurdle to overcome is that the two agents must "speak the same language." Two different booksellers (e.g., Amazon.com and Barnes & Noble) must use the same industry-specific vocabulary in their service descriptions. If they don't, then we have an *ontology translation problem*, an issue we'll address in section **??**,

making only two remarks here: (1) Within an industry there will be strong motivation to adopt a standard vocabulary, as is indeed already happening with XML; (2) the main place the translation problem will arise is when satisfying a request requires interaction of agents from multiple communities.

Assuming for now that the service description is in the same language as our service request, what we have to do is verify that there is a way of carrying out the request by talking to the target agent. (In general, we may have a collection of target agents to talk to, but we'll ignore that.)

This kind of verification is close to what AI researchers call a *planning problem*: Given a description of a system, an initial state of the system, and a goal, find a sequence of actions that achieve the goal in that system. Here the service description plays the role of system description and initial state. Once the action sequence has been found, during the *planning phase*, it must be executed. During this *plan execution* phase, the actions are executed in order. It is reasonable (we hope) to assume that the planning agent will succeed if it executes the plan; but there may well be situations where the plan exits prematurely with some sort of failure indication. In that case the agent may give up, or *replan*, starting from the situation it finds itself in halfway through the original plan.

Let's look at an example of planning and execution, involving a fictional bookseller, "Nile.com." . One thing you can do at Nile.com's web page is find out if they have a book in stock. Nowadays this is done by using the search facility, and visually inspecting the output, looking for phrases such as "In stock, usually ships within 24 hours." In an agent-oriented world, actions such as filling in a form and pushing a button will have dual descriptions in terms of agents sending messages. Similarly, outputs will be defined in terms of formal languages, as well as being displayable for human consumption.

We will formalize this by having `send` and `receive` actions:

- `(send` *agent message*`)`: Send the given message to the given agent; creates a message id that the sending agent can use to identify replies.
- `(receive` *agent message-id*`)`: Receive a message, sent in reply to the original sender's message.

The message to the bookseller is of the form `(search {<`key_1`, `val_1`>, ..., <`key_k`, `val_k`>})`.[3] The response is a list of "book descriptions," giving important information about each book that matches the search keys. These descriptions will also be in an XML dialect, but as usual we will use a more compact notation.

So the plan we are looking for might begin:

```
(series (tag s1
           (send Nile.com
                 (test-in-stock
                      <<author "Philip K. Dick">
                       <title "Ubik">>)))
        (tag r2 (receive Nile.com (value s1))))
           (test (= (value r2) empty-set)
```

[3] As before, what's actually sent is a piece of XML. This is an ongoing area of research; W3C's effort is described at `http://www.w3.org/MarkUp/Forms/`.

```
                (fail (not-in-stock ... ))
                 ... ))
```

In this plan, the `tag`s allow us to give names to steps in the plan. The `value` of a step is the result it returns. The value of `s1` is a message id that later `receive`s can refer to. The value of `r2` is the set of tuples received in answer to the `in-stock` query.

To formalize this in terms a planner can understand, we create *action definitions* such as

```
(:action (send ?a - agent ?msg - Message)
     :vars (?id - Message-id)
     :value ?id
     :effect (reply-pending ?a ?id ?msg))

(:action (receive ?a - agent ?id - Message-id)
     :vars (?msg - Message)
     :precondition (reply-pending ?a ?id ?msg)
     :effect (forall (?d - (Lst (Tup Attribute String))
                     ?sv - Message)
                (when (and (= ?msg (test-in-stock ?d))
                           (this-step-val ?sv))
                      (know-val (has-in-stock ?a ?d)
                                ?sv))))
```

This is an extension of PDDL (Planning Domain Definition Language) notation, which is in standard use in the AI planning world[**?**, **?**]. The details of the notation are not important here, but the gist is that sending a message creates a message id, so that a later reception can know what it's a response to. In addition, in the case where the message sent was an "in-stock" inquiry, one result of the action is that the planning agent knows whether the target agent has the book in stock. In other words, by executing this action the planning agent will have acquired new information.

This way of representing the effects of `receive` is too clumsy for practical use, because to be realistics the effect specification would have to list the effects of all the possible `send`s that the `receive` could be in answer to. A better idea is to have assertions of the form

`(message-exchange` *message-id*
sent-message
received-message
effect`)`

and have the `:effect` field of `:receive` consult these assertions:

```
:effect (when (and (this-step-val ?sv)
                   (message-exchange ?id ?smsg ?sv ?e))
              ?e)
```

Obviously, an AI planner can solve problems involving actions that acquire information only if it can reason about situations in which it doesn't already know everything. As it happens, many planning algorithms, including some of the most

efficient, cannot. They require it to be the case that the initial state of the world, the set of possible actions, and the effects of every action are all known. The only uncertainty is which action sequence will bring about a desired result. There has been much research on relaxing these assumptions, but no approach that is obviously correct.

Fortunately, the version of the problem we are confronted with is not as bad as the general case, because our agent knows at planning time exactly what it will and will not know at plan-execution time. In addition, we can avoid tackling extremely general formalizations of what it means for an agent to know something. For automated agents, we can appeal to the difference between computable and noncomputable terms. A term is *computable*[4] if it can be "evaluated," yielding a canonical term for an object of its type. For instance, the term `(+ 5 4)` is computable, because we can hand it to a programming-language processor and get back `9`. We will use the term *computational* for a term like `9` that is canonical in the sense alluded to, meaning that it can take part in further computations using standard algorithms. We write `(val (+ 5 4) 9)`, where `val` is a variant of equality that applies only to computable terms and their computational values. By contrast, `(number-of-planets sun)`, while it may also happen to denote nine, is not a computational representation of nine in the way the term `9` is. It is not even computable, because we cannot simply ask a Lisp system to evaluate `(number-of-planets sun)` and expect to get back `9`.[5]

A plausible principle for agents is

> *To know something is to have a computable term whose value is (a computational representation of) that something.*

We formalize this principle by introducing predicates expressing what the planning agent knows. (We currently do not provide for reasoning about what the target agents know; we believe that there is little symmetry between the two cases, because even if the planning agent believes that a target agent has a computable term denoting something, the planning agent won't know what that term is or how to evaluate it.)

One such predicate is `(know-val` e r`)`, which means that the agent knows the value of expression e, and that the value is the value of computable term r. For example, the agent might record

```
(know-val (book-isbn book21) (value step14))
```

meaning that `(val (value step14)` s`)` if and only if s is a string giving a legal ISBN (International Standard Book Number) for `book21`. Here we make use of the fact that after a plan step p has been executed, `(value` p`)` is a computable term.

We will also require a predicate `(know-val-is` e r v`)`, which is roughly equivalent to

[4] We adopt this term with some hesitation, because its usual meaning has somewhat different connotations. However, we can't think of a better one.

[5] Of course, there may be programs, say, a front end to a database of astronomical facts, in which one can do exactly this; in that context the term `(number-of-planets sun)` *would* be computable.

`(and (know-val` e r`) (val` r v`)))`

except that the planner will avoid trying to make such a goal true by changing the value of r.

A computational term representing a finite set is the familiar $\{r_1, \ldots, r_k\}$, where r_i is a computational term representing the i'th element of the set. Sometimes it is sufficient for an agent to have a partial listing of a set. To represent that situation, we have two further predicates

- `(known-elements` S r`)`: Meaning that r is a computable term whose value is a computational representation of the set of all objects the planning agents knows to be elements of S.
- `(known-elements-are` S r $\{r_1, \ldots, r_k\}$`)`: In which the elements are spelled out.

2.1 Proposed Planning Algorithm

Most previous work in the area of planning with incomplete knowledge—so-called *contingent planning*—has been done in the context of partial-order planning [**?**, **?**, **?**]. This fact is mainly a historical accident, because work on planning with incomplete knowledge happened to coincide with a period when partial-order planning was popular.

We are adding a contingent-planning ability to our Unpop planner [**?**], which is in the family of *estimated-regression search* planners [**?**]. These systems build a plan by starting with a null series of actions and adding actions to its trailing end. At each stage, it attempts to add the action that will maximally reduce the estimated effort required to finish the plan. The effort is estimated by constructing a tree of subgoals that relates the original goal to the current situation. The branches of the tree are simplified versions of the actual sequence of actions that will be required to solve the problem. The tree, called the *regression graph*, can be computed efficiently, but is only a heuristic estimate of the actual actions required, because many interactions between actions are ignored.

To handle contingent planning, Unpop must be modified thus:

1. The output of the planner can't be a simple sequence of actions; it must include `if-then-else` tests that send execution in different directions based on information gathered.
2. As a consequence, the space searched by the planner cannot be a simple space of action sequences. One alternative would be to let the space be the set of "action trees," each branch of which corresponds to a sequence. However, this idea has a couple of bugs that we will discuss below.
3. Given a goal of the form `(know-val` e `...)`, the planner must either verify that the planning agent already knows e, or find an action whose `value` can be be used to construct e. For a goal of the form `(send` a m`)`, the planner must verify that it knows, or can come to know, sufficient information to build m

To deal with this last issue, the planner must index actions by the values they compute, in much the way that planners traditionally index them by the effects they

can bring about. However, there are some differences. There will seldom be an exact alignment between what the planning agent knows and what it needs to know. For instance, if the value of an action `(ask-name ?c)` is `<(last-name ?c) (first-name ?c)>`, and the planning agent wants to know the last-name of `D`, it will have a goal `(know-val (last-name D) ?r)`. The term `(last-name D)` can be extracted from the action value by using the function `(elt` t i`)`, which gets the i'th element of a tuple t. So all the planner has to do is propose the action `(ask-name D)`, which will have, among other things, the effect `(know-val (last-name D) (elt (value` S`) 1))`, assuming S is the step with action `(ask-name D)`. A bit of care must be taken here to ensure that S is a placeholder for the correct step, which of course doesn't exist yet. We discuss this issue at greater length in section 3.1.

Let's look more closely at the search-space issue. As we said above, the most straightforward idea is to think of a partially constructed plan as a tree of actions, with the branch points occurring after information-gathering steps. A plan is completed successfully when every branch leads to a successful conclusion. One bug with this idea is that it may be asking too much to require every branch of a plan to succeed. Often there is a "normal" result of an information-gathering step, such that it is reasonable to expect the normal result to occur. If it might not occur, the best thing for the planner to do is tack on a short branch saying "Give up!" The resulting branching plan has one branch that succeeds and one that fails, which is perfectly all right. If Nile.com might not have your book, that is no reason to give up on the attempt to deal with them. Hence rather than require all branches to succeed, we require just one to succeed, hopefully the most likely one.

Another problem has to do with efficiency. Suppose a plan has a branch point fairly early, leading to subplans P_1 and P_2. In general, the planner has to do a search through different partial versions of P_1 and P_2. Suppose it eventually finds versions $P_{11}, P_{12}, \ldots, P_{1m}$ of P_1, and versions $P_{21}, \ldots\ P_{2n}$ of P_2. Using the tree representation, we must represent these as mn distinct trees. The numbers m and n might be around 50 in a realistic case, so we have 2500 different plans to think about. Worse, the computation the planner does for, say, $P_{1,23}$ is the same regardless of whether it is paired with $P_{2,13}$ or $P_{2,32}$, so the planner will have to do the same work over and over.

The best search space therefore turns out to be the one we started with: a set of sequences of steps, each representing a partial plan. The only difference is that each sequence may be annotated with zero or more *knowledge notes* recording what the planner will have learned at various points in the sequence. There is also a difference in what the planner must do when a complete sequence is found. It now may discard all the competing plans that reflect the same knowledge notes, and keep working on plans that represent other knowledge states. For instance, the planner may find a plan for buying a book assuming that there is a paperback edition. Having found it, it may continue to look for a plan to handle the case where it discovers that there is no paperback edition.

When the planner runs out of patience, it returns however many branches it can cobble together. If during execution it diverges from the branches it predicted would succeed, it must replan. In some such cases, the new information it has will allow it to find a good plan; but many times the problem will just not have a solution.

2.2 Scripts and Hierarchical Planning

So far, we seem to be assuming that service descriptions contain specifications of the effects of individual atomic transactions with the server. These are indeed important, but in practice many servers will also provide "scripts," that is, standard sequences of transactions that can be used to accomplish common goals.

For instance, a bookseller might provide a script for the action `(buy-from` m n d`)`, meaning "Buy n copies of something answering to the description d." (To keep things simple, we suppress the price argument we used earlier.) That script might look like:

```
(:method buy-from
    :params (?m - Merchant
             ?quant - Integer
             ?d - Item-description)
    :vars (?r - (Set ISBN) ?isbn - ISBN)
    :precondition
        (and (forall (x) (if (?d x) (is Book x)))
             (know-val-is (image book-isbn
                                  (set-of-all ?d))
                          ?r {?isbn}))
    :expansion
        (series (send ?m
                      (verify-in-stock ?isbn))
                ... ))
```

The notation `(set-of-all ?d)` is the set of all objects matching description `?d`. In traditional notation that would be written $\{x|$`(?d ?x)`$\}$. The function

`(image` f l`)`

creates a list with elements `(`f l_0`)`, `(`f l_1`)`, ..., `(`f l_{n-1}`)`, so `(image book-isbn ... )` changes a list of books into a list of their ISBNs, a computational object.

The idea behind scripts is that if the planning agent just wants to carry out the action `(buy-from ... )`, or any action that fits one of the scripts, it can save searching for a plan by just finding and tuning the appropriate script. (Tuning might include filling in actions to achieve goals for which the script supplies no action.)

This style of planning is usually called *hierarchical*, because the problem is to instantiate hierarchies of actions using large building blocks rather than assembling sequences of individual actions. Hierarchical planning is fairly well understood, and tends to be efficient when it is applicable at all (because the script writer has done most of the work already). There is an interesting research question here about how to get a planner to do both hierarchical and sequential planning. Our approach will be to augment the notion of partial plan to include partially expanded scripts as well as open goals, but the focus of this paper is on agent-communication issues, so we won't go into this any further. However, we do point out that the goal we started with, `(do-for-some ... (λ( ... ) (buy-from m 1 b)))`, is actually an *action* rather than a *propositional goal*, so we've been assuming that actions are part of problem specifications all along.

3 Ontology and Data Structure Translation

It's time we turned to our principal topic, which is how to cope with ontology and data-structure mismatch. We begin with the latter.

3.1 Glue Code

Assuming that the planning agent and target agent use the same ontology, there is still a potential mismatch problem. Suppose that the planning agent is dealing with a book seller that offers a discount if you order 10 or more books, not counting bulk orders. Somewhat artificially, let's suppose that the planning agent is responsible for sending the total at some point. That is, the planner contemplates executing the action:

```
(send G (non-bulk-total
           (size (set-of-all
                     (λ (b)
                       (intention (buy-from G 1 b)))))))
```

This looks complex, so let's break it down into parts.

```
(set-of-all (λ (b) (intention (buy-from G 1 b))))
```

is the set of all books `b` such that the planning agent intends to buy exactly 1 copy of `b` from `G`. The function `non-bulk-total` is a constructor that builds a message to send to the target agent — a computational object.

Obviously, the planning agent should know what it plans to buy. Using the principle of section 2, that means it must have a computable term for it. Suppose the following is true in the initial situation:

```
(λ    (know-val (image (b k)
                   (set-of-all
                       (λ (b k)
                         (intention
                             (buy-from G k b)))))
          pending-orders)
```

This formula states that the computable variable `pending-orders` contains (by stipulation) a set of triples <*author title quantity*> for every book the planning agent intends to buy some quantity of. Let's explain that more gradually. The `set-of-all` expression here is similar to the one we need to send, except that it denotes a set of tuples <b k> for every book b that the agent plans to buy k copies of. These tuples are not computational, but we can convert it to something that is by using `image`. While a book or an author is an abstract object in a universe of discourse, the name of the book or author is just a string, and the number of copies the agent intends to buy is represented as a sequence of binary digits. Furthermore, the use of `know-val` announces that the variable stored in `pending-orders` is computable, and its value will be a purely computational object, namely an ordinary tuple holding two strings and an integer.

The message the agent needs to send, and the data it has in its possession, are tantalizingly closely related, but not identical. We need a procedure that translates

from what the agent knows to what it needs to send. We call such a procedure *glue code*, because it connects two things together. In [**?**] we discussed how to generate glue code automatically; the same approach will work in this context, with some minor modifications to the assumptions we make about the source side. In the original paper we assumed that the things the agent "knows" are strung together in a tuple; now we posit that these entities are the values of an unordered collection of computable terms, of which only a subset may be relevant to building a particular data structure.

Space does not permit us to explain in detail how the algorithm works. We treat the glue-code-generation problem as finding a computable function f such that

`(`f *"things agent knows"*`) = "things agent needs"`

The right-hand side is called the *target*, the arguments to f are called the *source*. The algorithm operates by transforming the target until it contains only terms that appear in the source, in which case f can be produced by λ-abstraction (replacing terms with variables).

The output of the algorithm in our example should be

```
(non-bulk-total
    (size (filter (λ (b k) (= k 1))
                  pending-orders)))
```

The value of

`(filter` p l`)`

is a copy of list l containing just the elements satisfying predicate p. In this context, it means that we discard from `pending-orders` all the tuples corresponding to bulk orders.

The planning context adds another dimension to the problem of glue-code generation. In addition to the computable terms that the planner knows about, it must also entertain the possibility of generating new computable terms of the form `(value` *step*`)`, where *step* is a new step added to the plan. The open research question is how to fit this into the computation of the regression graph.

3.2 Ontology Translation

We now turn to the most difficult problem that web-based agents must cope with, the problem of reconciling disparate ontologies, or representational frameworks. The reason it is so difficult is that it often requires subtle judgments about the relationships between the meanings of formulas in one notation and the meanings of formulas in another. Furthermore, there is no obvious "oracle" that will make these judgments. For instance, we cannot assume that there is an overarching (possibly "global") ontology that serves as a court of appeals for semantic judgments. There are times when such a strategy will work, but only after someone has provided a translation from each of the disparate ontologies to the overarching framework, and there is no reason to expect either of these translation tasks to be any easier than the one we started with. Indeed, the more the overarching framework encompasses, the harder it will be to relate local ontologies to it. Hence the work of ontology reconciliation inevitably involves a human being to do the heavy lifting. The most we can

hope for is to provide a formal definition of the problem, and software tools[6] to aid in solving it.

The goal of these tools is to develop and maintain *ontology transformations*. An ontology transformation is a mechanism for translating a set of facts expressed in one ontology (O_1) into a set of expressions in another ontology (O_2), such that the new set "says the same thing" as the original set.

Ontology translation is partly a matter of syntax and partly a matter of logic. The logical issues include:

- *Vocabulary:* What symbols does the ontology use and what do they refer to?
- *Expressiveness:* What logical constructs are allowed?

The expressiveness issue may not sound ontological, but it can be. For instance, if the ontology allows us to talk about possible truth, it may commit us to assuming the existence of possible but nonactual worlds in which propositions false in this world are true.

In addition to such purely logical issues, computational questions about how facts are structured and accessed are often mixed into the ontology question. Examples:

- *Implicit content:* What facts are represented implicitly in a given formalism? For instance, if the formalism allows a list of objects at a certain point, does it imply that the list comprises all the objects with a certain property?
- *Indexing:* How are facts associated with "keys" so that they can be retrieved when necessary? Specifically, is every fact associated with a class of object it is true of?
- *Efficiency:* Is the language restricted in such a way as to make some class of inferences more efficient?

Past work in the area of ontology transformation [**?**, **?**] has addressed both logical and computational issues. We think it is more enlightening to separate them out. From the point of view of logic, computational issues affect mainly the *concrete syntax* of an ontology. Therefore it ought to be possible to find an abstract version O_a of any ontology O, such that any set of facts expressed in O can be translated into a set of facts in O_a. Furthermore, all abstract ontologies use the *same* syntax, so that there is no longer any need to mix syntactic and computational issues into logical ones. In other words, we assume that an ontology transformation $O_1 \rightarrow O_2$ can always be factored into three transformations $O_1 \rightarrow O_{1a} \rightarrow O_{2a} \rightarrow O_2$. This may not seem like an improvement at first, but it has some advantages. First, it allows us to focus on abstract$\rightarrow$abstract transformations, and put syntax on the back burner. Second, the translation $O \rightarrow O_a$ should not be very difficult, because it is essentially a matter of "parsing" a set of facts; going in the other direction, $O_a \rightarrow O$ is a matter of "generating" the concrete representation of a set of facts. Third, the transformation $O \leftrightarrow O_a$ has to be done just once for each ontology.

One might object that not all the content of a set of facts can be pulled out and made into explicit formulas, and therefore that our decomposition, however tidy, will not work in practice. We take this objection seriously, but for now our principal

[6] Such as those described by [**?**].

reply is that for ontologies in which it is valid the transformation problem is not very well defined no matter what approach you take to it.

Hence we will continue to employ our tactic of focusing on abstract rather than concrete data structures. We will assume that all facts are expressed in terms of *formal theories*, each of which we take to contain the following elements:

1. A set of *types*.
2. A set of *symbols*, each with a type.
3. A set of *axioms* involving the symbols.

In addition we introduce the concept of a *dataset*, that is, a set of facts expressed using a particular ontology. This concept abstracts away from the actual representations of, say, Nile.com's current inventory, and treats it as a set of identifiers and facts about them, which uses symbols from that ontology.

Once we have cleared away the syntactic underbrush, the ontology-transformation problem becomes much clearer. One is likely, in fact, to see it as trivial. Suppose one bookseller has a theory O_1 with a predicate `(in-stock x - Book t - Duration)`, meaning that `x` is in stock and may be shipped in time `t`. Another bookseller expresses the same information in its theory O_2, with two predicates, `(in-stock y - Book)` and `(deliverable d - Duration y - Book)`. We are presented with a dataset D_1 that is in terms of O_1, which contains fragments such as

```
(:constants ubik blade-runner - Book)
(:axioms (in-stock ubik (* 24 hr))
         (in-stock blade-runner (* 4 day))
          ... )
```

To translate this into an equivalent dataset that uses O_2, we must at least find a translation for the axioms. The types and constants need to be handled as well, but we'll set that aside for a moment. We will use the notation $D_1 \rightarrow D_2$ as a mnemonic for this sort of transformation problem.

With this narrow focus, it becomes almost obvious how to proceed: Treat the problem as a deduction from the terms of one theory to the terms of the other. That is, combine the two theories by "brute force," tagging every symbol with a subscript indicating which theory it comes from. Then all we need to do is supply a "bridging axiom" such as

```
(forall (b t) (iff (in-stock₁ b t)
                   (and (in-stock₂ b)
                        (deliverable₂ t b))))
```

which we can use to translate every axiom in D_1. More precisely, we can use it to augment the contents of D_2. Any time we need an instance of `(in-stock`$_2$ `x)` and `(deliverable`$_2$ `y x)`, the bridging axiom will tell us that `(in-stock`$_2$ `ubik)` and `(deliverable`$_2$ `(* 24 hr) ubik)` are true (and maybe other propositions as well). We then discard the subscripts, and we're done. Furthermore, elementary type analysis tells us that `ubik` is of type `Book`$_2$.

This idea is similar to the *lifting axioms* of [**?**]. The main difference is that they focused on axioms of the form `(if (`*axiom in one domain) axiom in another)*,

whereas we use `iff`. The reason for the difference is that we are interested in inferring facts of the form `(not (in-stock`$_2$ `x))`; we could avoid this sort of inference if we could rely on a closed-world assumption for the predicate `in-stock`.

Of course, the deductive approach does not solve all problems. Here is a list of some of the remaining issues:

1. It is potentially reckless to reduce ontology transformation to theorem proving. In the example, the required deduction was easy, but in general it could be undecidable, after finding zero, one, or two axioms, whether there are any more. However, we are inclined to think that most of the theorem-proving problems that arise during ontology translation are straightforward.

2. We attached subscripts to predicates and types, but not to other identifiers. That implies that we can just take a symbol like `ubik` over to the target theory. But suppose the target dataset must be compatible with some existing O_2 dataset, and the symbol `ubik` is already in use. In principle the deductive framework can accommodate this situation, by including a test for whether `ubik`$_1$ and `ubik`$_2$ refer to the same object, i.e., whether we can prove `(= ubik`$_1$ `ubik`$_2$`)`. It is often easy to show that they are not equal, by showing that they are of different types. But suppose we can't prove either that the two identifiers are equal or that they are unequal. What do we do then? Also, do we have to test all pairs of symbols for equality? (Two symbols could easily be provably equal even though they are spelled differently.)

   We glossed over similar problems with variables and types. We wrote `(forall (x y) ... )`, implying that `x` and `y` could live in both ontologies. We may want to allow that as a special case, but in the general case it is necessary to provide transformations for the values of variables. To modify our example somewhat, suppose that the types of the arguments of `deliverable` are actually `Integer` and `Book`, so that `(deliverable 24` b`)` means that b ships within 24 hours. But let's also suppose that the symbol `Book` happens to denote exactly the same sort of thing in both domains. Then our bridging axiom might become:

```
(forall (b - Book
         t1 - Duration1 t2 - Integer)
    (if (= t1 (* t2 hr)))
        (iff (in-stock1 b1 t1)
             (and (in-stock2 b2)
                  (deliverable2 t2 b2)))))
```

   Note that equality and Integer are not domain-specific. (Put another way, there is a standard ontology where such general-purpose things live, and all other ontologies inherit from it.)

3. As has been observed before, two ontologies often carve the world up differently. They may have different "granularity," meaning that one makes finer distinctions than the other; of course, O_1 might make finer distinctions than O_2 in one respect, coarser distinctions in another.

The last issue is likely to be the most troublesome. Here's an example: Suppose O_1 is the ontology we have been drawing examples from, a standard for the mainstream book industry. Now suppose O_2 is an ontology used by the *rare* book industry. The main difference is that the rare-book people deal in individual books, each with its own provenance and special features (e.g., an autograph by the author). Hence the word "book" means different things to these two groups. For the mainstream group, a book is an abstract object, of which there are assumed to be many copies. If a customer buys a book, it is assumed that he or she doesn't care which copy is sent, provided it's in good condition. For the rare-book industry, a book is a particular object. It may be an "instance" of an abstract book, but this is not a defining fact about it.

For example, if you buy Walt Whitman's *Leaves of Grass* from Amazon.com, you can probably choose from different publishers, different durabilities (hardcover vs. paperback, page weight), different prices, and various other features (scholarly annotations, large print, spiral binding, etc.). However, you certainly can't choose exactly which copy you will receive of the book you ordered; and you probably can't choose which poems are included, even though Whitman revised the book throughout his life. The versions in print today include the last version of each poem included in any edition.

If you buy the book from RareBooks.com, then there is no such thing as an abstract book of which you wish to purchase a copy. Instead, every concrete instance of *Leaves of Grass* must be judged on its own merits. Indeed, making this purchase is hardly a job for an automated agent, although it could be useful to set up an agent to tell you when a possibly interesting copy comes into the shop.

Let's look at all this more formally. Suppose that the planning agent uses the industry-standard ontology (O_2), and the broker puts it in touch with RareBooks.com, with a note that although it bills itself as selling books, its service description uses a different ontology (O_1). If after trying more accessible sources the planning agent's goal can't be achieved, then the broker may search for an existing ontology transformation that can be used to translate RareBooks's service description from O_1 to O_2.[7]

Let us sketch what some of the bridging axioms between O_1 and O_2 might look like. In particular, we need to infer instances of $(\texttt{is Book}_2\ x)$ given various objects of type $\texttt{Book}_1$ with various properties. Objects of type $\texttt{Book}_2$ we will call *commodity books*; an example is the Pocket Books edition of *Mein Kampf*. Objects of type $\texttt{Book}_1$ we will call *collectable books*; an example is a copy of *Mein Kampf* once owned by Josef Stalin. It is roughly true that many, but not all, rare books can be thought of as instances of particular commodity books. Two rare books are instances of the same commodity book if they have the same publisher, the same title, the "same" contents, and the same characteristics (e.g., hardcover, large print, and such).[8] We can produce the following bridge axioms:

$(\texttt{:functions (book-type x - Book}_1\texttt{) - Book}_2\texttt{)}$

[7] If it can't find one, all it can do is notify the maintainers of the ontologies of the problem; there is no way for the broker, the planning agent, or the end user to find a transformation on the fly.

[8] An easy way to tell if they are the same would be to check if they have the same ISBN, but the ISBN system has been in effect for only thirty years, so it won't apply to many rare books.

```
(:axioms (forall (b1₁ b2₁ - Book₁)
             (iff (and (= (publisher₁ b1₁)
                          (publisher₁ b2₁))
                       (= (title₁ b1₁) (title₁ b2₁))
                       (= (phys-charac₁ b1₁)
                          (phys-charac₁ b2₁))
                       (< (revision-dif₁ b1₁ b2₁) 1.5))
                  (= (book-type b1₁) (book-type b2₁))))
         (forall (b₁ - Book₁)
             (= (buy₁ b₁)
                (buy₂ (book-type b₁))))))
```

This should all be self-explanatory, except for the predicate `revision-dif`, which we suppose is in use in the rare book business to express how many revisions are found between an earlier and later copy of an author's work. We have introduced a new function `book-type`, which maps individual collectable books to their types, which are commodity books.

For axioms such as these to do the planning agent any good, it must be possible for the planning agent to use them to translate a rare-book dealer's service description. Suppose the agent is trying to buy a copy of *Lady Chatterly's Other Lover*, a little-known[9] sequel to D.H. Lawrence's famous work. Having exhausted the usual sources, it attempts to deal with RareBooks.com. The planning agent first translates the service description, so that all actions are in terms of `(book-type` b`)` instead of b. Assuming it can find a way to carry out its plan, at the last stage it must translate its messages back into talking about collectable books. This requires producing glue-code in the combined axiom set. Similarly, the first step in deciphering a message from the target agent is to apply glue code to rearrange the data structures into something the planning agent can decode.

4 Conclusions

Here are the main points we have tried to make:

1. Interagent communication requires a sophisticated level of representation of knowledge states, action definitions, and plans.
2. This representation can only be logic-based; no other notation has the expressive power. Embedding this logic in some form of XML/RDF/DAML notation is a good idea for web-based agents, but puts nontrivial demands on the representational power of those notations.
3. In spite of the expressivity, there are algorithms for manipulating logic-based expressions that might overcome computational-complexity problems.
4. In particular, planning algorithms are a natural fit to the idea of a *service description*. The service description specifies the possible interactions with an agent; a plan is a sequence of interactions to achieve a specific goal. Finding such plans is more or less what planning algorithms do.

[9] in fact, fictitious

5. Planning algorithms will, however, have to be extended in various ways, in order to cope with disparities between what it knows and what the target agent wants to receive.

6. There are two key disparities that must be dealt with: ontology mismatches and data-structure mismatches. The former requires human management of a formal inter-theory inference process. The latter requires automatic generation of "glue code" to translate data structures.

This is obviously work in progress. We are in the process of adapting our Unpop planner to handle hierarchical and contingency planning, and connecting it to the glue-code generator. We are building the architecture for managing ontology transformations.

Acknowledgements: This work was supported by DARPA, the Defense Advanced Research Projects Agency. Thanks to Dejing Dou for input.

References

[1] B. Bonet, G. Loerincs, and H. Geffner. A fast and robust action selection mechanism for planning. In *Proc. AAAI-97*, 1997.

[2] M. Burstein, D. McDermott, D. Smith, and S. Westfold. Derivation of glue code for agent interoperation. In *Proc. 4th Int'l. Conf. on Autonomous Agents*, pages 277–284, 2000.

[3] H. Chalupsky. Ontomorph: A translation system for symbolic logic. In *Proc. Int'l. Con. on Principles of Knowledge Representation and Reasoning*, pages 471–482, 2000. San Francisco: Morgan Kaufmann.

[4] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *Proc. Third International Conf. on Knowledge Representation and Reasoning*, pages 115–125, 1992. Morgan Kaufmann.

[5] G. Frank, A. Farquhar, and R. Fikes. Building a large knowledge base from a structured source. *IEEE Intelligent Systems* , 14(1), 1999.

[6] T. Gruber. Ontolingua: A Translation Approach to Providing Portable Ontology Specifications. *Knowledge Acquisition* , 5(2):199–200, 1993.

[7] D. McDermott. A Heuristic Estimator for Means-ends Analysis in Planning. In *Proc. International Conference on AI Planning Systems*, pages 142–149, 1996.

[8] D. McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003).

[9] D. McDermott. The 1998 Ai Planning Systems Competition. *AI Magazine* , 21(2):35–55, 2000.

[10] P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proc. of Conf. on Extending Database Technology (EDBT 2000)*, 2000.

[11] M. Peot and D. Smith. Conditional nonlinear planning. In J. Hendler, editor, *Proceedings of the First International Conf. on AI Planning Systems*, pages 189–197. 1992.

[12] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *J. of Artificial Intelligence Research* , 4:287–339, 1996.

A Framework for Ontology Integration

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{calvanese,degiacomo,lenzerini}@dis.uniroma1.it

Abstract. One of the basic problems in the development of techniques for the semantic web is the integration of ontologies. Indeed, the web is constituted by a variety of information sources, each expressed over a certain ontology, and in order to extract information from such sources, their semantic integration and reconciliation in terms of a global ontology is required. In this paper, we address the fundamental problem of how to specify the mapping between the global ontology and the local ontologies. We argue that for capturing such mapping in an appropriate way, the notion of *query* is a crucial one, since it is very likely that a concept in one ontology corresponds to a *view* (i.e., a query) over the other ontologies. As a result query processing in ontology integration systems is strongly related to view-based query answering in data integration.

1 Introduction

One of the basic problems in the development of techniques for the semantic web is the integration of ontologies. Indeed, the web is constituted by a variety of information sources, and in order to extract information from such sources, their semantic integration and reconciliation is required. In this paper we deal with a situation where we have various local ontologies, developed independently from each other, and we are required to build an integrated, global ontology as a mean for extracting information from the local ones. Thus, the main purpose of the global ontology is to provide a unified view through which we can query the various local ontologies.

Most of the work carried out on ontologies for the semantic web is on which language or which method to use to build the global ontology on the basis of the local ones [13, 2]. For example, the Ontology Inference Layer (OIL) [13, 2] proposes to use a restricted form of the expressive and decidable DL studied in [4] to express ontologies for the semantic web.

In this paper, we address what we believe is a crucial problem for the semantic web: how do we specify the mapping between the global ontology and the local ontologies. This aspect is the central one if we want to use the global ontology for answering queries in the context of the semantic web. Indeed, we are not simply using the local ontologies as an intermediate step towards the global one. Instead, we are using the global ontology for accessing information in the local ones. It is our opinion that, although the problem of specifying the mapping between the global and the local ontologies is at the heart of integration in the web, it is not deeply investigated yet.

We argue that even the most expressive ontology specification languages are not sufficient for information integration in the semantic web. In a real world setting, different ontologies

are build by different organizations for different purposes. Hence one should expect the same information to be represented in different forms and with different levels of abstraction in the various ontologies. When mapping concepts in the various ontologies to each other, it is very likely that a concept in one ontology corresponds to a *view* (i.e., a *query*) over the other ontologies. Observe that here the notion of "query" is a crucial one. Indeed, to express mappings among concepts in different ontologies, suitable query languages should be added to the ontology specification language, and considered in the various reasoning tasks, in the spirit of [4, 5]. As a result query processing in this setting is strongly related to view-based query answering in data integration systems [20, 17]. What distinguishes ontology integration from data integration as studied in databases, is that, while in data integration one assumes that each source is basically a databases, i.e., a logical theory with a single model, such an assumption is not made in ontology integration, where a local ontology is an arbitrary logical theory, and hence can have multiple models.

Our main contribution in this paper is to present a general framework for an ontology of integration where the mapping between ontologies is expressed through suitable mechanisms based on queries, and to illustrate the framework proposed with two significant case studies.

The paper is organized as follows. In the next section we set up a formal framework for ontology integration. In Sections 3 and 4, we illustrate the so called global-centric approach and local-centric approach to integration, and we discuss for each of the two approaches a specific case study showing the subtleties involved. In Section 5 we briefly present an approach to integration that goes beyond the distinction between global-centric and local-centric. Finally, Section 6 concludes the paper.

2 Ontology integration framework

In this section we set up a formal framework for *ontology integration systems* (OISs). We argue that this framework provides the basis of an *ontology of integration*. For the sake of simplicity, we will refer to a simplified framework, where the components of an OIS are the global ontology, the local ontologies, and the mapping between the two. We call such systems "one-layered". More complex situations can be modeled by extending the framework in order to represent, for example, mappings between local ontologies (in the spirit of [12, 6]), or global ontologies that act as local ones with respect to another layer.

In what follows, one of the main aspects is the definition of the semantics of both the OIS, and of queries posed to the global ontology. For keeping things simple, we will use in the following a unique semantic domain Δ, constituted by a fixed, infinite set of symbols.

Formally, an OIS $\mathcal{O}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, where $\mathcal{G}$ is the global ontology, $\mathcal{S}$ is the set of local ontologies, and $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is the mapping between $\mathcal{G}$ and the local ontologies in $\mathcal{S}$.

Global ontology. We denote with $\mathcal{A}_\mathcal{G}$ the alphabet of terms of the global ontology, and we assume that the global ontology $\mathcal{G}$ of an OIS is expressed as a theory (named simply $\mathcal{G}$) in some logic $\mathcal{L}_G$.

Local ontologies. We assume to have a set $\mathcal{S}$ of n local ontologies $\mathcal{S}_1, \ldots, \mathcal{S}_n$. We denote with $\mathcal{A}_{\mathcal{S}_i}$ the alphabet of terms of the local ontology $\mathcal{S}_i$. We also denote with $\mathcal{A}_\mathcal{S}$ the union of all the $\mathcal{A}_{\mathcal{S}_i}$'s. We assume that the various $\mathcal{A}_{\mathcal{S}_i}$'s are mutually disjoint, and each one is disjoint from the alphabet $\mathcal{A}_\mathcal{G}$. We assume that each local ontology is expressed as

a theory (named simply $\mathcal{S}_i$) in some logic $\mathcal{L}_{S_i}$, and we use $\mathcal{S}$ to denote the collection of theories $\mathcal{S}_1, \ldots, \mathcal{S}_n$.

Mapping. The mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is the heart of the OIS, in that it specifies how the concepts[1] in the global ontology $\mathcal{G}$ and in the local ontologies $\mathcal{S}$ map to each other.

Semantics. Intuitively, in specifying the semantics of an OIS, we have to start with a model of the local ontologies, and the crucial point is to specify which are the models of the global ontology. Thus, for assigning semantics to an OIS $\mathcal{O} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, we start by considering a *local model* $\mathcal{D}$ for $\mathcal{O}$, i.e., an interpretation that is a model for all the theories of $\mathcal{S}$. We call *global interpretation* for $\mathcal{O}$ any interpretation for $\mathcal{G}$. A global interpretation $\mathcal{I}$ for $\mathcal{O}$ is said to be a *global model for $\mathcal{O}$ wrt $\mathcal{D}$* if:

- $\mathcal{I}$ is a model of $\mathcal{G}$, and
- $\mathcal{I}$ satisfies the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$.

In the next sections, we will come back to the notion of satisfying a mapping wrt a local model. The semantics of $\mathcal{O}$, denoted $sem(\mathcal{O})$, is defined as follows:

$$sem(\mathcal{O}) = \{\, \mathcal{I} \mid \text{there exists a local model } \mathcal{D} \text{ for } \mathcal{O} \text{ s.t. } \mathcal{I} \text{ is a global model for } \mathcal{O} \text{ wrt } \mathcal{D} \,\}$$

Queries. Queries posed to an OIS $\mathcal{O}$ are expressed in terms of a query language $\mathcal{Q}_{\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$ and are intended to extract a set of tuples of elements of Δ. Thus, every query has an associated arity, and the semantics of a query q of arity n is defined as follows. The answer $q^{\mathcal{O}}$ of q to $\mathcal{O}$ is the set of tuples

$$q^{\mathcal{O}} = \{\langle c_1, \ldots, c_n \rangle \mid \text{for all } \mathcal{I} \in sem(\mathcal{O}),\ \langle c_1, \ldots, c_n \rangle \in q^{\mathcal{I}}\ \}$$

where $q^{\mathcal{I}}$ denotes the result of evaluating q in the interpretation $\mathcal{I}$.

As we said before, the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ represents the heart of an OIS $\mathcal{O} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$. In the usual approaches to ontology integration, the mechanisms for specifying the mapping between concepts in different ontologies are limited to expressing direct correspondences between terms. We argue that, in a real-world setting, one needs a much more powerful mechanism. In particular, such a mechanism should allow for mapping a concept in one ontology into a *view*, i.e., a query over the other ontologies, which acquires the relevant information by navigating and aggregating several concepts.

Following the research done in data integration [16, 17], we can distinguish two basic approaches for defining this mapping:

- the *global-centric approach*, where concepts of the global ontology $\mathcal{G}$ are mapped into queries over the local ontologies in $\mathcal{S}$;
- the *local-centric approach*, where concepts of the local ontologies in $\mathcal{S}$ are mapped to queries over the global ontology $\mathcal{G}$.

We discuss these two approaches in the following sections.

[1] Here and below we use the term "concept" for denoting a concept of the ontology.

3 Global-centric approach

In the global-centric approach (aka global-as-view approach), we assume we have a query language $\mathcal{V}_\mathcal{S}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and the mapping between the global and the local ontologies is given by associating to each term in the global ontology a *view*, i.e., a query, over the local ontologies. The intended meaning of associating to a term C in $\mathcal{G}$ a query V_s over $\mathcal{S}$, is that such a query represents the best way to characterize the instances of C using the concepts in $\mathcal{S}$. A further mechanism is used to specify if the correspondence between C and the associated view is *sound*, *complete*, or *exact*. Let $\mathcal{D}$ be a local model for $\mathcal{O}$, and $\mathcal{I}$ a global interpretation for $\mathcal{O}$:

- $\mathcal{I}$ satisfies the correspondence $\langle C, V_s, sound\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if all the tuples satisfying V_s in $\mathcal{D}$ satisfy C in $\mathcal{I}$,
- $\mathcal{I}$ satisfies the correspondence $\langle C, V_s, complete\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if no tuple other than those satisfying V_s in $\mathcal{D}$ satisfies C in $\mathcal{I}$.
- $\mathcal{I}$ satisfies the correspondence $\langle C, V_s, exact\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if the set of tuples that satisfy C in $\mathcal{I}$ is exactly the set of tuples satisfying V_s in $\mathcal{D}$.

We say that $\mathcal{I}$ *satisfies* the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if $\mathcal{I}$ satisfies every correspondence in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$.

The global-centric approach is the one adopted in most data integration systems. In such systems, sources are databases (in general relational ones), the global ontology is actually a database schema (again, represented in relational form), and the mapping is specified by associating to each relation in the global schema one relational query over the source relations. It is a common opinion that this mechanism allow for a simple query processing strategy, which basically reduces to unfolding the query using the definition specified in the mapping, so as to translate the query in terms of accesses to the sources [20]. Actually, when we add constraints (even of a very simple form) to the global schema, query processing becomes even harder, as shown in the following case study.

3.1 A case study

We now set up a global-centric framework for ontology integration, which is based on ideas developed for data integration over global schemas expressed in the Entity-Relationship model [3]. In particular, we describe the main components of the ontology integration system, and we provide the semantics both of the system, and of query answering.

The OIS $O = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}}\rangle$ is defined as follows:

- The *global ontology* $\mathcal{G}$ is expressed in the *Entity-Relationship model* (or equivalently as *UML class diagrams*). In particular, $\mathcal{G}$ may include:
  - typing constraints on relationships, assigning an entity to each component of the relationship;
  - mandatory participation to relationships, saying that each instance of an entity must participate as i-th component to a relationship;
  - ISA relations between both entities and relationships;

  - typing constraints, functional restrictions, and mandatory existence, for attributes both of entities and of relationships.

- The *local ontologies* $\mathcal{S}$ are constituted simply by a relational alphabet $\mathcal{A}_\mathcal{S}$, and by the extensions of the relations in $\mathcal{A}_\mathcal{S}$. For example, such extensions may be expressed as relational databases. Observe that we are assuming that no intensional relation between terms in $\mathcal{A}_\mathcal{S}$ is present in the local ontologies.

- The *mapping* $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ between $\mathcal{G}$ and $\mathcal{S}$ is given by a set of correspondences of the form $\langle C, V_s, sound \rangle$, where C is a concept (i.e., either an entity, a relationship, or an attribute) in the global ontology and V_s is a query over $\mathcal{S}$. More precisely,

  - The mapping associates a query of arity 1 to each entity of $\mathcal{G}$.
  - The mapping associates a query of arity 2 to each entity attribute A of $\mathcal{G}$. Intuitively, if the query retrieves the pair $\langle x, y \rangle$ from the extension of the local ontologies, this means that y is a value of the attribute A of the entity instance x. Thus, the first argument of the query corresponds to the instances of the entity for which A is defined, and the second argument corresponds to the values of the attribute A.
  - The mapping associates a query of arity n to each relationship R of arity n in $\mathcal{G}$. Intuitively, if the query retrieves the tuple $\langle x_1, \ldots, x_n \rangle$ from the extension of the local ontologies, this means that $\langle x_1, \ldots, x_n \rangle$ is an instance of R.
  - The mapping associates a query of arity $n + 1$ to each attribute A of a relationship R of arity n in $\mathcal{G}$. The first n arguments of the query correspond to the tuples of R, and the last argument corresponds to the values of A.

  As specified above, the intended meaning of the query $\mathcal{V}_s$ associated to the concept C is that it specifies how to retrieve the data corresponding to C in the global schema starting from the data at the sources. This confirms that we are following the global-as-views approach: each concept in the global ontology is defined as a view over the concepts in the local ontologies. We do not pose any constraint on the language used to express the queries in the mapping. Since the extensions of local ontologies are relational databases, we simply assume that the language is able to express computations over relational databases.

To specify the semantics of a data integration system, we have to characterize, given the set of tuples in the extension of the various relations of the local ontologies, which are the data satisfying the global ontology. In principle, one would like to have a single extension as model of the global ontology. Indeed, this is the case for most of the data integration systems described in the literature. However, we will show in the following the surprising result that, due to the presence of the semantic conditions that are implicit in the conceptual schema $\mathcal{G}$, in general, we will have to account for a set of possible extensions.

Example 1. Figure 1 shows the global schema $\mathcal{G}_1$ of a data integration system $\mathcal{O}_1 = \langle \mathcal{G}_1, \mathcal{S}_1, \mathcal{M}_1 \rangle$, where Age is a functional attribute, Student has a mandatory participation in the relationship Enrolled, Enrolled isa Member, and University isa Organization. The schema models persons who can be members of one or more organizations, and students who are

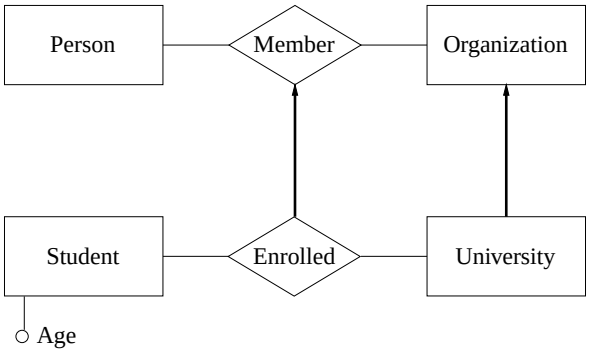


Figure 1: Global ontology of Example 1

enrolled in universities. Suppose that $\mathcal{S}$ is constituted by $\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3, \mathsf{S}_4, \mathsf{S}_5, \mathsf{S}_6, \mathsf{S}_7, \mathsf{S}_8$, and that the mapping $\mathcal{M}_1$ is as follows:

$$\begin{aligned}
\mathsf{Person}(x) &\leftarrow \mathsf{S}_1(x) \\
\mathsf{Organization}(x) &\leftarrow \mathsf{S}_2(x) \\
\mathsf{Member}(x, y) &\leftarrow \mathsf{S}_7(x, z) \wedge \mathsf{S}_8(z, y) \\
\mathsf{Student}(x) &\leftarrow \mathsf{S}_3(x, y) \vee \mathsf{S}_4(x) \\
\mathsf{Age}(x, y) &\leftarrow \mathsf{S}_3(x, y) \vee \mathsf{S}_6(x, y, z) \\
\mathsf{University}(x) &\leftarrow \mathsf{S}_5(x) \\
\mathsf{Enrolled}(x, y) &\leftarrow \mathsf{S}_4(x, y)
\end{aligned}$$

■

From the semantics of the OIS $\mathcal{O}$ it is easy to see that, given a local model $\mathcal{D}$, several situations are possible:

1. No global model exists. This happens, in particular, when the data in the extension of the local ontologies retrieved by the queries associated to the elements of the global ontology do not satisfy the functional attribute constraints.

2. Several global models exist. This happens, for example, when the data in the extension of the local ontologies retrieved by the queries associated to the global concepts do not satisfy the ISA relationships of the global ontology. In this case, it may happen that several ways exist to add suitable objects to the elements of $\mathcal{G}$ in order to satisfy the constraints. Each such ways yields a global model.

Example 2. Referring to Example 1, consider a local model $\mathcal{D}_1$, where S_3 contains the tuple $\langle \mathsf{t}_1, \mathsf{a}_1 \rangle$, and S_6 contains the tuple $\langle \mathsf{t}_1, \mathsf{a}_2, \mathsf{v}_1 \rangle$. The query associated to Age by the mapping $\mathcal{M}_1$ specifies that, in every model of $\mathcal{O}_1$ both tuples should belong to the extension of Age. However, since Age is a functional attribute in $\mathcal{G}_1$, it follows that no model exists for the OIS $\mathcal{O}_1$. ■

Example 3. Referring again to Example 1, consider a local model $\mathcal{D}_2$, where S_1 contains p_1 and p_2, S_2 contains o_1, S_5 contains u_1, S_4 contains t_1, and the pairs $\langle \mathsf{p}_1, \mathsf{o}_1 \rangle$ and $\langle \mathsf{p}_2, \mathsf{u}_1 \rangle$ are in the join between S_7 and S_8. By the mapping $\mathcal{M}_1$, it follows that in every model of $\mathcal{O}_1$, we

have that $\mathsf{p}_1, \mathsf{p}_2 \in \mathsf{Person}$, $\langle \mathsf{p}_1, \mathsf{o}_1 \rangle, \langle \mathsf{p}_2, \mathsf{u}_1 \rangle \in \mathsf{Member}$, $\mathsf{o}_1 \in \mathsf{Organization}$, $\mathsf{t}_1 \in \mathsf{Student}$, and $\mathsf{u}_1 \in \mathsf{University}$. Moreover, since $\mathcal{G}_1$ specifies that Student has a mandatory participation in the relationship Enrolled, in every model for $\mathcal{O}_1$, t_1 *must* be enrolled in a certain university. The key point is that nothing is said in $\mathcal{D}_2$ about *which* university, and therefore we have to accept as models all interpretations for $\mathcal{O}_1$ that differ in the university t_1 is enrolled in. ∎

In the framework proposed, it is assumed that the first problem is solved by the queries extracting data from the extension of the local ontologies. In other words, it is assumed that, for any functional attribute A, the corresponding query implements a suitable data cleaning strategy (see, e.g., [15]) that ensures that, for every local model $\mathcal{D}$ and every x, there is at most one tuple (x, y) in the extension of A (a similar condition holds for functional attributes of relationships).

The second problem shows that the issue of query answering with incomplete information arises even in the global-as-view approach to data integration. Indeed, the existence of multiple global models for the OIS implies that query processing cannot simply reduce to evaluating the query over a single relational database. Rather, we should in principle take *all* possible global models into account when answering a query.

It is interesting to observe that there are at least two different strategies to simplify the setting, and overcome this problem that are frequently adopted in data integration systems [16, 20, 17]:

- Data integration systems usually adopt a simpler data model (often, a plain relational data model) for expressing the global schema (i.e., the global ontology). In this case, the data retrieved from the sources (i.e., the local ontologies) trivially fits into the schema, and can be directly considered as the unique database to be processed during query answering.

- The queries associated to the concepts of the global schema are often considered as exact. In this case, analogously to the previous one, it is easy to see that the only global extension to be considered is the one formed by the data retrieved by the extension of the local ontologies. However, observe that, when data in this extension do not obey all semantic conditions that are implicit in the global ontology, this single extension is not coherent with the global ontology, and the OIS is inconsistent. This implies that query answering in meaningless. We argue that, in the usual case of autonomous, heterogeneous local ontologies, it is very unlikely that data fit in the global ontology, and therefore, this approach is too restrictive, in the sense that the OIS would be often inconsistent.

The fact that the problem of incomplete information is overlooked in current approaches can be explained by observing that traditional data integration systems follow one of the above mentioned simplifying strategies: they either express the global schema as a set of plain relations, or consider the sources as exact (see, for instance, [11, 19, 1]).

In [3] we present an algorithm for computing the set of certain answers to queries posed to a data integration system. The key feature of the algorithm is to reason about both the query and the global ontology in order to infer which tuples satisfy the query in all models of the OIS. Thus, the algorithm does not simply unfold the query on the basis of the mapping, as usually done in data integration systems based on the global-as-view approach. Indeed, the algorithm is able to add more answers to those directly extracted from the local ontologies, by exploiting the semantic conditions expressed in the conceptual global schema.

Let $\mathcal{O} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$ be an OIS, let $\mathcal{D}$ be a local model, and let Q be a query over the global ontology $\mathcal{G}$. The algorithm is constituted by three major steps.

1. From the query Q, obtain a new query $expand_{\mathcal{G}}(Q)$ over the elements of the global ontology G in which the knowledge in $\mathcal{G}$ that is relevant for Q has been compiled in.

2. From $expand_{\mathcal{G}}(Q)$, compute the query $unfold_{\mathcal{M}_{\mathcal{G},\mathcal{S}}}(expand_{\mathcal{G}}(Q))$, by unfolding $expand_{\mathcal{G}}(Q)$ on the basis of the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$. The unfolding simply substitutes each atom of $expand_{\mathcal{G}}(Q)$ with the query associated by $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ to the element in the atom. The resulting $unfold_{\mathcal{M}_{\mathcal{G},\mathcal{S}}}(expand_{\mathcal{G}}(Q))$ is a query over the relations in the local ontologies.

3. Evaluate the query $unfold_{\mathcal{M}_{\mathcal{G},\mathcal{S}}}(expand_{\mathcal{G}}(Q))$ over the local model $\mathcal{D}$.

The last two steps are quite obvious. Instead, the first one requires to find a way to compile into the query the semantic relations holding among the concepts of the global schema $\mathcal{G}$. A way to do so is shown in [3]. The query $expand_{\mathcal{G}}(Q)$ returned by the algorithm is exponential wrt to Q. However, $expand_{\mathcal{G}}(Q)$ is a union of conjunctive queries, which, if the queries in the mapping are polynomial, makes the entire algorithm polynomial in data complexity.

Example 4. Referring to Example 3, consider the query Q_1 to $\mathcal{O}_1$:

$$\mathsf{Q}_1(x) \leftarrow \mathsf{Member}(x, y) \wedge \mathsf{University}(y)$$

It is easy to see that $\{\mathsf{p}_2, \mathsf{t}_1\}$ is the set of certain answers to Q_1 with respect to $\mathcal{O}_1$ and $\mathcal{D}_2$. Thus, although $\mathcal{D}_2$ does not indicate in which university t_1 is enrolled, the semantics of $\mathcal{O}_1$ specifies that t_1 is enrolled in *a* university in all legal database for $\mathcal{O}_1$. Since Member is a generalization of Enrolled, this implies that t_1 is in $\mathsf{Q}_1^{\mathcal{O}}$, and hence is in $unf_{\mathcal{M}_1}(exp_{\mathcal{G}_1}(\mathsf{Q}_1))$ evaluated over $\mathcal{D}_2$. ∎

4 Local-centric approach

In the local-centric approach (aka local-as-view approach), we assume we have a query language $\mathcal{V}_{\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$, and the mapping between the global and the local ontologies is given by associating to each term in the local ontologies a *view*, i.e., a query over the global ontology. Again, the intended meaning of associating to a term C in $\mathcal{S}$ a query V_g over $\mathcal{G}$, is that such a query represents the best way to characterize the instances of C using the concepts in $\mathcal{G}$. As in the global-centric approach, the correspondence between C and the associated view can be either sound, complete, or exact. Let $\mathcal{D}$ be a local model for $\mathcal{O}$, and $\mathcal{I}$ a global interpretation for $\mathcal{O}$:

- $\mathcal{I}$ satisfies the correspondence $\langle V_g, C, sound\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if all the tuples satisfying C in $\mathcal{D}$ satisfy V_g in $\mathcal{I}$,

- $\mathcal{I}$ satisfies the correspondence $\langle V_g, C, complete\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if no tuple other than those satisfying C in $\mathcal{D}$ satisfies V_g in $\mathcal{I}$,

- $\mathcal{I}$ satisfies the correspondence $\langle V_g, C, exact\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if the set of tuples that satisfy C in $\mathcal{D}$ is exactly the set of tuples satisfying V_g in $\mathcal{I}$.

As in the global-centric approach, we say that $\mathcal{I}$ *satisfies* the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if $\mathcal{I}$ satisfies every correspondence in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$.

Recent research work on data integration follows the local-centric approach [20, 17, 18, 6, 8]. The major challenge of this approach is that, in order to answer a query expressed over the

global schema, one must be able to reformulate the query in terms of queries to the sources. While in the global-centric approach such a reformulation is guided by the correspondences in the mapping, here the problem requires a reasoning step, so as to infer how to use the sources for answering the query. Many authors point out that, despite its difficulty, the local-centric approach better supports a dynamic environment, where local ontologies can be added to the systems without the need for restructuring the global ontology.

4.1 A case study

We present here an OIS architecture based on the use of Description Logics to represent ontologies [6, 7]. Specifically, we adopt the Description Logic $\mathcal{DLR}$, in which both classes and n-ary relations can be represented [4]. We first introduce $\mathcal{DLR}$, and then we illustrate how we use the logic to define an OIS.

4.1.1 The Description Logic $\mathcal{DLR}$

Description Logics[2] (DLs) are knowledge representation formalisms that are able to capture virtually all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases [9, 10].

One of the distinguishing features of these logics is that they have optimal reasoning algorithms, and practical systems implementing such algorithms are now used in several projects.

In DLs, the domain of interest is modeled by means of *concepts* and *relations*, which denote classes of objects and relationships, respectively. Here, we focus our attention on the DL $\mathcal{DLR}$ [4, 6], whose basic elements are *concepts* (unary relations), and *n-ary relations*. We assume to deal with an alphabet $\mathcal{A}$ constituted by a finite set of atomic relations, atomic concepts, and *constants*, denoted by P, A, and a, respectively. We use R to denote arbitrary relations (of given arity between 2 and n_{max}), and C to denote arbitrary concepts, respectively built according to the following syntax:

$$\begin{array}{rcl} C & ::= & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists[i]R \mid (\leq k\,[i]R) \\ R & ::= & \top_n \mid P \mid i/n : C \mid \neg R \mid R_1 \sqcap R_2 \end{array}$$

where i denotes a component of a relation, i.e., an integer between 1 and n_{max}, n denotes the *arity* of a relation, i.e., an integer between 2 and n_{max}, and k denotes a nonnegative integer. We consider only concepts and relations that are *well-typed*, which means that only relations of the same arity n are combined to form expressions of type $R_1 \sqcap R_2$ (which inherit the arity n), and $i \leq n$ whenever i denotes a component of a relation of arity n.

The semantics of $\mathcal{DLR}$ is specified as follows. An *interpretation* $\mathcal{I}$ is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$, and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each constant an element of $\Delta^{\mathcal{I}}$ under the unique name assumption, to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$, such that the conditions in Figure 2 are satisfied. Observe that, the "$\neg$" constructor on relations is used to express difference of relations, and not the complement [4].

A $\mathcal{DLR}$ knowledge base is a set of inclusion assertions of the form

$$C_1 \sqsubseteq C_2 \qquad\qquad R_1 \sqsubseteq R_2$$

[2] See `http://dl.kr.org` for the home page of Description Logics.

$$
\begin{array}{rcl}
\top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(\exists [i] R)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists \langle d_1, \ldots, d_n \rangle \in R^{\mathcal{I}}.\, d_i = d\} \\
(\leq k\, [i] R)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \sharp\{\langle d_1, \ldots, d_n \rangle \in R_1^{\mathcal{I}} \mid d_i = d\} \leq k\} \\
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n \\
P^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} \\
i/n : C^{\mathcal{I}} & = & \{\langle d_1, \ldots, d_n \rangle \in \top_n^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\} \\
(\neg R)^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}} \\
(R_1 \sqcap R_2)^{\mathcal{I}} & = & R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}
\end{array}
$$

Figure 2: Semantic rules for $\mathcal{DLR}$ (P, R, R_1, and R_2 have arity n)

where C_1 and C_2 are concepts, and R_1 and R_2 are relations of the same arity. An inclusion assertion $C_1 \sqsubseteq C_2$ (resp., $R_1 \sqsubseteq R_2$) is satisfied in an interpretation $\mathcal{I}$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ (resp., $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$). An interpretation is a *model* of a knowledge base $\mathcal{K}$, if it satisfies all assertions in $\mathcal{K}$. $\mathcal{K}$ *logically implies* an inclusion assertion ρ if ρ is satisfied in all models of $\mathcal{K}$.

Finally, we introduce the notion of query expression in $\mathcal{DLR}$. We assume that the alphabet $\mathcal{A}$ is enriched with a finite set of variable symbols, simply called *variables*. A *query expression* Q over a $\mathcal{DLR}$ knowledge base $\mathcal{K}$ is a non-recursive datalog query of the form

$$Q(\vec{x}) \leftarrow conj_1(\vec{x}, \vec{y}_1) \vee \cdots \vee conj_m(\vec{x}, \vec{y}_m)$$

where each $conj_i(\vec{x}, \vec{y}_i)$ is a conjunction of *atoms*, and $\vec{x}$, $\vec{y}_i$ are all the variables appearing in the conjunct. Each atom has one of the forms $R(\vec{t})$ or $C(t)$, where $\vec{t}$ and t are variables in $\vec{x}$ and $\vec{y}_i$ or constants in $\mathcal{A}$, R is a relation of $\mathcal{K}$, and C is a concept of $\mathcal{K}$. The number of variables of $\vec{x}$ is called the *arity* of Q, and is the arity of the relation denoted by the query Q. We observe that the atoms in query expressions are arbitrary $\mathcal{DLR}$ concepts and relations, freely used in the assertions of the KB.

Given an interpretation $\mathcal{I}$, a query expression Q of arity n is interpreted as the set $Q^{\mathcal{I}}$ of n-tuples of constants $\langle c_1, \ldots, c_n \rangle$, such that, when substituting each c_i for x_i, the formula

$$\exists \vec{y}_1.conj_1(\vec{x}, \vec{y}_1) \vee \cdots \vee \exists \vec{y}_m.conj_m(\vec{x}, \vec{y}_m)$$

evaluates to true in $\mathcal{I}$.

$\mathcal{DLR}$ is equipped with effective reasoning techniques that are sound and complete with respect to the semantics. In particular, checking whether a given assertion logically follows from a set of assertions is EXPTIME-complete in (assuming that numbers are encoded in unary), and query containment, i.e., checking whether one query is contained in another one in every model of a set of assertions, is EXPTIME-hard and solvable in 2EXPTIME [4].

4.1.2 $\mathcal{DLR}$ local-centric OIS

We now set up a local-centric framework for ontology integration, which is based on ideas developed for data integration over $\mathcal{DLR}$ knowledge bases [6, 5]. In particular, we describe

the main components of the ontology integration system, and we provide the semantics both of the system, and of query answering.

In this setting, an OIS $O = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$ is defined as follows:

- The *global ontology* $\mathcal{G}$ is a $\mathcal{DLR}$ knowledge base.
- The *local ontologies* $\mathcal{S}$ are again seen as a set of relations each giving the extension of an ontology-concept in the ontology. We observe that again we have only extensional knowledge on such relations in $\mathcal{S}$.
- The *mapping* $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ between $\mathcal{G}$ and $\mathcal{S}$ is given by a set of correspondences of the form $\langle V_g, T, as \rangle$, where T is a relation of a local ontology, $\mathcal{V}_g$ is a query expression over $\mathcal{G}$, and as is either $sound$, $complete$, or $exact$.

Observe that we could partition the global ontology in several parts, one for each local ontology, modeling the intensional knowledge on the local ontology wrt the OIS, plus one for the reconciled global view of such ontologies. By making use of the so called interschema assertions [12] the different parts can be related to each at the intesional level. For simplicity we do not deal with interschema assertion in this case study, however it is immediate to extend the framework presented here to include them as well [6, 7].

Query answering in this setting requires quite sophisticated techniques that take into account the knowledge both in the global ontology and in the mapping in answering a query posed over the global ontology with the data contained in the local ontologies. Such query answering techniques are studied in [5].

Example 5. Consider for example the OIS $\mathcal{O}_d = \langle \mathcal{G}_d, \mathcal{S}_d, \mathcal{M}_d \rangle$ defined as follows:

- The global ontology $\mathcal{G}_d$ is the $\mathcal{DLR}$ knowledge base

$$\begin{array}{rcl} \mathsf{American} \sqcap \exists[1](\mathsf{RELATIVE} \sqcap 2 : \mathsf{Doctor}) & \sqsubseteq & \mathsf{Wealthy} \\ \mathsf{Surgeon} & \sqsubseteq & \mathsf{Doctor} \end{array}$$

  expressing that Americans who have a doctor as relative are wealthy, and that each surgeon is also a doctor.

- The set $\mathcal{S}_d$ of local ontologies consists of two ontologies, containing respectively the relations T_1 and T_2, with extensions $\{\mathsf{ann}, \mathsf{bill}\}$ and $\{\mathsf{ann}, \mathsf{dan}\}$.
- The mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is $\{\langle \mathsf{V}_1, \mathsf{T}_1, sound \rangle, \langle \mathsf{V}_2, \mathsf{T}_2, sound \rangle\}$, with

$$\begin{array}{rcl} \mathsf{V}_1(x) & \leftarrow & \mathsf{RELATIVE}(x, y) \wedge \mathsf{Surgeon}(y) \\ \mathsf{V}_2(x) & \leftarrow & \mathsf{American}(x) \end{array}$$

Given the query expression $\mathsf{Q}_w(x) \leftarrow \mathsf{Wealthy}(x)$ over $\mathcal{G}_d$, asking for those who are wealthy, we have that the only answer in $\mathsf{Q}_w^{\mathcal{O}_d}$ is ann. Consider an additional local ontology, consisting of a relation T_3 with an extension not containing bill, and mapped to $\mathcal{G}$ by the correspondence $\langle \mathsf{V}_3, \mathsf{T}_3, exact \rangle$, with $\mathsf{V}_3(x) \leftarrow \mathsf{Wealthy}(x)$. Then, from the constraints in $\mathcal{G}_d$ and the information we have on the correspondences, we can conclude that bill is not an answer to the query asking for the Americans. ∎

5 Combining the global-centric and local-centric approaches

The global-centric and the local-centric approach can be combined together into an approach using unrestricted mappings, in which the restrictions on the direction of the correspondence between global and local ontologies are overcome [14]. In the unrestricted approach, we have both a query language $\mathcal{V}_\mathcal{S}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and a query language $\mathcal{V}_\mathcal{G}$ over the alphabet $\mathcal{A}_\mathcal{G}$, and the mapping between the global and the local ontologies is given by relating views over the global ontology to views over the local ontologies. Again, the intended meaning of relating the view V_g over the global ontology to the view V_s over the local ontology is that V_s represents the best way to characterize the objects satisfying V_g in terms of the concepts in $\mathcal{S}$. Analogously to the other cases, the correspondences between V_g and V_s can be characterized as sound, complete, or exact. Let $\mathcal{D}$ be a local model for $\mathcal{O}$, and $\mathcal{I}$ a global interpretation for $\mathcal{O}$:

- $\mathcal{I}$ satisfies the correspondence $\langle V_g, V_s, sound\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if all the tuples satisfying satisfying V_s in $\mathcal{D}$ satisfy V_g in $\mathcal{I}$,
- $\mathcal{I}$ satisfies the correspondence $\langle V_g, V_s, complete\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if no tuple other than those satisfying V_s in $\mathcal{D}$ satisfy V_g in $\mathcal{I}$,
- $\mathcal{I}$ satisfies the correspondence $\langle V_g, V_s, exact\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if the set of tuples that satisfy V_g in $\mathcal{I}$ is exactly the set of tuples satisfying V_s in $\mathcal{D}$.

Again, we say that $\mathcal{I}$ *satisfies* the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$, if $\mathcal{I}$ satisfies every correspondence in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{D}$.

Example 6. Consider the OIS $\mathcal{O}_u = \langle \mathcal{G}_u, \mathcal{S}_u, \mathcal{M}_u\rangle$, where both $\mathcal{G}_u$ and the two ontologies S_1 and S_2 forming $\mathcal{S}_u$ are simply sets of relations with their extensions.

- The global ontology $\mathcal{G}_u$ contains two binary relations, WorksFor, denoting researchers and projects they work for, and Area, denoting projects and research areas they belong to.
- The local ontology S_1 contains a binary relation InterestedIn denoting persons and fields they are interested in, and the local ontology S_2 contains a binary relation GetGrant, denoting researchers and grants assigned to them, and a binary relation GrantFor denoting grants and projects they refer to.
- The mapping $\mathcal{M}_u$ is formed by the following correspondences
  - $\langle \mathsf{V}_1, \mathsf{InterestedIn}, complete\rangle$, with $\mathsf{V}_1(r, f) \leftarrow \mathsf{WorksFor}(r, p) \wedge \mathsf{Area}(p, f)$
  - $\langle \mathsf{WorkFor}, \mathsf{V}_2, sound\rangle$, with $\mathsf{V}_2(r, p) \leftarrow \mathsf{GetGrant}(r, g) \wedge \mathsf{GrantFor}(g, p)$

This situation can be represented neither in the global-centric nor in the local-centric approach. ∎

Query answering in this approach is largely unexplored, mainly because it combines the difficulties of the other ones. However, in a real world setting, this may be the only approach that provides the appropriate expressive power.

6 Conclusions

We have presented a general framework for ontology integration, where a global ontology is used to provide a unified view for querying local ontologies, as in the semantic web. The framework represents a sort of design space for the problem of integrating ontologies within semantic web applications. We have argued that the mapping between the global and the local ontologies is the main aspect of the framework, and we have discussed various approaches for specifying such a mapping. Independently of the approach, we have stressed that the notion of query is crucial for the task of ontology integration.

The two case studies we have presented have shown the need of sophisticated techniques for query answering in an ontology integration system. The two case studies illustrated simplified settings, drawn from data integration. One should expect things to become even more complex when ontology integration is considered in its full generality. Recently several proposals have been made, based on the idea of expressing ontologies as knowledge bases, e.g., in Description Logics [13, 2], and applying automated reasoning techniques for several services in the design of and the interaction with the semantic web. We believe however that such an idea needs to be extended by considering queries as first order citizens and having the ability to reason on them.

References

[1] M. Bouzeghoub and M. Lenzerini. Special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 2001. To appear.

[2] J. Broekstra, M. Klein, D. Fensel, and I. Horrocks. Adding formal semantics to the Web: building on top of RDF Schema. In *Proc. of the ECDL 2000 Workshop on the Semantic Web*, 2000.

[3] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, 2001. To appear.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[5] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[7] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 361–371, 2000.

[9] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.

[10] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[11] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *RIDE-DOM*, pages 124–131, 1995.

[12] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[13] S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, S. Melnik, M. Klein, and J. Broekstra. Knowledge representation on the web. In *Proc. of the 2000 Description Logic Workshop (DL 2000)*, pages 89–97. CEUR Electronic Workshop Proceedings, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/, 2000.

[14] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[15] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An extensible framework for data cleaning. Technical Report 3742, INRIA, Rocquencourt, 1999.

[16] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997.

[17] A. Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999.

[18] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

[19] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. D. Ullman, and M. Valiveti. Capability based mediation in TSIMMIS. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 564–566, 1998.

[20] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

A Scalable Framework for the Interoperation of Information Sources

Prasenjit Mitra, Gio Wiederhold, and Stefan Decker *
Infolab, Stanford University
Stanford, CA, USA 94305
{mitra, gio, stefan}@db.stanford.edu

Abstract.

Resolving heterogeneity among information systems is a crucial problem if we wish to gain value from the many distributed resources available to us. Problems of heterogeneity in hardware, operating systems, and data structures have been widely addressed, but issues of diverse semantics have been handled mainly in an ad-hoc fashion. In this paper, we present ONION*, a system based a scalable approach to interoperation of information systems by articulating their associated ontologies. An articulation focuses on the semantically relevant intersection of information resources with respect to a type of application. However, ontologies obtained from diverse sources are represented using different conceptual models. We have designed a simple intermediate conceptual model - the* ONION *conceptual model - that we use to transform ontologies into before we generate semantic correspondences or articulations between them.*

In ONION*, application-dependent articulation rules that capture the correspondence between concepts in different ontologies are established between source ontologies semi-automatically. Finally we present an ontology algebra, based on the articulation rules, for the composition of ontologies.*

1 Introduction

Today a large number of diverse information sources - databases, knowledge bases, collections of documents - are available on the Internet. Often, we cannot answer a query from a single source, and need to compose knowledge from multiple sources. Intelligent searching and querying on the World Wide Web - the largest collection of distributed information and knowledge sources - often requires composing information from heterogeneous information sources. Today, the bulk of this composition is done by the end-user. Not only is this extremely tedious and time-consuming, but also, often, the end-user does not have any idea of the semantics used by the builder of the information source. In this paper, we present a brief overview of the ONION (ONtology compositION) system, which takes a principled approach to enable semi-automatic interoperation among heterogeneous information sources.

* This work was partially supported by a grant from the Air Force Office of Scientific Research (AFOSR).

1.1 Heterogeneity

Most information sources have been independently constructed and are autonomously maintained. Attempts have been made to integrate information from these various information sources into a monolithic information source [1], [2]. Such an approach creates maintenance and scalability problems. When an information source is to be added, the large information source must be restructured. Often such maintenance leads to substantial delays [3].

Some researchers have tried to first build a standard ontology or global schema and then build information sources that conform to the ontology or schema [4], [5]. Even though the approach has worked for small communities, it is almost impossible to come up with an agreed-to-standard for knowledge in larger domains, especially among groups that have different applications in mind.

Besides, it is prohibitively expensive to restructure existing knowledge so that it conforms to the standard ontology even if such a beast ever came into being.

1.2 Maintenance

Everyday new discoveries expand our knowledge, and change the views of the universe that we live in. Therefore, even if information sources start off with a common ontology, such an ontology has to be updated periodically. The maintainers of the information sources that use the standard ontology will have to agree on the updates being proposed and on the restructuring of the ontology. They may have entirely different applications in mind or may not subscribe to a newly discovered theory. Furthermore, some participants might see the changes required to support the proposed updates as an unnecessary imposition since restructuring the information source will require substantial effort on their part. Thus generating new consensus on updates to the standard ontology is a time-consuming and tenuous process. For fast changing fields, arriving at a consensus within a short period of time is not even feasible. Therefore, we need a system where the information sources are autonomously maintained.

1.3 A Realistic Setting

We, believe that the information sources should be autonomous and we should not require them to conform to a standard ontology in order to allow composition of knowledge from them. Instead of integrating information sources, we intend to enable interoperation among them.

Unfortunately, the composition of knowledge from multiple independently maintained information sources is a hard problem. Independently constructed information sources are heterogeneous and often use different vocabularies and conceptual models. The organization of class-subclass hierarchies are substantially different. Often, they use different terms to represent the same concept and the same term to represent entirely different concepts. In order to interoperate among such information sources we need to resolve their semantic heterogeneity.

Karp [6] proposes a strategy for database interoperation. We extend Karp's approach to apply to not only databases, but also to knowledge bases and information sources.

As in [7], [8], and [6], we assume that information sources are independently created and maintained. In Karp's system, each database comes with a schem a which is saved in a Knowledge Base of Databases. Correspondingly, we assume that associated with each in-

formation source is an ontology. However, we do not require all ontologies to be saved in a central repository.

The ontologies associated with information sources are based on some existing, known vocabularies and conceptual models. Native drivers and wrappers provide access to the ontol ogies and help us restructure the information if needed. We establish application-specific *articulation rules*, i.e., rules that establish correspondence between concepts in different ontologies, semi-automatically.

Queries are rewritten using the articulation rules. Before a query is dispatched to a source, the terms in the query are rewritten using the articulation rules that indicate the semantic correspondence between the terms in the query and those in the source. This rewriting ensures that a source gets a query that conforms to the vocabulary and the semantics of the source. During query planning, optimization is enabled based on the algebraic properties of the operations.

In this paper, we describe the ONION system and highlight our approach to interoperation. In Section 2, we describe the common conceptual model that ONION uses for its internal representation of ontologies. In Section 3 we discuss the semi-automatic articulation of ontologies. In Section 4 we outline an Ontology Algebra that we use to compose information from diverse sources. Section 5 concludes the paper.

2 The ONION Conceptual Model

The heterogeneity among information sources needs to be resolved to enable meaningful information exchange or interoperation among them. The two major sources of heterogeneity among the sources are as follows. First, different sources use different conceptual models and modeling languages to represent their data and meta-data. Second, sources using the same conceptual model differ in their semantics. The ONION system uses a common ontology format, which we have described below. It first converts all external ontologies to this common format and then resolves the semantic heterogeneity among the objects in the ontologies that it is articulating.

Melnik, et al., [9] have shown how to convert ontologies and different classes of conceptual models into those using one common format. For example, say one information source uses UML [10] and another using DAML+OIL [11]. ONION will convert the ontologies associated with both information sources to the ONION *conceptual model* described below. Since the number of classes of such conceptual models that are in use and that we want to support is small, we will provide wrappers which will convert from these models to the ONION format.

Instead of converting all ontologies from their native models to the ONION format, an alternative is to do so declaratively. That is, first generate rules that correlate parts of one ontology to parts of another based on semantic similarity. Then these rules could be used to transform ontologies as required. However, this approach would require us to create and manipulate articulation rules that would not only have semantic information but also have information about how we should transform the conceptual models underlying each ontology. These rules would be more complex since they would have information about reformating the ontologies, and would be less usable than the rules required once both ontologies have been converted to a common format. Besides, by converting to the ONION format, we eliminate the necessity of n^2 pariwise conversions among n ontologies and instead reduce it to n

conversions (of all the ontologies to the common format).

We solve the problem of establishing correspondences among ontology formats and the problem of establishing articulations among the concepts in the ontologies differently because we believe that the small number of conceptual modeling formats that we intend to support (currently XML, RDF, DAML+OIL) can be converted to use one common conceptual model, whereas the number of concepts and thus objects used in ontologies are rather large and creating a huge, integrated, common, global ontology is untenable and unmaintainable.

Information sources were, are and will be modeled using different conceptual models. We do not foresee the creation of a *de facto* standard conceptual model that will be used by all information sources. On the other hand, we need a common ontology format for our internal representation. We use the ONION format to represent the source ontologies and manipulate them to create the articulation ontology. The design choices for the conceptual model that we will transform the various source ontologies to range from the least common denominator of the different conceptual models used by the various sources to the greatest common multiple of them. Instead of choosing a model that has various complex features that capture the intricacies of all the conceptual models, we strive to keep our model simple.

2.1 A Graph-Oriented Conceptual Model

Our common conceptual model for the internal representation of ontologies is based on the work done by Gyssens, et al.,[12]. In its core, we represent an ontology as a graph. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E.

An edge *e* is written as (n_1, α, n_2) where n_1 and n_2 are two nodes belonging to the set of nodes V and α is the label of the edge between them. The label of a node *n* is given by a function $\lambda(n)$ that maps the node to non-null string. In the context of ontologies, the label is often a noun-phrase that represents a concept. The label α of an edge $e = (n_1, \alpha, n_2)$ is a string given by $\alpha = \delta(e)$. The label of an edge is the name of a semantic relationship among the concepts and can be null if the relationship is not known. The domain of the functions λ and δ is the universal set of all nodes and edges respectively (from all graphs) and the range is the set of strings (from all lexicons). For the rest of the paper, we will assume that the function λ maps a node to a unique label (the concatenation of the name of the node in the ontology and the name of the ontology), and thus will use the label of a node as a unique identifier of the node. To represent an edge, we can substitute the label of a node for a node and write edge $e = (\lambda(n_1), \alpha, \lambda(n_2)$.

The graph in the ONION conceptual model can be expressed using RDF [13]. Each edge in our graph is coded as an RDF sentence, with the two nodes being the subject and the predicate and the relationship being the property. However, in order to keep our model simple, we have not included the containers that provide collection semantics in RDF. If the children of a node need to be ordered we use a special relationship, as explained below. By choosing RDF, we can use the various tools that are available and do not have to write parsers and other tools for our model.

The set of logical rules R are rules expressed in a logic-based language. Although, theoretically, it might make sense to use first-order logic as the rule language due to its greater expressive power, to limit the computational complexity we will use a simpler language like Horn Clauses. A typical rule $r \in R$ is of the form $CompoundStatement \Rightarrow Statement$.

A *CompoundStatement* is the conjunction of multiple Statements. A *Statement* is of the form $(Concept\ Relationship\ Concept)$. A *Concept* can either be a label of a node in the ontology graph or a variable that can be bound to a node (in the ontology graph) representing a concept. A $Relationship$, as in an edge label in the ontology graph, expresses a relation between the two $Concept$s. A detailed description of the rule language can be found in [14].

2.2 Semantic Relationships in ONION

The ONION *articulation generator* can easily derive better semantic matches among concepts in a pair of ontologies if it has some semantic information about the relationships used in the ONION ontology model. Certain conceptual models allow only strictly-typed relationships with pre-defined semantics. For instance, relationships like SubClassOf, AttributeOf, etc., have very clearly defined semantics in most object-relational databases. A system that knows the exact semantics of the relationships in a conceptual model can use the information, e.g., to find better matches between concepts in two ontologies or to perform type-checking and flag errors.

Other models allow any user-defined relationships without any restriction. For instance, relationships like $OwnerOf$ tend to be interpreted according to the semantics associated to it by the local application. Such relationships need not be strictly typed and a general system that imports such a model does not know of the application-specific semantic interpretation of the relationships. This approach provides enormous flexibility and can accommodate a large number of relationships. However, since the semantics of these relationships are not exactly known by the system, it cannot use them for matching related concepts or for type-checking.

The ONION conceptual modeling encourages the use of a set of strictly-typed relationships with precisely defined semantics. The set of relationships that our articulation generator knows the semantics of is $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$.

In ONION, we assign the conventional semantics to each of these relationships. Some of these relationships impose type-restrictions on the two nodes they relate. Some of the relationships (like $SubClassOf$, $InstanceOf$) are somewhat similar to those in RDF-Schema but the set of relationships that have defined semantics in our conceptual model is different and much smaller to maintain its simplicity.

The following is a description of the semantics of he set of pre-defined relationships available in our common conceptual model:

SubClassOf: The relationship is used to indicate that one concept is a subclass of another. The two concepts that it relates must be of type Class. For example, the statement $(Car\ SubClassOf\ Vehicle)$ denotes that the concept Car is a subclass of concept $Vehicle$. That is any instance of the class Car is also an instance of the class $Vehicle$ and all the attributes of the class $Vehicle$ are also attributes of the class Car. The relationship $SubClassOf$ is transitive and in the absence of an explicit rule in an ontology that states the $SubClassOf$ relationship is transitive, we will add one to the ontology before reasoning or rewriting the queries using the rules.

AttributeOf: This relationship indicates that a concept is an attribute of another concept, e.g., an edge $(ConceptA\ AttributeOf\ ConceptB)$ indicates that $ConceptA$ is an attribute of $ConceptB$. $ConceptB$ has to be of type Class or of type Object and $ConceptA$ is of type Class. This relationship, also referred to as PropertyOf in some information models, has typically the same semantics as attributes in (object-)relational databases .

PartOf: This relationship indicates that a concept is a part of another concept, e.g., an edge $(Chassis\ PartOf\ Car)$ indicates that $Chassis$ is part of a Car. The first concept is of type Class while the second concept can be of type Class or Object. In relational databases, such relationships are often coded as attributes, but we believe that this relationship is sufficiently different semantically from the relationship $AttributeOf$ to warrant separate consideration.

InstanceOf: This relationship indicates that an object is an instance of a class. Therefore, the first concept in the relationship is of type object and the second of type Class. For example, an edge $(MyCar\ InstanceOf\ Car)$ indicates that $MyCar$ is an instance of the Class Car.

ValueOf: This relationship is used to indicate the value of an attribute of an object, e.g., $(\text{"}29\text{"}\ ValueOf\ Age)$. Thus, the first concept is of type literal and the second of type Class. Typically, the second concept (in our example, the class Age), in turn has an edge (in our example, $(Age\ AttributeOf\ PersonA)$) from the object it describes.

2.3 Sequences

XML is becoming the dominant format for expressing data and meta-data on the web. Like SGML and other markup languages primarily designed to express documents, XML imposes order among its elements. By itself, the graphical ONION model, described above, does not impose order among the children of a node. In order to express order, we introduce a special relationship, namely $Sequence$, which is very similar to the container $Sequence$ in RDF. For example, a list ranking cars can be described using the edges $(MoneyLineRanking\ Sequence\ CarRankingList$ $1\ HondaAccord)$, and $(CarRankingList\ :2\ FordTaurus)$. The intermediate node Car-RankingList represents the list object and its elements form an ordered sequence. In an edge of the form $(ConceptA\ Sequence\ ConceptB)$ the first concept can be a class or an object and the second concept is an object representing the list. The individual elements of the list can be objects or classes and are related to the list-object via the relationships $:1, :2, \ldots, :N$ where the list has N elements.

In ONION conceptual model, we do not require that every relationship must belong to the small set of relationships whose semantics are predefined. The model is flexible enough to allow any other user-defined relationship. The articulation generator will not be able to use the relationships, whose semantics it is not aware of, unless the semantics are captured using rules in the source ontology. For example, if the source ontology uses a relationship $Is-A$ and has a rule that says that "Is-A" is transitive, the articulation generator can use that information to generate matches. The articulation rules that the articulation generator generates uses only the relationships whose semantics are predefined to establish correspondences among nodes in the source ontologies.

The articulation generator generates matches among nodes in the two source ontologies that is supplied to it and does not attempt to match relationships among ontologies. The articulation generator uses only relationships whose semantics are clearly defined to it to derive meaningful matches among the nodes and ignores the relationships that it does not know the semantics of. Therefore, if two RDF models have the relationships "Buyer" and "Owner" and for the purposes of the application we want to generate a match between the two, we need to represent these relationships as nodes in the ONION model and then run the articulation generator to match them.

2.4 Reference and Subsumption

In conceptual models, especially those used to model documents, like XML, SGML, OEM etc. [15], where there are nested objects and entities, an object is modeled as a subtree in a graph. The entire subtree rooted at a node comprises the object that the node represents. When a query asks for the object, the entire subtree is returned. Such models assume that an object subsumes all objects that are in its subtree. If any relationship needs to be expressed between two objects a reference to the second object is used. The reference is denoted by having a node with the the identifier of the second object and having an edge to this node. The use of this additional node that refers to a different object helps preserve the tree structure of the models, which is required for documents, since they are in essence serialized entities.

In our model, however, even though many of the relationships, with pre-defined semantics, are essentially subsumptive in nature, we intend to keep the concept of an object as simple as possible. Faced with the question of defining the scope of an object in our common conceptual model, we take the minimal approach. In our world, a single node represents a concept: a class, an object, or a value. All edges are referential in nature. Thus, when a query asks to select an object, only the node representing the object is returned and not the entire subtree rooted at the node. This minimal definition of an object helps us keep the articulation rules and the resulting ontology intersections as small as possible. As we will see later, the larger the intersection, the greater the cost when using the articulation to answer queries. Thus we make the choice to keep the definition of an object as simple as possible.

Apart from the graph model, our conceptual model allows us to declaratively supply rules. Some features in other models can be converted using the rules to capture their semantics. If this is not possible, relationships which are not interpreted by ONION can be used. Some features still cannot be expressed using the ONION model.

The common conceptual model is used to bring ontologies to a common format - so that the articulation generator needs to understand only one format. So if a feature cannot be translated into our common conceptual model, it will not be matched with similar features carrying similar semantic messages in other ontologies. However, such information will still be accessible from the individual ontology and the engine associated with the individual sources.

We resolve the heterogeneity with respect to ontology models and modeling languages by building wrappers that convert ontologies using various conceptual models to an ontology in our common conceptual model. However, the second problem of semantic heterogeneity among the concepts used in the source models still remains. In the next section, we will summarize various methods that we use to automatically suggest ontology articulations.

3 Resolving Semantic Heterogeneity

An important requirement for the application scenarios that our system will be used for is high precision. In distinction to research tasks, casual browsing, and web-surfing, the cost of eliminating false hits is very high in business environments. At this point we believe that resolving semantic heterogeneity entirely automatically is not feasible. We, therefore, advocate a semi-automatic approach wherein an automatic *articulation generator* suggests matches between concepts in the two ontologies it is articulating. A human expert, knowledgeable about the semantics of concepts in both ontologies, validates the generated suggested matches using a GUI tool. An expert can delete a suggested match or say that the match is irrelevant for the

application at hand. The expert can also indicate new matches that the articulation generator might have missed. The process of constructing an articulation is an iterative process and after the expert is satisfied with the rules generated, they are stored and used when information needs to be composed from the two ontologies.

In order to keep the cost of computation and especially maintenance (which often dominates other costs in established business environments) low, we strive to make the articulations minimal. Currently, the onus is on the expert to keep the articulation minimal. In future, we hope to make the automated heuristics aware of the needs of the application and minimize the articulations.

The matching algorithms that we use can be classified into two types - iterative and non-iterative.

Non-iterative Algorithms

Non-iterative algorithms are ones that generate the concepts that match in the two ontologies in one pass. These algorithms do not generate any new matches based on existing matches. The non-iterative algorithms that we employ involve matching the nodes based on their content.

The articulation generator looks at the words that appear in the label of the two nodes (or associated with the two nodes, e.g., if the nodes are documents or if more elaborate descriptions of the concepts that are represented using the nodes are available) that it seeks to match and generates a measure of the similarity of the nodes depending upon the similarity of the words used in their descriptions or labels.

The non-iterative methods that we currently use primarily refer to dictionaries and the Nexus [16] and also use several semantic indexing techniques based on the context of occurrence of words in a corpus. Since the articulation generator is modular in nature, it should be easy to add any other sophisticated heuristic (like consulting WordNet [17]) that allows us to generate semantic similarity measures between phrases.

Iterative Algorithms

Iterative algorithms require multiple iterations over the two source ontologies in order to generate semantic matches between them. These algorithms look for structural isomorphism between subgraphs of the ontologies, or use the rules available with the ontologies and any seed rules provided by an expert to generate matches between the ontologies. Iterative algorithms are typically used after the non-iterative algorithms have already generated some semantic matches between the ontologies and use these generated matches as its base.

For example, one heuristic we use is to look at the attributes of each node and see if the attributes of the two nodes have matched. If a reasonably large number of attributes are the same, the two nodes are related. If all the attributes of one node are also attributes of another node, the articulation generator indicates that the second node is a subclass of the first node. Another heuristic matches nodes based on the matches between their parent (or child) nodes. The expert has the final decision whether to bless this educated guess generated by the articulation generator.

Due to space limitations, we will not describe in detail all the heuristic algorithms that we use to match ontologies, but refer the interested reader to [18].

In the next section, we will briefly define an Ontology Algebra, which allows us to systematically compose information from diverse information sources. Since we focus on small, well-maintained ontologies in order to achieve high-precision, but we still want to serve substantial applications, we will often have to combine results of prior articulations. The ontology algebra provides the compositional capability, and thus enhances the scalability of our approach.

4 Ontology Algebra

When we compose information from multiple information sources it is important to do so in a principled fashion, especially when the number of such sources is large. The key to scalability is the systematic and effective composition of information.

In this section, we present an algebra that allows us to compose information to any level. By retaining a log of the articulation and subsequent composition process, we can also, with minimal adaptations, replay the composition whenever any of the sources change[16]. Without such a capability, integrated ontologies soon became stale and useless. Redoing a substantial integration manually is rarely done, because of the cost, and the realization that the work will be obsolete again in a short time.

The algebra has one unary operator: Select, and three binary operations:Intersection, Union, and Difference. The unary operator allows us to highlight and select portions of an ontology that are relevant to the task at hand. Given an ontology and a node, the select operator selects the subtree rooted at the node. Given an ontology and a set of nodes, the select operator selects only those edges in the ontology that connect the nodes in the given set.

Each binary operator takes as operands two ontologies that we want to articulate, and generates an ontology as a result, using the articulation rules. The articulation rules are generated by an articulation generation function briefly discussed above.

4.1 Intersection

Intersection is the most important and interesting binary operation. The intersection of two ontologies $O1 = (N1, E1, R1)$, and $O2 = (N2, E2, R2)$ with respect to the set of articulation rule generating f unction AR is:
$OI_{1,2} = O1 \cap_{AR} O2$, where $OI_{1,2} = (NI, EI, RI)$,
$NI = Nodes(AR(O1, O2))$,
$EI = Edges(E1, NI \cap N1) + Edges(E2, NI \cap N2) + Edges(Arules(O1, O2))$,
and $RI = Rules(O1, NI \cap N1) + Rules(O2, NI \cap N2) + AR(O1, O2) - Edges(AR(O1, O2))$.
The nodes in the intersection ontology are those nodes that appear in the articulation rules. The edges in the intersection ontology are the edges among the nodes in the intersection ontology that were either present in the source ontologies or have been established as an articulation rule. The rules in the intersection ontology are the articulation rules that have not already been modeled as edges and those rules present in the source ontology that use only concepts that occur in the intersection ontology.

The articulation rules are of two types - ones that are simple statements expressing binary relationships and the more complex rules expressed in Horn Clauses that are mostly supplied by the expert. An example of rules of the former type is: $(O1.CarSubclassOfO2.Vehicle)$ and one of the more complex logic-based ones is a conjunctive rule of the form: e.g. con-

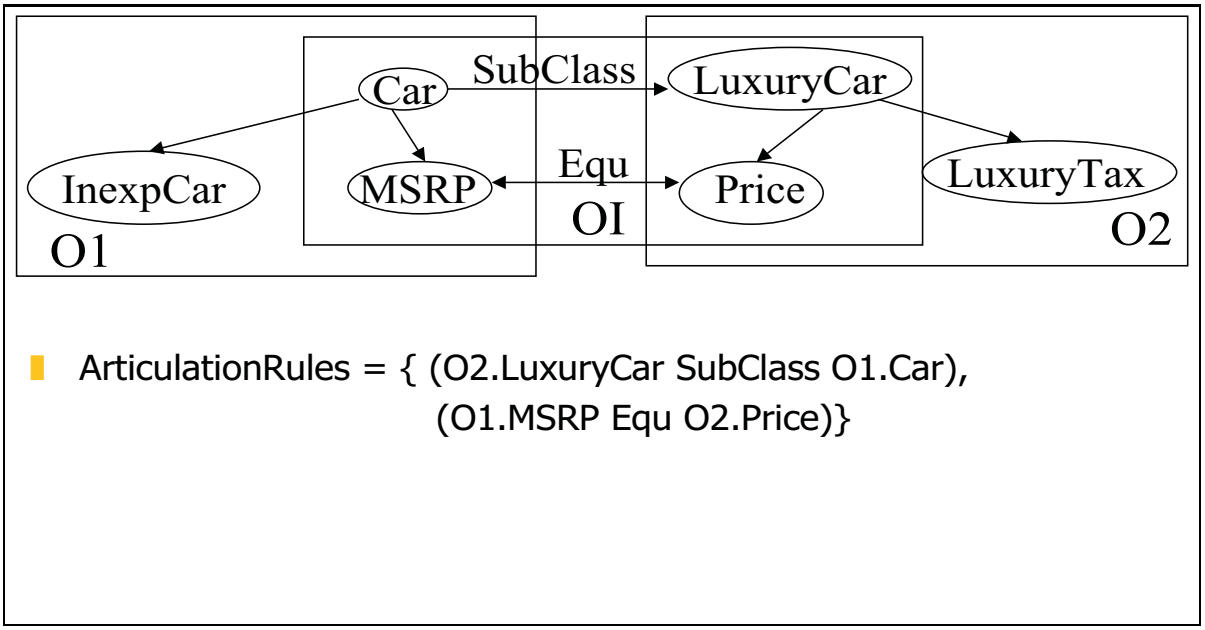


Figure 1: The Intersection Ontology OI of Source Ontologies $O1$ and $O2$

junctive rules of the form $(O1.XInstanceOfO1.Car), (YPriceOfX), (Y > 30000) \Rightarrow (O1.XSubClassOfO2.LuxuryCar)$. The former set of rules are modeled as edges in the articulation ontology and the second set of rules which require some form of reasoning to derive statements from are left as rules belonging to the articulation ontology. These rules will be processed during the query evaluation process only when necessary.

For all articulation generator functions, we require that $O1 \cap_{AR} O1 = O1$, that is the articulation generator function should generate such articulation rules that upholds the above-mentioned property as a sanity-check. Articulation generator functions that do not satisfy the above equality are *unsound* and for the purposes of our compositions, we do not use any unsound articualtion generator function.

In Figure 1, we show two ontologies $O1$, $O2$, the articulation rules between them and the intersection ontology OI. Equ is a short-hand that we use when to indicate classes that are equivalent in the two ontologies.

Note that since we consider each node as an object instead of the subtree rooted at the node, we will get only the node in the intersection by virtue of its appearing in an articulation rule and not automatically include its attributes or subclasses. Again, a minimal linkage serves our needs better than inclusion of possibly irrelevant concepts. Inclusion of attributes will be required to define subclass relationships among nodes in the source ontologies precisely.

Each node in the intersection has a label which contains the URI of the source in which it appears. If the attributes of the object that it represents are required, the application's query processor has to get that information from the original source. Defining the intersection with a minimal outlook reduces the complexity of the composition task, and the maintenance costs, which all depend upon the size of the articulation.

4.2 Union

The union OU between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ is expressed as $OU = O1 \cup_{AR} O2 = (VU, EU, RU)$ where
$VU = V1 \cup V2 \cup VI_{1,2}$,
$EU = E1 \cup E2 \cup EI_{1,2}$,
and $RU = R1 \cup R2 \cup RU_{1,2}$,
and where $OI_{1,2} = O1 \cap_{AR} O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies.
The union operation combines two source ontologies retaining only one copy of the concepts in the intersection. Though queries are often posed over the union of several information sources, we expect this operation to be rarely applied to entire source ontologies. The union of two source ontologies is seldom materialized, since our objective is not to integrate source ontologies but to create minimal articulations and interoperate based on them. However, we do expect that larger applications will often have to combine multiple articulations and here is where the union operation is handy.

4.3 Difference

The difference between two ontologies $O1$ and $O2$, written as $O1 - O2$, includes portions of the first ontology that are not common to the second ontology. The difference can hence be rewritten as $O1 - (O1 \cap_{AR} O2)$. The nodes, edges and rules that are not in the intersection ontology but are present in the first ontology comprise the difference.

One of the objectives of computing the difference is to optimize the maintenance of articulation rules. An articulation might need to be updated when one of the source ontologies that it articulates is changed. A change in the source ontology is to be forwarded to the articulation engine.

The articulation engine then checks if the changes are confined to the difference between the ontology and the other ontologies that it has been articulated with. If the change happens to be in the difference, then it does not occur in the intersection and is not related to any of the articulation rules that establish semantic bridges between ontologies. Therefore, the articulation rules do not need to be changed. If the changes to a source ontology, instead, is not in the difference, the articulation in which it occurs needs to be updated to reflect the change in the source ontology.

Using a formal process minimizes the maintenance costs in two ways: first of all we can recognize when a change in a source does not require a change in the articulation rules, and if a change is required we can rapidly regenerate the affected articulations, and adapt them to the new situation.

5 Conclusion

In this paper we present a brief overview of the ONION system used for the interoperation of information sources. ONION uses a simple conceptual model to which different ontology models are mapped using wrappers. The articulation generator is then applied to ontologies expressed using the sc ONION conceptual model to generate semantic correspondences leading to articulation rules among concepts in the source ontologies. A domain expert vali-

dates the generated rules or supplies new rules. These rules form the basis of interoperation among the autonomously maintained information sources. Finally, we briefly highlighted an ontology algebra that provides the formal basis for composition of information and the maintenance of the articulations. The ONION approach supports precise composition of information from multiple diverse sources by not relying on simple lexical matches, but requiring human-validated articulation rules among such sources. This approach allows the reliable exploitation of information sources that are autonomously maintained without any imposition on the sources themselves. The algebra based on the articulation rules allows systematic, composition, which unlike integration is much more scalable. When sources change maintenance is rapid since the effect of the changes can be determined using the algebra and the composition can be regenerated where needed.

References

[1] Cia factbook: http://www.cia.gov/cia/publications/factbook/. 2000.

[2] O. Ritter, P. Kocab, M. Senger, D. Wolf, and S. Suhai. Prototype implementation of the integrated genomic database. *Computers and Biomedical Research*, 27:97–115, 1994.

[3] Diane E. Oliver. *Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary*. PhD thesis, Stanford University, 2000.

[4] Information integration using infomaster, http://infomaster.stanford.edu/infomaster-info.html.

[5] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.

[6] Peter D. Karp. A strategy for database interoperation. *Journal of Computational Biology*, 2(4):573–583, 1996.

[7] Michael D. Siegel Cheng Hian Goh, Stuart E. Madnick. Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html.

[8] Cheng Hian Goh, Stéphane Bressan, Stuart Madnick, and Michael Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–270, 1999.

[9] Sergey Melnik. Declarative mediation in distributed systems. In *Proceedings of the International Conference on Conceptual Modeling (ER'00)*, 2000.

[10] Unified modeling language: http://www.omg.org/technology/uml/index.htm. 2000.

[11] Daml+oil http://www.daml.org/2001/03/daml+oil-index. 2001.

[12] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proc. PODS*, pages 417–424, 1990.

[13] Resource description framework(rdf) model and syntax specification, w3c recommendation http://www.w3.org/tr/rec-rdf-syntax. 1999.

[14] P. Mitra. The onion rule language http://www-db.stanford.edu/ prasen9/rule_lang.pdf. Technical report, Infolab, Stanford University, May 2001.

[15] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. *To appear in Theoretical Computer Science http://osage.inria.fr/verso/PUBLI/allbykey.php?mytexte=abiteboul*, 2001.

[16] J. Jannink. *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*. PhD thesis, Stanford University, 2000.

[17] Wordnet - a lexical database for english. http://www.cogsci.princeton.edu/ wn/. Technical report, Princeton University.

[18] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

On the Integration of Topic Maps and RDF Data

Martin S. Lacher and Stefan Decker
Database Group
Stanford University
Stanford, CA 94306
{lacher,stefan}@db.stanford.edu

Abstract. Topic Maps and RDF are two independently developed paradigms and standards for the representation, interchange, and exploitation of model-based data on the web. Each paradigm has established its own user communities. Each of the standards allows data to be represented as a graph with nodes and labeled arcs which can be serialized in one or more XML- or SGML-based syntaxes. However, the two data models have significant conceptual differences. A central goal of both paradigms is to define an interchangeable format for the exchange of knowledge on the Web. In order to prevent a partition of the Web into collections of incompatible resources, it is reasonable to seek ways for integration of Topic Maps with RDF. A first step is made by representing Topic Map information as RDF information and thus allowing Topic Map information to be queried by an RDF-aware infrastructure. To achieve this goal, we map a Topic Map graph model to the RDF graph model. All information from the Topic Map is preserved, such that the mapping is reversible. The mapping is performed by modeling the graph features of a Topic Map graph model with an RDF graph. The result of the mapping is an RDF-based internal representation of Topic Maps data that can be queried as an RDF source by an RDF-aware query processor.

1 Introduction

Different Communities are currently working on the vision of a Semantic Web: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. In order to make this vision a reality for the Web, supporting standards, technologies and policies must be designed to enable machines to make more sense of the Web, with the result of making the Web more useful for humans. One issue for the Semantic Web is how to allow for inter-operable representations of data on the Web. RDF [1] and Topic Maps [2] are two independently developed standards, which can be used to represent data on the web in an inter-operable fashion. Both standards have established a large user community and will most likely be building blocks of the future Semantic Web. To prevent a partition of the Semantic Web into incompatible subsets, ways for inter-operation of overlapping standards like RDF and Topic Maps have to be found. By Interoperability, we mean for example that any Topic Map source of data can be queried with an RDF-aware query infrastructure and vice versa. Both directions are equally important, as both standards have their advantages and disadvantages and are equally likely to be used on the future Semantic Web. We chose to begin with the

approach of making Topic Map sources queriable for an RDF infrastructure. The reason for this is partly because the RDF community has established a query infrastructure (eg. [9]), which can be reused for querying Topic Map resources. The Topic Map community is in the process of standardizing a query language, commmercial packages already offer proprietary query languages [1]. Other approaches that make RDF sources available to Topic Map aware query infrastructure have been proposed and their relation to this work is presented in Section 6.

Our approach to integration of Topic Maps and RDF data fllows the layered approach to data interoperability proposed in [12]. This approach splits data models into different layers, much like the layers in a network protocol stack. This layered model is useful for understanding complex data model interoperation, since the integration problem complexity is broken into smaller problem parts. An introduction to the layered approach to data interoperability is given in Section 2. We make a Topic Map RDF-queriable by performing a mapping between the two data models on a layer, on which in both of the models, data is represented as a graph. Thus, in fact, our mapping is a mapping between two types of graphs. The mapping is performed by modeling the Topic Map graph with an RDF graph. On top of the graph layer, there may be additional semantics, which we do not consider in this paper. For example, the graph may be used to represent UML data, DAML+OIL data or Topic Map data. Figure 1 shows an overview of the architecture that we have in mind for the integration of different sources.

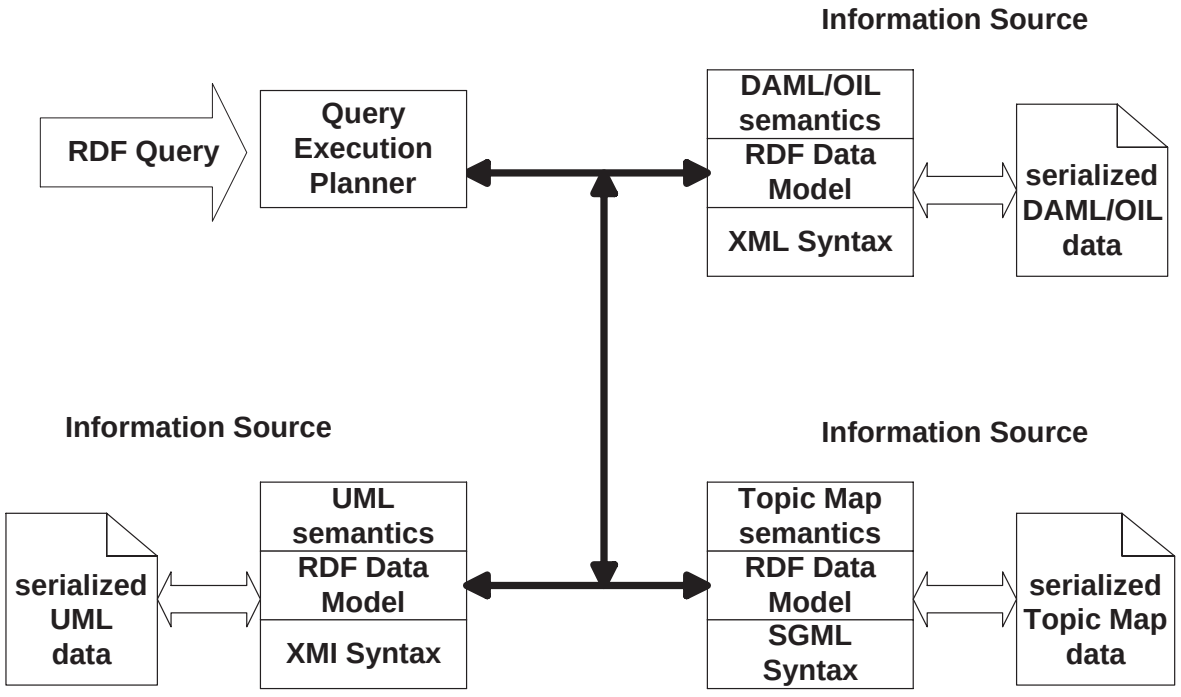


Figure 1: Overview of the integration of different data sources.

Each of the data sources in Figure 1 stores persistent data according to a certain serialization syntax. From each of these persistent data, a memory data model based on RDF as a low-level object model can be built. This RDF model in all information resources can then be queried by

[1]See http://k42.empolis.co.uk/tmql.html

an RDF-aware query infrastructure. This way, information sources with different model-based data representations can be integrated.

The remainder of this paper is organized as follows: We will first introduce the data models of RDF and Topic Maps with respect to the layered interoperability approach. General familiarity with RDF and Topic Maps is assumed. Thereafter, in Section 3, we will present our integration approach in more detail including two small exemplary mappings. Section 4 presents a real world application example for the joint querying of a Topic Map information source and an RDF information source. Section 5 shortly describes the implementation of our mapping approach. Section 6 gives a brief overview of related work. Finally, in Section 7 we summarize our contributions.

2 Overview of the data models

In this section, we will give a brief overview of the RDF and Topic Map data models with respect to the layered model introduced in [12].

2.1 The layered interoperability model

The layered model of data interoperability in [12] breaks up the problem of data model integration into a stack of layers which are quasi-independent from each other. This approach resembles the ISO protocol stack for network interoperation. The different layers presented are from bottom to top, the syntax layer, the object layer and the semantic layer. Each of those layers actually has sublayers, but we do not require such a detailed perspective on the layers here. The syntax layer is concerned with a serialization syntax for persistent storage of data. The object layer is concerned with how to assign identity to objects or how binary relations are represented. The semantic layer is concerned with the interpretation of the objects and their relationships.

We will not present details on each of the layers and their involvement in the mapping. The important essence is, that our approach works by performing a graph transformation on the object layer, which can be performed quasi-independently from the other layers. This independence is possible, because any semi-structured data model [15] can be represented as a directed graph, which is also the data model of RDF. Thus, any kind of semi-structured data model can be represented by RDF on the object layer. How the RDF graph is interpreted on a higher level can differ again for different data models. In this paper we will not consider the issue of mapping those higher level semantics. We will only look at RDF as the common denominator for data representation and query purposes. The Topic Map semantic on a higher level will thus be conserved with our mapping and only the representation on the object layer will be mapped to RDF.

2.2 RDF

The Resource Description Framework Model and Syntax Specification [1], which became a World Wide Web Consortium (W3C) Recommendation in February 1999, defines the RDF data model and a basic serialization syntax. The RDF Data model is essentially a directed, labeled graph: it consists of entities, identified by unique identifiers, and binary relationships between those entities. In RDF, a binary relationship between two specific

entities is represented by a statement (or triple). An RDF statement can be represented in a graph as two nodes and a directed arc between the nodes. The node with the outgoing arc is called subject of the statement, the arc is called property and the node with the incoming arc is called object of the statement. The RDF data model distinguishes between resources, which have URI identifiers, and literals, which are just strings. The subject and the predicate of a statement are always resources, while the object can be a resource or a literal.

Taking the perspective of the layered interoperability model, RDF has several possible syntaxes on the syntax layer, among which there is one basic and on abbreviated syntax defined in [1]. On the object layer, the RDF model is a directed graph, as described above. The semantic layer of RDF is minimal. Together with additional languages like DAML+OIL, more more complex semantics can be expressed with RDF.

2.3 Topic Maps

Topic Maps [2] have been standardized in 1999. A Topic Map is defined as a collection of Topic Map documents, which adhere to a certain SGML syntax defined in the standard document. The SGML Syntax of those documents is described in the standard along with an informative conceptual model for memory representation of Topic Maps. Topic Maps can be used as a format for the representation of multi-dimensional subject-based indices for document collections. Topic Maps can also be used as a format for interoperable knowledge representation.

The original ISO standard specified an SGML syntax for the exchange of Topic Maps. To make Topic Maps applicable on the Web, the XML Topic Maps standard has been drafted [3]. XTM defines an XML syntax for Topic Maps and gives a specific, albeit slightly simplified, data model of a Topic Map. Both the SGML syntax and the XML syntax incorporate syntax shortcuts for complex data model constructs. The XML syntax presented in [3] has been appended to [2] after publication.

Several representations on the object layer have been proposed for Topic Maps. We will adhere to the graph representation described in [6]. This graph representation knows four different kinds of arcs and three different kinds of arcs. The nodes do not differ in their properties, but in which arcs they can be connected to. Additionally, each node can have several *subject identity points*. Subject identity points serve partly serve as unique identifiers for the nodes.

The semantics of the graph nodes defined on the object layer is that of subjects, defined as anything that can be referred to in human discourse. These subjects are divided into *topics*, *associations* and *scopes*, corresponding to the three different node types. Topics can have a number of characteristics, which can be bound to them by means of associations. The processing models described in [7] and [6] state some semantic constraints on the graph, which have to be enforced in order to produce a *consistent* Topic Map. Basically, these constraints ensure that no duplicate topics occur in a consistent Topic Map.

3 Integration Approach

Our general approach is that we model a graph representation of a Topic Map with the means that an RDF graph gives us. This is an approach that has been termed "modeling the model" in [13]. In this approach, all information from the source model is preserved and just represented

in another format. Thus, this transformation can also be seen as a syntax transformation. We picked this approach because it has an advantage over an approach that would perform a semantic mapping between representations. A semantic mapping will most likely incur loss of information and thus make an inverse mapping impossible. The semantic mapping approach is called “mapping the model”in [13].

3.1 Semi-structured data

Our integration goal is to generate a memory internal representation of a Topic Map, which can be queried with an RDF query infrastructure. This means that the surface syntax of the two data models is not of interest for our task. Thus, our approach is applicable for both the SGML syntax as well as the XML syntax. However, our implementation only considers the XML (XTM) syntax. We implemented the processing model proposed in [6] to construct a Topic Map graph model from an XTM document.

RDF is closely related to the concept of semi-structured data, identified in the database community [11], [15] as a means for data integration [10] [14] and transformation [4]. Any kind of data that can be represented as a graph is called semi-structured data. Thus, if heterogeneous data sources are transformed into a graph representation in some standard representation format, all this data can be queried with the same query infrastructure in the same query. This makes joint queries over multiple data sources possible.

RDF can be used to represent semi-structured data as a graph. This also applies to Topic Maps data , since there is a graph representation defined for Topic Maps [6]. Topic Maps have the expressive power of a schema language and can be used to represent ontologies. An RDF adapter for Topic Maps makes a Topic Map information source RDF queriable.

We will now describe the different aspects of the representation of Topic Maps as RDF with respect to the layered data model described in [12].

3.2 Object layer

The representation of Topic Maps as RDF is a graph transformation on the object layer of the layered data model. The object layer describes how object identity is established and how binary relationships are described in a certain data model.

The RDF Model and Syntax description gives a graph model for RDF. The Topic Maps standard does not enforce a certain internal representation for a Topic Map. Instead, several processing models have been proposed, which describe how to deserialize an abbreviated syntax into a consistent graph-based internal data structure [7], [6]. We use the graph model presented in [6]. This graph model has the characteristics described in Section 2. Our goal is to map the Topic Map graph representation onto an RDF graph representation without any loss of information. We do this by mapping each element of the Topic Map graph described in [6] to a corresponding construct in RDF.

A prerequisite for the mapping is, that the Topic Map graph is consistent, i.e. there are no redundant elements in the Topic Map graph [6]. A Topic Map graph $tm = (N, A, S)$ consists of a set of nodes N, which have the three types a, t and s, a set of arcs A, which have the different types $associationMember$, $associationScope$, $associationTemplate$ and $scopeComponent$ and a set of resources S which indicate or constitute a subject. The ***associationMember*** arc has a t-node attached as a role label. Each node has at most one subject

constituting resource and any number of subject indicating resources attached to it. The connection with these resources is not part of the graph [6], but taken care of in the implementation domain. However, for a mapping without loss of information, we need to consider those resources as well.

An RDF Model graph $r = (R, L, ST)$ consists of a set of resources R, a set of literals L and a set of statements ST. Our mapping m maps the set of all consistent Topic Maps TM to the set of all RDF Models R. The set of Nodes N is mapped to the set of resources R in RDF. The set of arcs A is mapped to the set of statements ST in RDF. The set of subject indicating/constituting resources S is also mapped to the set of resources R in RDF.We map the Topic Map graph to an RDF graph by first mapping the graph nodes and then mapping the arcs.

Each node in the Topic Map graph is mapped to a resource in the RDF model. The ID of the RDF resource is the ID of one of the subject identity points of the Topic Map node. If there is no subject identity point for the node, an ID is generated. For the rest of the subject identity points, statements are generated, which connect the subject identity points to its node in the RDF graph. An RDF statement is generated, which identifies the type of node that has been mapped. The Topic Map graph model knows three different kinds of nodes. We make use of the namespace capability of RDF to define the three types of nodes available in Topic Maps. The node types are defined as shown in Figure 4. An exemplary mapping of a Topic Map node to an RDF graph is shown in figure 2.

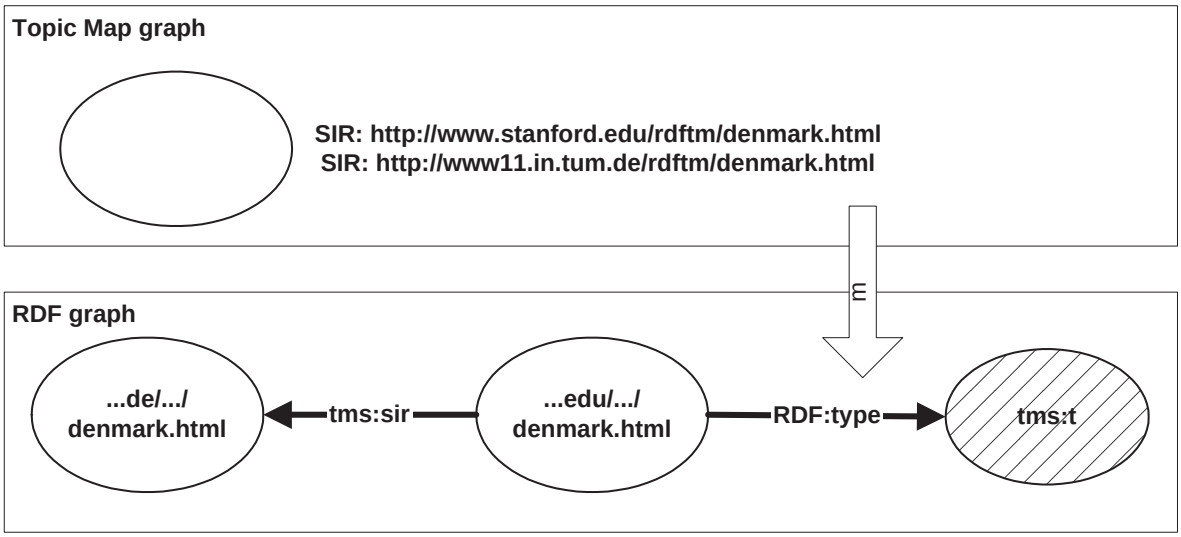


Figure 2: Exemplary mapping of a Topic Map node to an RDF graph

After all the nodes have been mapped, we map the arcs in the Topic Map graph to statements in the RDF graph. For each arc between two nodes n_1 and n_2 we generate an RDF statement. The property of the statement corresponds to the arc type in the Topic Map graph. The corresponding properties are defined in the schema in Figure 4. Although arcs in the Topic Map graph are not explicitly directed, they have an implicit directionality given through the node types at each arc end. Thus, the RDF graph is not more constrained than the Topic Map graph in that respect. If the mapped arc is an $associationMember$ arc, it has a role label in the Topic Map graph. To represent this in the RDF graph, we reify the RDF statement signifying this arc and bind the role label node to this statement with the $roleLabel$

property defined in Figure 4. The mapping of an $associationMember$ arc between two nodes is shown in Figure 3.

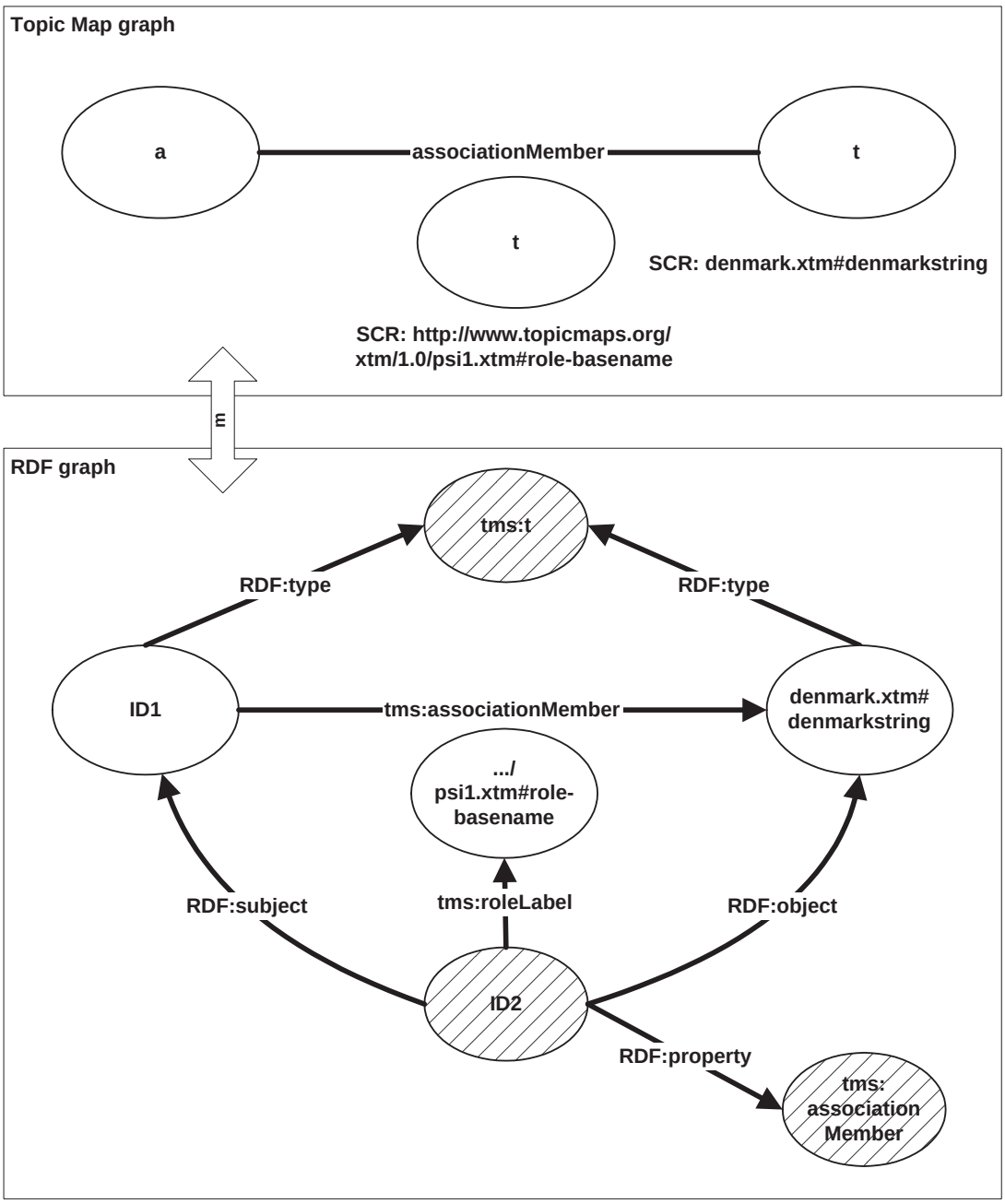


Figure 3: Exemplary mapping of a Topic Map associationMember arc to an RDF graph

3.3 Semantic layer

RDF can be the basis for an ontology definition language and Topic Maps can be seen as an ontology definition language. RDF requires additional vocabulary such as DAML+OIL for ontology definition and RDF itself merely provides the object layer in this data model stack. Topic Maps on the other side have richer semantics, and provide a number of features of an

ontology definition language. For a comparison on the semantic layer, DAML+OIL based on RDF is a more appropriate candidate for a comparison with Topic Maps. However, this will not be investigated in this paper.

4 Application Example

We will now present an example for the integration of two heterogeneous information sources for querying. The first source is a Topic Map serialized in XTM [3] based on the CIA World Fact Book. The second source is the Open Directory[2], which is represented in RDF. We would like to find travel information for countries which have petroleum as a natural resource. Countries with petroleum as a natural resource can be found in the CIA World Fact Book and travel information can be found in the Open Directory collection of web pages. First, we will present how a part of the Topic Map source is mapped to RDF. Then, we will show how the two information source can be jointly queried.

4.1 Mapping the Topic Map source to RDF

As a first preparatory step for our integration approach, we defined an RDF Schema which defines the node and arc types of a Topic Map graph. Figure 4 shows the RDF schema definition.

```
<rdf:RDF xmlns:rdf="http://www.w3c.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdf="http://www.w3c.org/2000/01/rdf-schema#"
  xmlns:tms="http://www-db.stanford.edu/rdftmmapping/tm-schema#"
  xmlns="http://www-db.stanford.edu/rdftmmapping/tm-schema#">
  <rdfs:Class ID="t"/>
  <rdfs:Class ID="a"/>
  <rdfs:Class ID="s"/>
  <rdf:Property ID="associationMember"/>
  <rdf:Property ID="associationScope"/>
  <rdf:Property ID="associationTemplate"/>
  <rdf:Property ID="scopeComponent"/>
  <rdf:Property ID="roleLabel">
  <rdf:Property ID=="sir">
</rdf:RDF>
```

Figure 4: The RDF Schema for an RDF-based Topic Map

For the actual construction of an RDF representation of a Topic Map graph, the next step is the generation of a graph representation from a (XTM) Topic Map document. For this purpose we implemented an API for Topic Maps which exposes a graph-based data structure and allows us to directly operate on the Topic Map constructs for the graph construction. The API also conforms with the processing model presented in [6], which is required to generate a valid Topic Map graph from an abbreviated syntax. Figure 5 shows a short snippet of a Topic Map with information from the CIA World Fact Book in the form of an XTM document. Processing this XTM document results in the graph shown in Figure 4.1.

After processing the XTM document snippet according to the processing model, the generated graph for this short XTM document snippet looks like this:

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

```
<topic id="denmark">
  <basename>
    <baseNameString>Denmark</baseNameString>
  </basename>
</topic>
<association id="denmark-has-petroleum">
  <member>
    <roleSpec>
      <topicRef xlink:href="#country"/>
    </roleSpec>
    <topicRef xlink:href="#denmark"/>
  </member>
  <member>
    <roleSpec>
      <topicRef xlink:href="\#natural-resource">
    </roleSpec>
    <topicRef xlink:href="petroleum">
  </member>
</association>
<topic id="country"/>
<topic id="natural-resource"/>
```

Figure 5: XTM document subpart

Figure 4.1 shows the Topic Map graph that is generated according to the XTM processing model. The ellipses represent nodes, the lines represent arcs with different types. The role labels for association member arcs are connected to the arcs via another arc, the role label arc. The graph that is induced by the XTM snippet above basically represents a topic node that represents the subject Denmark. The graph also represents the fact that Denmark has petroleum as a natural resource. It also shows that the base name "Denmark" has been assigned to the Denmark topic.

We will now represent this graph as an RDF graph. In fact, the transformation of the graph is performed during the construction of the Topic Map graph according to the transformation guidelines presented above. To construct the graph, we generate RDF triples. Figure 7 shows the mapped RDF graph.

It can be seen in Figure 7 that the graph can be translated in a straightforward manner. The RDF graph has additional type edges to signify the node types. All nodes in the graph which have no type edges are assumed to be of type topic in this graph. As IDs of each of the nodes we used the ID of either the respective XTM element, or generated an ID. The additional role topics, which are attached to the association member edges in the Topic Map graph, are modeled by reification of a statement in RDF: The statement that signifies the association member edge from a topic to an association is reified and becomes the subject in another statement that has the role topic as an object and the RDF-Schema-defined roleLabel as its property.

Although the mapping transforms undirected arcs into directed arcs, the mapping between the two graph representations is still a bijective mapping. The direction of arcs in the Topic Maps graph model is implicit. For querying purposes, arcs in the RDF graph of a Topic Map have to be queried in two directions.

By translating all graph constructs mentioned in the XTM processing model to an RDF

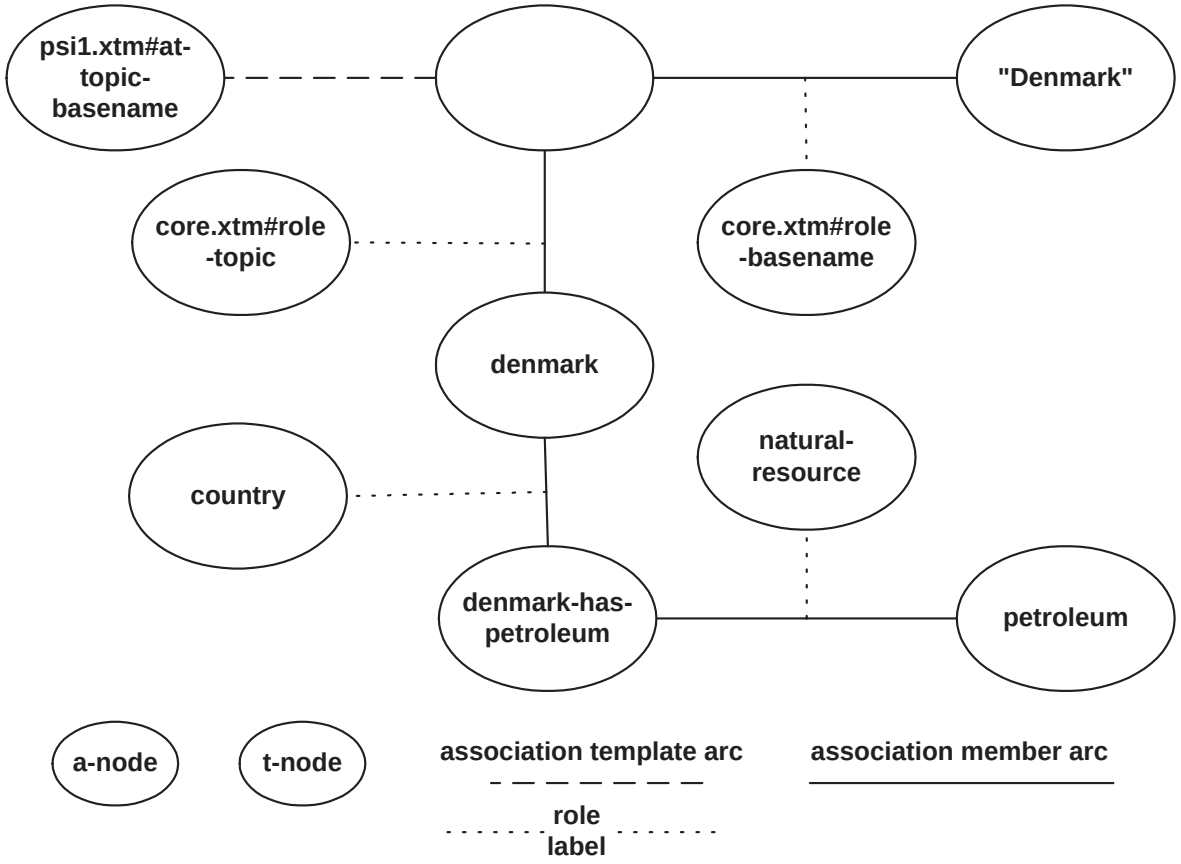


Figure 6: The generated Topic Map graph

graph we essentially generated an RDF representation of a Topic Map. We can now query this RDF graph with an RDF query language. An example for the utility of this will now be shown.

4.2 Joint querying of the information sources

As an example for the usefulness of our integration approach, consider the following scenario: We would like to find Web pages about travel in countries, which exploit petroleum as a natural resource. The available resources include a Topic Map constructed from the CIA world fact book [3], which includes general resources about countries, but no Web pages about travel. To retrieve the requested travel pages, we access the Open Directory collection of Web pages. The Open Directory is a large Web page directory constructed in a collaborative way by a large number of expert volunteers. The directory structure of the Open Directory is represented in RDF. With our integration approach, a query processor can now query both information sources and integrate the results into one query result. The distributed and heterogeneous nature of the information sources remains transparent to the user.

For our query example, we will assume the existence of a query engine which can query distributed information resources that are represented in RDF. The basis for such a query engine can be the query infrastructure and F-Logic syntax presented in [9]. The extension that

[3] http://www.cia.gov/cia/publications/factbook/

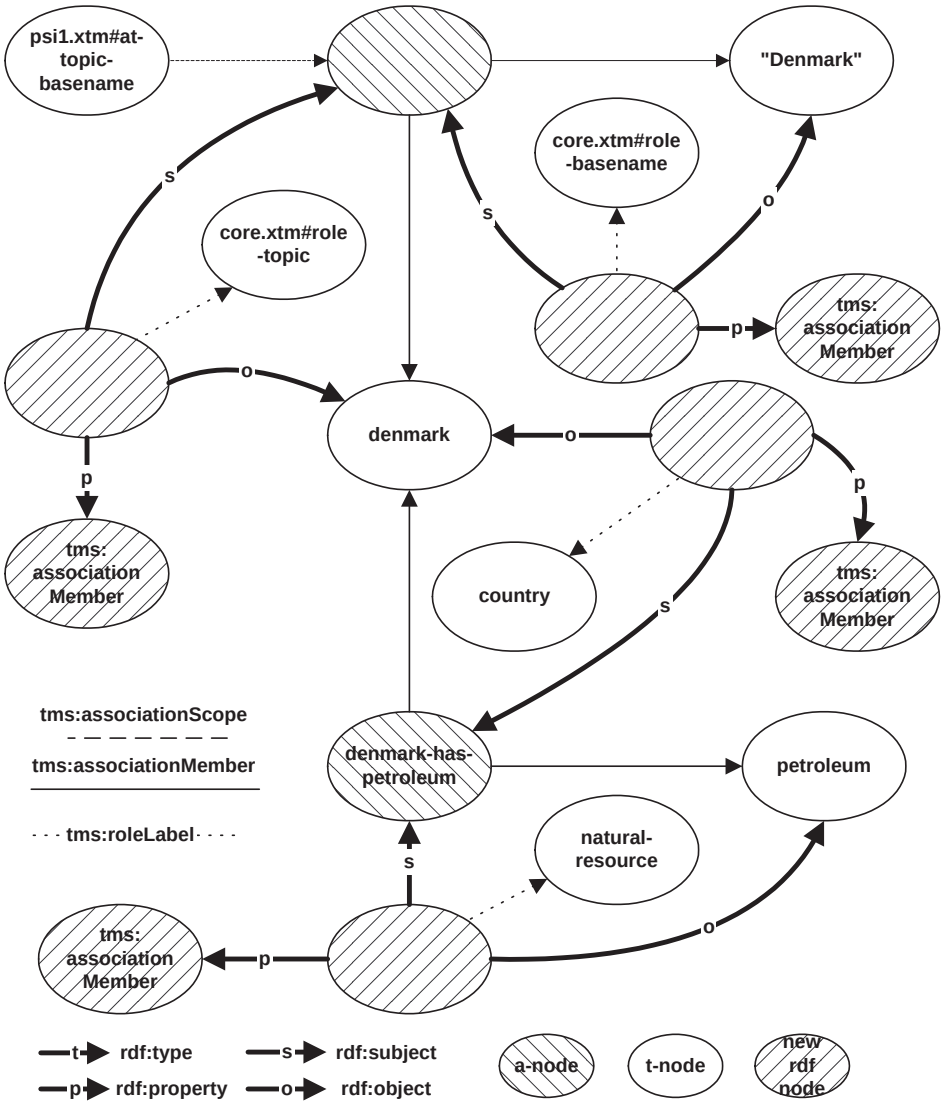


Figure 7: The generated RDF Topic Map graph

would have to be performed is to alllow to specify information sources to which certain parts of the query have to be directed. Figure 8 shows an example of a query in F-Logic syntax, as introduced in [9]. The query uses the assumed capability of the query engine to specify source with the @ character. The query in Figure 8 can now be posed to query the two above mentioned information sources.

This example query assumes the existence of a name mapping, which resolves the naming differences between resources (`mapsTo` property). Please note also that the query in Figure 8 is simplified in that naming conventions of DMOZ are not considered here. Also, the `Travel_and_Tourism` property will have to be constructed from the `DMOZCountry` URI.

The query answers queries over two different sources: the CIA World Factbook and the DMOZ Open Directory. The structure of the query language mimics RDF and is `subject [predicate->object] source`. The first part of the query retrieves all countries, which have petroleum as a natural resource. This part of the query can be answered from the CIA

```
FORALL pages <- Country, DMOZCountry, X, Y, Z
  Y[tms:roleLabel->country;
    rdf:object->Country
  ]@CIA_WORLD_FACTBOOK
  and
  X[tms:roleLabel->natural-resource;
    rdf:object->petroleum;
    rdf:subject->
      Z[tms:associationMember->Country
      ]@CIA_WORLD_FACTBOOK
  ]@CIA_WORLD_FACTBOOK
  and
    Country[mapsTo->DMOZCountry]
  and
    DMOZCountry[Travel_and_Tourism ->
      dmozpage[links->pages]
    ]@DMOZ.
```

Figure 8: Query in F-Logic Syntax over DMOZ and RDF-based Topic Map

World Factbook Topic Map, in the RDF representation given above. Now we are able to query the DMOZ data for travel information on this country. The result of the query is a list of web pages from DMOZ categories like *Top/ Regional/ Europe/ Denmark/ Travel*, etc.

It can be seen that by representing a Topic Map in RDF, the information source becomes queriable with an RDF query language. But the actual query also requires a query processor, which can handle the distributed sources.

5 Implementation

The implementation of our RDF adapter for Topic Maps can handle the XTM syntax of Topic Maps. Both [2] and [3] constrain their normative part of the standard on the specification of an exchange syntax for Topic Maps. In order to represent a Topic Map with RDF, a graph model has to be constructed from a Topic Map document. Our implementation considers the XTM syntax and constructs a graph representation according to the processing model presented in [6]. The construction of the graph model is performed through a graph-based API proposed in [5]. The implementation of this API simplifies the realization of the processing model, since the underlying data model is the same for both. Along with the creation of the API objects, an equivalent set of RDF triples is generated.

For parsing the XTM document we use a SAX-based parser, which feeds events to our implementation of the processing model, which then constructs the RDF graph. After constructing the graph, the redundancy rules are enforced.

6 Related Work

In [8], a general approach to integration of heterogeneous model-based information has been presented. It is shown that in principle all model based information can be represented by an RDF based meta-model. It is shown that this also includes Topic Maps. However, the authors do not go into details about this specific mapping.

In [13] two general approaches to the integration have been proposed. The first approach shows how Topic Maps can be modeled with RDF vocabulary and vice versa. The second approach shows how a semantic mapping between the two standards can be performed. Semantic mappings bear the disadvantage that inherently, the transformation is lossy and the transformation is not bijective. The examples show the general approach of mapping Topic Maps to RDF.

Also, representing RDF data as Topic Map data is possible, but for the purpose of querying various sources through one query infrastructure, the inverse direction is the easier solution. RDF has the simpler data model, allowing more efficient and simpler storage and query facilities than Topic Maps. Pure syntax transformations have been proposed[4], but this approach disregards the need for a processing model to generate the Topic Map graph from the serialized syntax.

We have shown that from the point of view of an integrated Semantic Web it is desirable to be able to query a Topic Map source with an RDF query. This can be achieved if the Topic Map source itself represents its data as RDF data. The problem of integration of RDF and Topic Maps has been approached with little success so far. Most Integration approaches have lead to the conclusion that RDF is not expressive enough to represent Topic Maps. What we aim to achieve is not to convert a Topic Map document into a number of serialized RDF statements, which would render the document difficult to read. Instead we aim to generate an internal representation of a Topic Map, which is really a set of RDF statements. This way, a data source which stores Topic Map data can be queried as if it was an RDF source. Thus, what we need to achieve is a mapping of an internal Topic Map representation to an internal representation of a set of RDF statements.

7 Conclusion

Interoperability is of greatest importance for the future Semantic Web. We suggested a way to achieve interoperability between Topic Maps and RDF, which enables the joint querying of RDF and Topic Maps information sources. Our work builds on existing work on general approaches for the integration of model based information resources. In contrast to those general approaches we showed a detailed mapping specifically from XTM Topic Maps to RDF. We achieved this by adopting an internal graph representation for Topic Maps, which has been published as part of one of the processing models for Topic Maps. We perform a graph transformation to generate an RDF graph from the Topic Map graph representation. The Topic Map source can now be queried with an RDF query language together with RDF information sources. We see this as a first step towards the integration of the many heterogeneous information sources available on the Web today and in the future.

References

[1] Resource Description Framework (RDF) Model and Syntax Specification, Feb. 1999. W3C Recommendation.

[2] ISO/IEC 13250: Topic Maps, Dec. 1999. ISO/IEC FCD, April 1999.

[3] XML Topic Maps (XTM) 1.0, Mar. 2001. topicmaps.org Specification, http://www.topicmaps.org/xtm/1.0/.

[4] http://lists.w3.org/Archives/Public/www-rdf-interest/2001Mar/0062.html

[4] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and Translation for Heterogeneous Data. In *6th International Conference on Database Theory*, 1997.

[5] Khalil Ahmed. Developing a Topic Map Programming Model. In *Knowledge Technologies 2001*, Mar. 2001.

[6] Michel Biezunski and Steven R. Newcomb. topicmaps.net's Processing Model for XTM 1.0, version 1.0.1, May 2001. http://www.topicmaps.net/pmtm4.htm.

[7] Michel Biezunski and Steven R. Newcomb. Xml Topic Maps Processing Model 1.0, Mar 2001. http://www.topicmaps.org/xtm/1.0/xtmp1.html.

[8] Shawn Bowers and Lois Delcambre. Representing and Transforming Model-Based Information. In *International Workshop on the Semantic Web (SemWeb), in conjunction with ECDL 2000*, Sep. 2000.

[9] Stefan Decker, Dan Brickley, Janne Saarela, and Jürgen Angele. A Query and Inference Service for RDF. In *QL '98 - Query Languages Workshop*, Dec. 1998.

[10] Hector Garcia-Molina, Jan Hammer, K. Ireland, Y. Papakonstantinou, Jeff Ullman, and Jennifer Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *AAAI Symposium on Information Gathering*, pages 61–64, Mar. 1995.

[11] Jan Hammer, J. Mc Hugh, and Hector Garcia-Molina. Semistructured Data. In *First East-European Workshop on Advances in Databases and Information Systems - ADBIS '97*, Sep. 1997.

[12] Sergey Melnik and Stefan Decker. A Layered Approach to Information Modeling and Interoperability on the Web. In *ECDL '00 Workshop on the Semantic Web*, Sep. 2000.

[13] Graham D. Moore. RDF and Topic Maps - An Exercise in Convergence. In *XML Europe 2001*, May 2001.

[14] Y. Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE '95*, Mar. 1995.

[15] Dan Suciu. An Overview of Semi-Structured Data. *SIGACTN: SIGACT News*, 29:pp. 28–38, 1998.

An infrastructure for formally ensuring interoperability in a heterogeneous semantic web

Jérôme Euzenat
INRIA Rhône-Alpes
655 avenue de l'Europe, 38330 Montbonnot Saint-Martin (France)
Jerome.Euzenat@inrialpes.fr

Abstract. Because different applications and different communities require different features, the semantic web might have to face the heterogeneity of the languages for expressing knowledge. Yet, it will be necessary for many applications to use knowledge coming from different sources. In such a context, ensuring the correct understanding of imported knowledge on a semantic ground is very important. We present here an infrastructure based on the notions of transformations from one language to another and of properties satisfied by transformations. We show, in the particular context of semantic properties and description logics markup language, how it is possible (1) to define properties of transformations, (2) to express, in a form easily processed by machine, the proof of a property and (3) to construct by composition a proof of properties satisfied by compound transformations. All these functions are based on extensions of current web standard languages.

1. Introduction

The idea of a "semantic web" [Berners-Lee 2001] supplies the (informal) web as we know it with annotations expressed in a machine-processable form and linked together. Taking advantage of this semantic web will require the manipulation of knowledge representation formalisms.

There are several reasons why the semantic web could suffer from diversity and heterogeneity. One main reason is that it depends on content providers and content providers have diverse goals and focal points that will not lead them to invest on the same area of the semantic web. Yet these areas of interest will overlap meaningfully and putting part of their content together will be required for taking advantage of them in unexpected applications [Wiederhold 1999]. Another reason arises from the observation that the web sites and web pages are more often generated on demand depending on (1) the device on which they will be displayed and (2) the preferences of the users. There is no reason why the semantic web resources would not require the same kind of operations. There are several other reasons for expecting heterogeneity including legacy knowledge bases and systems, learning curves…

Because we think that nothing better can happen to the semantic web than having well suited languages for each task while preserving interoperability, we aim at providing a path toward this goal. This paper is a short description of the technicalities involved in a solution to interoperability despite diversity.

Imagine a second-hand hardware provider company willing to build a semantic web support for its business involving repair and printers. Because the company core competence is neither technical support, nor printers, it will prefer to reuse knowledge models (or ontologies, which can be quickly described as conceptual schemes of knowledge bases) from authoritative sources. Additionally, the company has decided to use a particular representation and deduction formalism for processing knowledge (similar to the SHIQ language for which the FaCT reasoner can perform subsumption test). This company will find a technical support ontology written in DAML-ONT and a printer ontology written in OIL that fulfill its requirements.

The problem then consist of importing these two ontologies in the SHIQ language in a semantically valid way (i.e. preserving the consequences they entail). The solution will resort to transforming each ontology into a common format and transforming this format into a form compatible with SHIQ. This can be achieved by a homemade transformation or by assembling transformations available through the web (see Figure 1). Of course, the transformation system engineer will choose transformations that satisfy the desired properties (consequence preservation). To that extent, (s)he will refer to the properties advertised for each transformation. But (s)he has to make sure that the assertions are correct (they can be erroneous, or valid within a specific context…). There are two basic alternatives to this problem: trusting or checking.

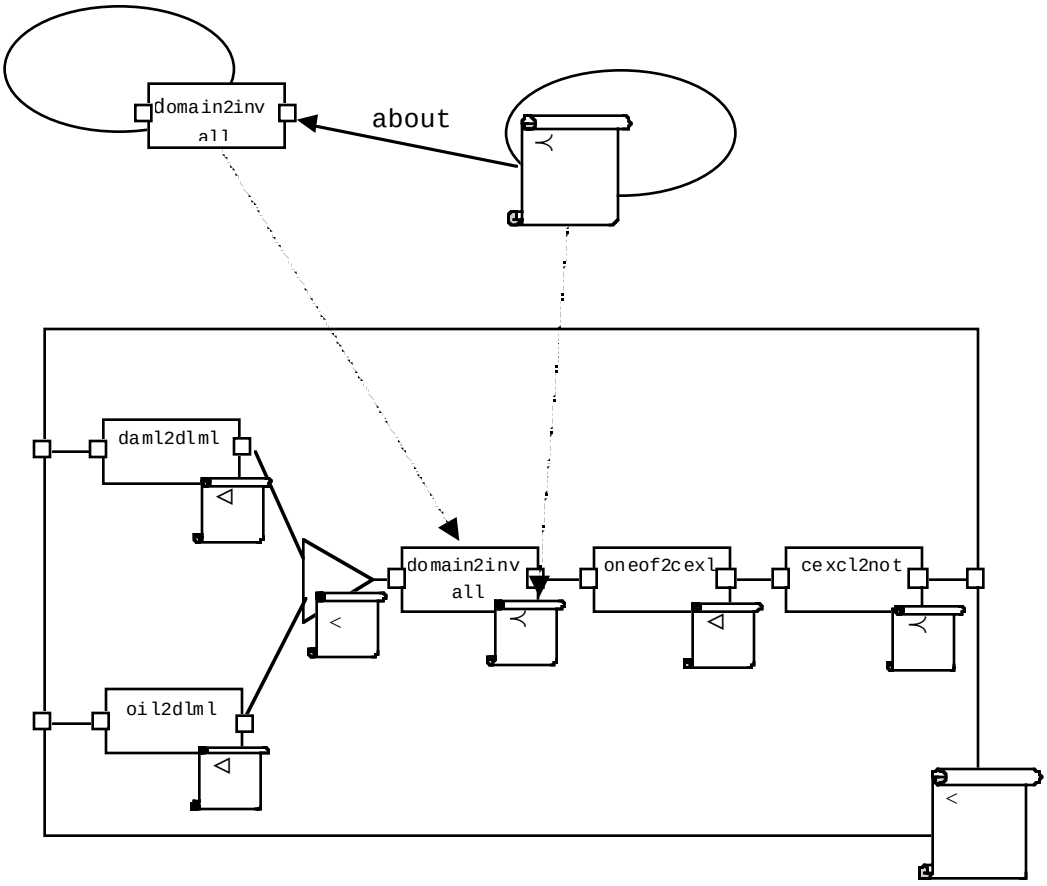


Figure 1 : The complete construction of a transformation, by composing more elementary transformations gathered from the web, and the proof of consequence preservation by composing lemmas.

Checking is possible if the proofs of the asserted properties are available somewhere. It will then be possible to check the properties satisfied by the transformations and to deduce those satisfied by the compound transformations. In order to contribute to the global web of transformations, the transformation system engineer will publish the new compound transformation and the proofs of its properties.

The framework presented here is distributed, modular, incremental (i.e. anyone can add a new transformation, a new assertion or a new proof at any time) and ensure a high level of safety. In these matters, it is fully adequate for the semantic web.

The remainder presents the building blocks of such an infrastructure. The presentation is based on the simple example above (the complete example has been implemented in DLML and XSLT). First, we describe DLML, an XML encoding of knowledge representation language and the kind of transformations that can be performed on these languages (§2). Then several consequence-preserving properties are introduced (§3). The proofs of such properties are expressed in such a way that machines can manipulate them (§4). Last, we

introduce an environment for building, checking, proving and publishing transformation and proofs by composition (§5).

2. A family of representation languages : DLML

In order to simplify the presentation and to facilitate the transformations, we will restrict ourselves to a set of languages that act as pivot languages between the actual representation languages used in the semantic web.

In this presentation, a language L will be a set of expressions. A representation (r) is a set of expressions in L. In this framework, a model of a set of assertions $r \subseteq L$, is an interpretation I satisfying all the assertions in r. An expression δ is said to be a consequence of a set of expression r if it is satisfied by all models of r (this is noted $r|=_L \delta$). A family of languages is a set $\boldsymbol{L}$ of languages that share constructors having the same interpretation in all the languages. A family can be structured such that a language $L \vee L'$, such that any formula of L or L' is a formula of $L \vee L'$, always exists. The "family of languages" approach [Euzenat 2001c] is an interesting case, because it enables a fast implementation of meaning-preserving transformations. Using a family of languages makes the representations easier to understand because the elements have the same meaning across languages. It will enable the fragmentation of these transformations into unit transformations and the precise characterization of the transformation properties.

A good example of a family of languages is the description logics for which an extensive language hierarchy has been defined [Donini 1994]. This presentation will focus on our "Description Logic Markup Language" (DLML) on which we have carried out experiments. DLML [Euzenat 2001d] is a modular system of document type descriptions (DTD) encoding the syntax of many description logics (§2.1). The actual system contains the description of more than 40 constructors and 25 logics. To DLML is associated a set of transformations (written in XSLT) enabling the conversion of a representation from one logic to another (§2.2).

Note that we do not put forth DLML as the standard language of the semantic web but rather as one of the many languages that can be used for transformation purposes. DLML is used here as a proof of concept. The general framework, however, will work with other languages.

2.1 Modular Encoding

Description logics allow the manipulation of two kinds of terms: concepts and roles. Below are one role description stating that the role `inktype` has for domain of application the `InkPrinter` concept and one concept term description stating that a `ColorInkPrinter` is an `InkPrinter` whose `inktype`(s) are all instances of the `ColorInkType` concept.

```
inktype ≤ (domain InkPrinter)
ColorInkPrinter ≤ (and InkPrinter (all inktype ColorInkType))
```

Term descriptions are built from sets of atomic concept (resp. role) names and term constructors. They are constrained by equations of the kind above where two terms are related by a formula constructor (here ≤). A terminology is a set of such equations.

Concept terms are interpreted as sets of individuals of the domain of interpretation and roles are sets of pairs of individuals. The interpretation I of the constructors above is :

$$
\begin{aligned}
I((\texttt{and } c_1, \ldots c_n)) &= I(c_1) \cap \ldots I(c_n) \\
I((\texttt{all } r\ c)) &= \{ x \in D\ ;\ \forall y\ ;\ \langle x,y \rangle \in I(r) \Rightarrow y \in I(c) \} \\
I((\texttt{inv } r)) &= \{ \langle x,y \rangle\ ;\ <y,x> \in I(r) \} \\
I((\texttt{domain } c)) &= \{ \langle x,y \rangle\ ;\ x \in I(c) \}
\end{aligned}
$$

As usual, a model of a terminology is an interpretation I which satisfies all the assertions of the terminology.

DLML takes advantage of the modular design of description logics by describing individual constructors separately. The modular encoding of the description logics is made of three kind of DTD: atoms (introducing the atomic terms), term constructors (e.g., `all`, `and`, `not`) and formula constructors (e.g. =, ≤). An arbitrary number of these XML files are put together in order to form a particular logic.

For instance below is the content of the DTD of the `INV` (converse of a role) constructor:

```
<!ELEMENT dl:INV (%dl:RDESC;)>
```

We have also defined the notion of Document Semantic Description (DSD) which enables the description of the formal semantics of an XML language (just like the DTD or schemas express the syntax). To the DTD above is associated a DSD describing the semantics of the operator (i.e. $I((\texttt{inv}\ r))=(I(r))^{-1}$):

```
<dsd:denotation match="dl:INV">
  <mml:eq/>
  <mml:apply>
    <mml:inverse/> <!-- converse for binary relations -->
    <dsd:apply-interpretation select="*[1]"/>
  </mml:apply>
</dsd:denotation>
```

In the experimental DSD language, the XML elements are identified by XPATH [Clark 1999b] expressions (`dl:INV` or `*[1]` standing for any term of constructor `INV` and any first argument of the term). The syntax is very similar to that of XSLT [Clark 1999a] (with `denotation`, `interpretation` and `apply-interpretation` corresponding to `template` and `apply-template`). The remaining expressions are mathematical symbols expressed in MathML [Carlisle 2001].

The DLML family of languages provides the DTD and DSD of all the covered operators and can build automatically those of a particular logic from its DLML description. The DLML logic descriptions are like the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE dlml:logic SYSTEM "dlml.dtd">

<dlml:logic name="shiq" version="1.0">
        <dlml:atoms/>

        <dlml:cop name="anything"/>
        <dlml:cop name="nothing"/>
        <dlml:cop name="and"/>
        <dlml:cop name="or"/>
        <dlml:cop name="not"/>
        <dlml:cop name="all"/>
        <dlml:cop name="some"/>
        <dlml:cop name="csome"/>
        <dlml:cop name="catleast"/>
        <dlml:cop name="catmost"/>
        <dlml:rop name="inv"/>
        <dlml:rop name="trans"/>

        <dlml:cint name="cprim"/>
</dlml:logic>
```

From this description, two XSLT stylesheets can generate the DTD and DSD corresponding to the language. They can be used for expressing SHIQ terminologies in XML.

2.2 Transformations

What can such a DTD for description logics be good for? Once a language is encoded in XML, it is very easy to transform syntactically a representation into another one. A transformation is an algorithmic manner to generate one representation from another (not necessarily in the same language). A transformation $\tau: L \rightarrow L'$, from a representation r of L generates a representation $\tau(r)$ in L'.

More precisely, we take advantage of the XSLT transformation language ("XML Style Language Transformations" [Clark 1999a]) recommended by W3C, for which we put forward a compound transformation language (see §5.1).

The first application is the import and export of terminologies from a description logic. In our example, the representations in OIL and DAML-ONT are imported in DLML through transformations. Then, the result is exported to SHIQ (the FaCT system [Bechhofer1999a] has an XML entry point). These transformations are simple XSLT stylesheets.

More elaborate transformations can be developed. The imported representations are then merged and three successive steps (inspired from those of OIL [Horrocks 2000]) are applied to the result: the three steps concern the suppression of the `DOMAIN` constructor with the help of the `ALL` and `INV` constructors (`domain2allinv`), the suppression of the `ONE-OF` constructor with the help of new exclusive concepts (`oneof2orcexcl`) and the elimination of the exclusion introducers with the help of the `NOT` constructor (`cexcl2not`).

The piece of stylesheet presented below converts a terminology containing the `DOMAIN` restrictions on roles (attributes) in a terminology which replaces them by a `ALL` constraint on the inverse (`INV`) of the role applied on the whole universe (`ANYTHING`). For instance, it will convert:

```
inktype ≤ (domain InkPrinter)
```

into:

```
AnyThing ≤ (all (inv inktype) InkPrinter)
```

Both formulas equally say that only `InkPrinters` can have the `inktype` attribute.

```
<xsl:template match="dl:TERMINOLOGY">
  <dl:TERMINOLOGY>
    <xsl:comment>Introduction of the DOMAIN</xsl:comment>
    <dl:CPRIM>
      <dl:ANYTHING />
      <dl:AND>
        <xsl:apply-templates select="dl:RPRIM " mode="gatherdomain" />
      </dl:AND>
    </dl:CPRIM>
    <xsl:comment>The terminology</xsl:comment>
    <xsl:apply-templates />
  </dl:TERMINOLOGY>
</xsl:template>

<!-- gather domains in role introduction and add this for root -->

<xsl:template match="dl:RPRIM " mode="gatherdomain">
  <dl:ALL>
    <dl:INV>
      <dl:RATOM><xsl:value-of select="dl:RATOM[1]/text()"/></dl:RATOM>
    </dl:INV>
    <xsl:apply-templates select="dl:DOMAIN/*" />
  </dl:ALL>
</xsl:template>

<!-- usual processing -->
```

```
<xsl:template match="*|@*|text()">
  <xsl:copy><xsl:apply-templates select="*|@*|text()"/></xsl:copy>
</xsl:template>

<xsl:template match="dl:RPRIM">
  <dl:RPRIM>
    <dl:RATOM><xsl:value-of select="dl:RATOM[1]/text()"/></dl:RATOM>
    <xsl:choose>
      <xsl:when test="dl:DOMAIN">
        <dl:ANYRELATION/>
      </xsl:when>
      <xsl:otherwise><xsl:copy-of select="."/></xsl:otherwise>
    </xsl:choose>
  </dl:RPRIM>
</xsl:template>
```

This stylesheet gathers all the `DOMAIN` constraints of relations in a range (`ALL`) constraint of the inverse (`INV`) of the relation and applies it to `ANYTHING`. Then, it reproduces the whole terminology with domain constraints suppressed (i.e. replaced by `ANYRELATION`) [1].

Such transformations are assembled for transforming terminologies in one logic into another, equivalent, one. This is what is achieved in our example involving the three transformations.

3. Properties : consequence preservation

Operationally, the content of the previous section is sufficient for importing a representation from one language to another. However, it does not provide any idea of what properties are satisfied by each transformation step, nor by the transformation process as a whole. In order for the semantic web to be safely used by machines, it is necessary to define what properties have to be satisfied by the transformations. We focus here on the consequence preservation property (a semantic property) which is described in §3.1. In the context of families of languages we have described a set of more precisely characterized properties that entail consequence preservation. They are presented in the following subsections (§3.2-3.4).

3.1 Transformation properties

A property is a Boolean predicate about the transformation (e.g., "preserving information" is such a predicate — it is true or false of a transformation — and is satisfied if there exists an algorithmic way to recover r from $\tau(r)$). We consider more closely preservation properties which preserve (or counter-preserve) an order relation between the source representation (r) and the target representation ($\tau(r)$). There can be many such properties (content or structure preservation, traceability, and confidentiality…) affecting different aspects of the representation. They can be roughly classified as:

- — Syntactic properties : like the completion ($\tau(r) << r$, in which << denotes structural subsumption between representations) ;
- — Semantic properties : like consequence preservation ($\tau(r) \Rightarrow r$, i.e. equation 2 below) ;
- — Semiotic properties : like interpretation preservation (let σ be the interpretation rules and $|\equiv_i$ be the interpretation of individual i, $\forall \delta \in L$, $\forall$ i,j, $r,\sigma |\equiv_i \delta \Rightarrow \tau(r),\tau(\sigma) |\equiv_j \tau(\delta)$).

[1] This transformation is not sufficient to eliminate all occurrences of domain. For instance, (all (domain C) C') has to be transformed into (or (not C) (all anyrelation C')). But this is sufficient for our demonstration.

In the context of the communication of formal representations, we would like to warrant the preservation of the meaning of the representations. This can be defined by the two complementary equations:

$$\forall r \subseteq L,\ \forall \delta \in L,\ r \models_L \delta \Rightarrow \tau(r) \models_{L'} \tau(\delta) \quad (1)$$

$$\forall r \subseteq L,\ \forall \delta \in L,\ \tau(r) \models_{L'} \tau(\delta) \Rightarrow r \models_L \delta \quad (2)$$

Generalized interoperability is, of course, out of reach. Consequently we study restricted cases of these equations. In the context of the "family of languages" approach we identified several properties presented below which are more precise and entail equation (1).

3.2 Language inclusion

The simplest transformation is the transformation from one logic to a syntactically more expressive one (i.e. which adds new constructors). The transformation is then trivial, but yet useful, because the initial representation is valid in the new language; it is thus identity:

$$\forall \delta \in L,\ r \models_L \delta \Rightarrow r \models_{L'} \delta$$

This trivial interpretation of semantic interoperability is one strength of the "family of languages" approach because, in the present situation, nothing has to be done for gathering knowledge. For this case, one can define the relation between two languages L and L' as $L<L'$ which has to comply with $L \subseteq L'$. We can then define $L=L'$as equivalent to $L<L'$ and $L'<L$. This defines the syntactic structure of $\boldsymbol{L}$.

This simple property is satisfied by the merge operation that puts together the two representations issued from the DAML-ONT translation and the OIL translation.

3.3 Model preservation

If $L<L'$ does not hold, the transformation is more difficult. The initial representation r can be restricted to what is (syntactically) expressible in L': $\tau_{<}(r)$. However, this operation (which is correct) is incomplete because it can happen that a consequence of a representation expressible in L' is not a consequence of the expression of that representation in L':

$$\exists \delta \in L';\ \tau_{<}(r) \not\models_{L'} \delta \text{ and } r \models_L \delta$$

To solve this problem, as stated in [Visser 2000a], it is necessary to deduce from r in L whatever is expressible in L'. Let $\tau_{\prec}(r) = \tau_{<}(\mathrm{Cn}(r))$ be this expression. It is such that $\forall r \subseteq L$, $\forall \delta \in L \wedge L',\ r \models_L \delta \Rightarrow \tau_{\prec}(r) \models_{L'} \delta$.

The previous proposal is restricted in the sense that only expressions of the source language are allowed in the target language, though there exist equivalent non-syntactically comparable languages. This is the case of the description logic languages ALC and ALUE which are known to be equivalent while none has all the constructors of the other. For that purpose, one can define $L \prec L'$ if and only if the models are preserved, i.e.

$$\exists \tau_{\prec};\ \forall r \subseteq L,\ \forall \langle I, D\rangle;\ \langle I, D\rangle \models_{L'} \tau_{\prec}(r) \Rightarrow \langle I, D\rangle \models_L r$$

This property is satisfied by the `domain2allinv` and `cexcl2not` transformations.

The $\tau_{\prec}$ transformation is not easy to produce (and can generally be computationally expensive) but we show, in §4.1, how this can be practically achieved.

3.4 Model isomorphism

Another possibility is to define $\lhd$ as the existence of an isomorphism between the models of r and those of $\tau_{\lhd}(r)$:

$$\exists \tau_{\lhd};\ \forall \langle I', D'\rangle,\ \exists \langle I, D\rangle;\ \forall r \subseteq L,\ \langle I', D'\rangle \models_{L'} \tau_{\lhd}(r) \Rightarrow \langle I, D\rangle \models_L r$$

This also ensures that $\forall r \subseteq L,\ \forall \delta \in L,\ r \models_L \delta \Rightarrow \tau_{\lhd}(r) \models_{L'} \tau_{\lhd}(\delta)$.

This property is satisfied by the `oneof2orcexcl` transformation. It can be used in order to use a prover built for a logic with a logic whose models are isomorphic to it.

This provides a structure based on semantics to the family of languages $\boldsymbol{L}$. Summarizing, the syntactic and semantic structure of a language family provides different semantic properties characterizing transformations, all of them entailing consequence preservation.

We have considered only transformations that do preserve all information because languages have at least the same expressivity. It can happen that representations are imported to a language of lower expressivity. In such a case, consequence preservation cannot be ensured. Some information must be lost by the transformation. This can be subject to properties that characterize the kind of information that can be sacrificed.

4. Proofs, annotations and proof-checking

The approach to semantic interoperability defended here is based on transformations and their properties. Hence, in order to ensure formally the properties of transformations, one must exhibit a proof of the property. In fact, the proof and the transformation can be strongly tied together to the extent that they are built together (§4.1). In such a case, the publication of the proof is as important as the publication of the transformation (§4.2). The proof can be checked thus providing confidence with the corresponding transformation (§5.2).

4.1 From proofs to transformations

When providing transformations from one language to another, it is useful to prove the properties that are satisfied by the transformations (e.g. that the transformation terminates or that it preserves interpretations). For instance, the proof that the `domain2allinv` transformation preserves interpretations is as follows (inference rules are in brackets):

	$r \leq (\texttt{domain}\ C)$	[hypothesis](0)
$\Rightarrow$	$I(r) \subseteq I((\texttt{domain}\ C))$	[dsd/syn-to-sem](1)
$\Rightarrow$	$I(r) \subseteq \{\langle x,y\rangle \in D^2;\ y \in I(C)\}$	[dsd/expand-interp](2)
$\Rightarrow$	$\forall \langle x,y\rangle \in I(r),\ y \in I(C)$	[sets/incl-in](3)
$\Rightarrow$	$\forall x \in D,\ \forall y,\ \langle x,y\rangle \in I(r) \Rightarrow y \in I(C)$	[pc/quant-intro](4)
$\Rightarrow$	$\forall x \in D,\ \forall y\ \langle x,y\rangle \in \{\langle w,z\rangle;\ \langle z,w\rangle \in I(r)\} \Rightarrow y \in I(C)$	[set/in-incl](5)
$\Rightarrow$	$D \subseteq \{x \in D,\ \forall y;\ \langle x,y\rangle \in \{\langle w,z\rangle;\ \langle z,w\rangle \in I(r)\} \Rightarrow y \in I(C)\}$	[dsd/retract-interp](6)
$\Rightarrow$	$D \subseteq \{x \in D;\ \forall y;\ \langle x,y\rangle \in I((\texttt{inv}\ r)) \Rightarrow y \in I(C)\}$	[dsd/retract-interp](7)
$\Rightarrow$	$I(\texttt{AnyThing}) \subseteq \{x \in D;\ \forall y;\ \langle x,y\rangle \in I((\texttt{inv}\ r)) \Rightarrow y \in I(C)\}$	[dsd/retract-interp](8)
$\Rightarrow$	$I(\texttt{AnyThing}) \subseteq I((\texttt{all}\ (\texttt{inv}\ r)\ C))$	[dsd/retract-interp](9)
$\Rightarrow$	$\texttt{AnyThing} \leq (\texttt{all}\ (\texttt{inv}\ r)\ C)$	[dsd/sem-to-syn](10)

This proof, like many language equivalence proofs in description logics, shows that whatever term built from some term constructor (here `DOMAIN`) is expressible with other term constructors (here `ALL`, `INV` and `ANYTHING`) though preserving the interpretation of the terms. One characteristic of such proofs in term-based languages is that they are constructive: they exhibit a transformation from one language to the other. They can thus be translated into a transformation (and this results in the XSLT example presented in §2.2).

Another example is the transformation from ALUE to ALC, which is based on the argument that any `NOT` constructor can be pushed down the term structure:

$(\texttt{not}\ c)$	$\Leftrightarrow$	$(\texttt{anot}\ c)$	for c atomic
$(\texttt{not}\ (\texttt{anot}\ c))$	$\Leftrightarrow$	c	
$(\texttt{not}\ (\texttt{not}\ c))$	$\Leftrightarrow$	c	
$(\texttt{not}\ (\texttt{all}\ r\ c))$	$\Leftrightarrow$	$(\texttt{csome}\ \texttt{r}\ (\texttt{not}\ \texttt{c}))$	
$(\texttt{not}\ (\texttt{and}\ c_1, \ldots\ c_n))$	$\Leftrightarrow$	$(\texttt{or}\ (\texttt{not}\ c_1) \ldots\ (\texttt{not}\ c_n))$	

(not (some *r*)) ⇔ (all *r* Nothing)

This proof can be turned into a transformation, which applies the rules (from left to right) recursively on the structure of the terms. In DLML, many of the transformations across languages have been designed together with their proofs. We did this for the above transformations. This principle, which is the instantiation of the Curry-Howard correspondence, can be applied to many transformations.

4.2 Proof annotations

If the designers build proofs of some properties, it is desirable, especially in a worldwide distributed environment, to publish these proofs. It is thus useful to be able to represent them. The representation of the proof itself can be provided in MathML [Carlisle 2001] and OMDoc [Kohlhase 2000] a language extending MathML towards the expression of mathematical macrostructures (e.g., theories, theorems, axioms, and proofs). In this formalism, the two first steps of the proof above would look like:

```
<omd:proof id='domain2allinvpr' for='domainelim' theory='dlml'>
  <omd:hypothesis id='domain2allinv_0'/>
  <omd:derive id='domain2allinv_1'>
    <omd:FMP>
      <omd:assumption id='domain2allinv_0'>
        <OMOBJ>
          <dl:rprim>
            <dl:ratom>r</dl:ratom>
            <dl:domain>
              <dl:catom>C</dl:catom>
            </dl:domain>
          </dl:rprim>
        </OMOBJ>
      </omd:assumption>
      <omd:conclusion id='domain2allinv_1cl'>
        <OMOBJ>
          <mml:apply><mml:subset/>
            <dsd:apply-interpretation>
              <dl:ratom>r</dl:ratom>
            </dsd:apply-interpretation>
            <dsd:apply-interpretation>
              <dl:domain><dl:catom>C</dl:catom></dl:domain>
            </dsd:apply-interpretation>
          </mml:apply>
        </OMOBJ>
      </omd:conclusion>
    </omd:FMP>
    <omd:method><omd:ref theory='dsd' name='syn-to-sem'/></omd:method>
    <omd:premise xref='domain2allinv_0'/>
  </omd:derive>
  <omd:derive id='domain2allinv_2'>
    <omd:FMP>
      <omd:assumption id='domain2allinv_1cl'/>
      <omd:conclusion id='domain2allinv_2cl'>
        <OMOBJ>
          <mml:apply><mml:subset/>
            <dsd:apply-interpretation>
              <dl:ratom>r</dl:ratom>
            </dsd:apply-interpretation>
            <dsd:apply-interpretation>
              <dl:domain><dl:catom>C</dl:catom></dl:domain>
            </dsd:apply-interpretation>
          </mml:apply>
```

```
          </OMOBJ>
        </omd:conclusion>
      </omd:FMP>
      <omd:method><omd:ref theory='dsd' name='expand-interp'/></omd:method>
      <omd:premise xref='domain2allinv_1cl'/>
      <omd:conclusion>
    </omd:derive>
    …
    <omd:derive id='domain2allinv_10'>
      <omd:FMP>
        <omd:assumption id='domain2allinv_9cl'/>
        <omd:conclusion id='domain2allinv_10cl'>
          <OMOBJ>
            <dl:cprim>
              <dl:anything/>
              <dl:all>
                <dl:inv><dl:ratom>r</dl:ratom></dl:inv>
                <dl:catom>C</dl:catom>
              </dl:all>
            </dl:cprim>
          </OMOBJ>
      </omd:conclusion>
      </omd:FMP>
      <!-- this is substitution of interpretation by its definition -->
      <omd:method><omd:ref theory='dlml' name='completeness'/></omd:method>
      <omd:premise xref='domain2allinv_9cl >
    </omd:derive>
    <omd:conclude id='domain2allinv_10'>
      <omd:FMP>
        <omd:assumption id='domain2allinv_9cl'/>
        <omd:conclusion id='domain2allinv_10cl'>
          <OMOBJ>
            <dl:cprim>
              <dl:anything/>
              <dl:all>
                <dl:inv><dl:ratom>r</dl:ratom></dl:inv>
                <dl:catom>C</dl:catom>
              </dl:all>
            </dl:cprim>
          </OMOBJ>
      </omd:conclusion>
      </omd:FMP>
      <!-- this is substitution of interpretation by its definition -->
      <omd:method><omd:ref theory='dlml' name='completeness'/></omd:method>
    </omd:conclude>
</omd:proof>
```

The namespace prefix are `omd` for OMDoc, `mml` for MathML, `dsd` for DSD and `dl` for DLML. We took some liberty with OMDoc (e.g. instead of OpenMath objects — OMOBJ — we put MathML expressions, because DSD is based on MathML instead of OpenMath). However, this is just a matter of syntax: the relevant part is the ability of OMDoc for representing proofs.

It is also useful to attach the property and the proof to the transformations. One solution consists of adding it to the transformation structure. There are two problems with this solution: the XSLT language does not enable this, though Transmorpher does, and this would prevent people who are not owner of the transformation to claim properties and publish proofs. Hence the best solution seems to use RDF for annotating the transformations from the outside.

5. Composing transformations, composing proofs

In a family of languages, composing transformations can be a very convenient way to transform from one language to another. This is what has been proposed in the introductory example. Each elementary transformation can be used in various compound transformations. We have developed a system, Transmorpher, for dealing with such composition of more elementary transformations (called transformation flows, §5.1). Transmorpher is an environment for defining transformations and assembling them, on one hand, annotating them by properties they satisfy or those they must satisfy and proving the properties of compound transformations on the other hand. The proof of properties of components can be gathered from the web and checked (§5.2) and the proof of the compound transformation can be obtained by composing the properties of the components (§5.3). Once the proofs are produced, both the transformation and the proof can be exported to the web (§5.4).

5.1 Transmorpher

In order to prove or check the properties of transformations, it is necessary to have a representation of these transformations. The XSLT language enables the expression of a transformation in XML but is relatively difficult to analyze. In order to overcome that problem, we have designed and developed in collaboration with the FluxMedia company, the Transmorpher environment [Euzenat 2001b]. It is a layer on top of XSLT allowing the expression of complex transformation flows such as the one of Figure 2 (which is that of the example). A transformation flow is the composition of elementary transformation instances whose input/output are connected by channels. A transformation flow is itself a transformation.

One of the goals of Transmorpher is the encapsulation of XSLT, used for performing the transformations, such that transformations are easier to analyze through special purpose syntax and hierarchical decomposition. This should facilitate the description of proofs through “Lemmas” attached to component transformations.

Transmorpher enables the definition and processing of generic transformations of XML documents. It provides XSLT extensions for:

- Describing straightforwardly simple transformations (removing elements, replacing attribute names, merging documents...);
- Composing transformations by connecting their (multiple) input and output;
- Applying transformations until closure;
- Applying regular expression substitution;
- Calling external transformation engines (such as XSLT).

Transmorpher describes the transformation flows in XML. Input/output channels carry the information, mainly XML, from one transformation to another. Transformations can be other transformation flows or elementary transformations. Transmorpher provides a set of abstract elementary transformations (including their execution model) and one default instantiation. Among elementary transformations are external calls (e.g. XSLT), dispatchers, serializers, query engines, iterators, mergers, generators and rule sets. Figure 2 presents the representation of the above transformation flow in Transmorpher graphic format.

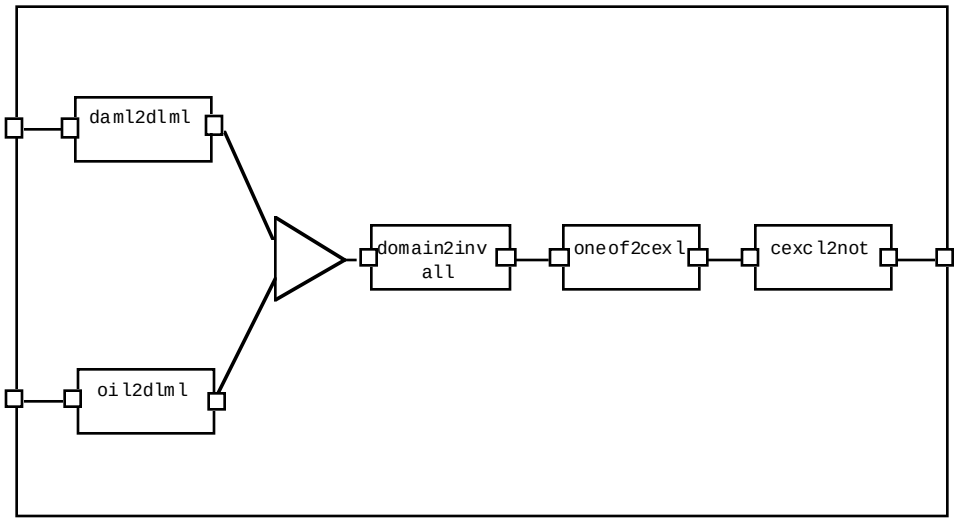


Figure 2 : Transmorpher description of the importation of DAML-ONT and OIL fragments into the DLML representation of SHIQ.

Transmorpher is mainly a set of documented Java classes (which can be refined or integrated into other software) and a transformation flow processing engine. A transformation flow can be expressed by programming in Java or providing an XML description. Figure 2 is the description of the following transformation flow:

```
<process name="assemble-onto" in="i1 i2" out="o">
  <apply-external type="xslt" name="daml2dlml" file="daml2ldaml.xsl"
     in="i1" out="o1"/>
  <apply-external type="xslt" name="oil2dlml" file="oil2loil.xsl"
     in="i2" out="o2"/>
  <merge type="concat" name="ldaml+ldaml" in="o1 o2" out="o3"/>
  <apply-external type="xslt" name="domain2allinv"
     file="domain2allinv.xsl" in="o3" out="o4"/>
  <apply-external type="xslt" name="oneof2cexclor" file="oneof2or.xsl"
     in="o4" out="o5"/>
  <apply-external type="xslt" name="cexcl2not" file="cexcl2not.xsl"
     in="o5" out="o"/>
</process>
```

An extension of Transmorpher consists of attaching assertions to the transformations in a transformation flow in order to tell if a property is assumed, proved or to be checked. This will allow real experimentation of proving properties of compound transformations.

5.2 Towards proof checking

Proof-carrying code [Necula 1998] is an infrastructure in which a program is provided with the proof of the properties that it satisfies. A client system that wants to run the former program will check the proof against this program in order to ensure that it can do it safely. These principles can be applied to the verification of the transformations and their properties as soon as a representation of the proof is available.

In order to be able to check proofs of semantic properties such as (1) or (2), it is necessary to have (a) the representation of the transformation which is provided by XSLT or by Transmorpher, (b) the semantics of the transformation language, (c) the representation of the semantics of the logics provided by their DSD and (d) the representation of the proof like the one described above. Of these elements, the only missing one is the representation of the semantics of XSLT. There are several attempts, however, to provide a semantics for XSLT fragments that can be used [Wadler 2000, Bex 2000]. Another path consists of defining a

transformation language simpler than XSLT but with a clean semantics. This is partly the case of Transmorpher.

Checking is the opposite of trusting. Both approaches have different advantages: trusting does not require to spend time checking the arguments while checking does not require to maintain a heavy model of trust and is independent of who provides the arguments. Proof-carrying code can be applied to untrusted items. So if someone needs particular transformations satisfying particular properties, she can try to find such transformations and proof of properties on the web and check them.

Unlike watermarking, proof-carrying code does not require any encoding of the transformation because it checks the proof against the program. The program can have been modified, if the checker finds that the proof is still valid, then this is all that is required. It is not even required that the proofs are provided with the program. In fact, someone can publish an automatic proof of the termination of the above transformation web site not connected to the DLML one and the proof-checker must be able to decide if the proof is valid or not.

5.3 Proof by composition

Once the properties of elementary transformations are available, either by checking, trusting or proving, an interesting point is the elaboration of the proof of properties for transformation flows.

If each of these more elementary transformations is annotated by the assertion of the properties it satisfies, the property concerning the compound transformation remains to be computed. A very simple example is the termination property on finite input that is preserved through composition, but not by iteration until saturation. Model preservation for its part is preserved through both composition and iteration.

This can be exemplified with the properties that have been considered in §3. It is possible to establish the properties of the composition of two transformations given their own properties. This yields (the very simple) Table 1.

	$>$	$\succ$	$\lhd$	$\leq$	Ø
$>$	$>$	$\succ$	$\lhd$	$\leq$	Ø
$\succ$	$\succ$	$\succ$	$\lhd$	$\leq$	Ø
$\lhd$	$\lhd$	$\lhd$	$\lhd$	$\leq$	Ø
$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	Ø
Ø	Ø	Ø	Ø	Ø	Ø

Table 1 : Composition table for the semantic relations on transformations (≤ is consequence preservation).

Table 1 shows that the transformation flow above, that assembles all transformations of the example, is indeed consequence preserving.

5.4 Safe transformation development cycle

The techniques presented here provide a framework in which transformations from one representation language to another are available from the network and proofs of various properties of these transformations are attached to them. It is noteworthy that transformations and proofs do not have to come from the same origin. They can even be produced by the application.

The transformation system engineer can gather these transformations and their proofs, check the proofs before importing them in the transformation development environment. She will then be able to create a new transformation flow and generate the proofs of the required

properties. Finally, she will be able to publish on the network the transformation and its proof.

Given two languages with their semantics, in order to transform representations in one language into representations in the other that satisfy some properties, the following transformation edition process (see Figure 1) can be attempted:

1. Fetching transformations that can help performing part of the task ;
2. Fetching assertions and proofs about these transformations ;
3. Checking the proof or trusting the assertions of properties about the transformations ;
4. Composing transformations into a global transformation that is supposed to do the transformation;
5. Proving that this composition preserves the properties that are required by the global transformation ;
6. Publishing transformation, assertions and proofs for others to use it.

Then, the problem proposed in introduction will be reduced to: gather available ontologies, create a safe transformation flow for importing them in the current knowledge processing environment and apply the transformation flow. The transformation flow can be applied at any time for updating the compound ontology and its properties will remain valid as long as the languages remain the same.

6. Conclusion

We have presented a framework for formally ensuring semantic interoperability in the semantic web. Interoperability is assured by transformations that have to satisfy some client-defined properties. The proof of properties are encoded in a machine-readable way so that the client can check them. Transmorpher enables the composition of these transformations into a more elaborate one whose proof of properties can be facilitated by simple composition of the properties of its components (either proof-checked or trusted).

If enough actors are interested in sharing transformations safely instead of developing again and again the same transformation, here is an architecture enabling its formal and modular realization. We strongly believe that there will be a strong interest in such a framework in the context of the growing use of XML and XML transformations inside and across companies. In fact, if semantic properties are more related to the semantic web, many other properties of general interest can be taken into account by this framework.

The main strength of the framework is not its sophistication, but rather its relative simplicity. Its distributed, modular and incremental characteristics make it adapted to the web. No doubt that it will not be practical in all cases, but it works for cases like the one presented.

This framework is very close to that of proof-carrying code [Necula 1998] of which it is an instantiation on particular programs and properties. Moreover it is fully based on widely available XML technologies (XML, XPATH, XSLT, MATHML, OMDOC, RDF) or local extensions (DLML, DSD, Transmorpher). For a description of complementary work on the topic of semantic interoperability (e.g. [Masolo 1999, Chalupsky 2000, Ciocoiu 2000]), see [Euzenat 2001a].

This infrastructure is a prospective framework for which many pieces are already available and several of them linked together. The main part of it, with the notable exception of proof-checking, has already been implemented as a proof of concept. The DLML framework is operational and several experiments have been made with XSLT transformations. Transmorpher is an ongoing work whose basic functions are operational. The OMDoc and DSD languages are available.

We have some examples of proof (mainly of model preservation or model isomorphism) in description logics that should be a very good first testbed for the application of these concepts. We also have examples of transformations between heterogeneous representations (e.g. description logics and syllogistic).

The proof-checker is the difficult point because we will need one that can interface easily with the kind of proofs required by the framework. There are two issues to be solved next : generalization and scalability.

Generalization requires a lot of fundamental work about topics such as generalizing from DLML to other representation languages (we have superficially investigated syllogisms and considered DAML-ONT as a description logic language), generalizing semantics properties, generalizing to other (e.g. structural, semiotic) properties, generalizing the kind of proofs required. We are currently committed to investigate the semantic properties more thoroughly.

Robustification and scalability will be required in order to consider the workability of the whole system. Positive elements are the intrinsic distribution of our framework and the fact that any element can be replaced by another with similar interface.

Acknowledgements

The author is indebted to Heiner Stuckenschmidt who proposed the introductory example for presentation of the "family of languages" approach. Anonymous reviewers are thanked for suggesting several improvements of the paper.

References

[Berners-Lee 2001] Tim Berners-Lee, James Hendler, Ora Lassila, The semantic web, *Scientific american* 279(5):35-43, 2001, http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

[Bechhofer 1999] Sean Bechhofer, Ian Horrocks, Peter Patel-Schneider, Sergio Tessaris, A proposal for a description logic interface, Proc. int. workshop on description logics, Linköping (SE), CEUR-WS-22, 1999 http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-22/bechhofer.ps

[Bex 2000] Geert Jan Bex, Sebastian Maneth, Frank Neveu, A formal model for an expressive fragment of XSLT, *Lecture notes in computer science* 1861:1137-1151, 2000

[Carlisle 2001] David Carlisle, Patrick Ion, Robert Miner, Nico Poppelier (eds.), Mathematical markup language (MATHML) version 2.0, Recommendation, W3C, 2001 http://www.w3.org/TR/MathML2

[Chalupsky 2000] Hans Chalupsky, OntoMorph: a translation system for symbolic knowledge, Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US), pp471-482, 2000

[Clark 1999a] James Clark (ed.), XSL transformations (XSLT) version 1.0, Recommendation, W3C, 1999 http://www.w3.org/TR/xslt

[Clark 1999b] James Clark, Stephen DeRose (eds.), XML path language (XPath) version 1.0, Recommendation, W3C, 1999. http://www.w3.org/TR/xpath

[Ciocoiu 2000] Mihai Ciocoiu and Dana Nau, Ontology-based semantics, Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US), pp539-546, 2000 http://www.cs.umd.edu/~nau/papers/KR-2000.pdf

[Donini 1994] Francesco Donini, Maurizio Lenzerini, Daniele Nardi, Andrea Schaerf, Deduction in concept languages: from subsumption to instance checking, *Journal of logic and computation* 4(4):423-452, 1994

[Euzenat 2001a] Jérôme Euzenat, Towards a principled approach to semantic interoperability, Proc. IJCAI workshop on Ontologies and information sharing, Seattle (WA US), 2001 to appear

[Euzenat 2001b] Jérôme Euzenat, Laurent Tardif, XML transformation flow processing, Proc. 2nd Extreme markup languages, Montréal (CA), 2001, to appear, http://transmorpher.inrialpes.fr/paper

[Euzenat 2001c] Jérôme Euzenat, Heiner Stuckenschmidt, The 'family of languages' approach to semantic interoperability, submitted, 2001

[Euzenat 2001d] Jérôme Euzenat, Preserving modularity in XML encoding of description logics, Proc. 13th description logic workshop, Stanford (CA US), 2001 to appear

[Horrocks 2000] Ian Horrocks, A denotational semantics for Standard OIL and Instance OIL, 2000, http://www.ontoknowledge.org/oil/downl/semantics.pdf

[Kohlhase 2000] Michael Kohlhase, OMDoc : an open markup format for mathematical documents, SEKI report SR-00-02, Universität des Saarlandes, Saarebrucken (DE), 2000 http://www.mathweb.org/src/mathweb/omdoc/doc/omdoc/omdoc.ps

[Masolo 2000] Claudio Masolo, Criteri di confronto e costruzione di teorie assiomatiche per la rappresentazione della conoscenza: ontologie dello spazio e del tempo, Tesi di dottorato, Università di Padova, Padova (IT), 2000

[Necula 1998] George Necula, Compiling with proofs, PhD thesis, Carnegie Mellon university, Pittsburgh (PA US), 1998

[Visser 2000a] Ubbo Visser, Heiner Stuckenschmidt, G. Schuster, Thomas Vögele, Ontologies for Geographic Information Processing, Computers in Geosciences, 2001, to appear http://www.tzi.de/buster/papers/Ontologies.pdf

[Wadler 2000] Philip Wadler, A formal semantics of patterns in XSLT, *Markup technologies*, 1999 http://www.cs.bell-labs.com/who/wadler/papers/xsl-semantics/xsl-semantics.pdf

[Wiederhold 1999] Gio Wiederhold, Jan Janninck, Composing diverse ontologies, Proc. 8th IFIP working group on databases working conference on database semantics, Rotorua (NZ), 1999 http://www-db.stanford.edu/SKC/publications/ifip99.html

Describing Computation within RDF

Chris Goad
The Behavior Engine Company
10 Sixth Street, Suite 108
Astoria, OR 97103
cg@behaviorengine.com

Abstract. *A programming language is described which is built within RDF. Its code, functions, and classes are formalized as RDF resources. Programs may be expressed directly using standard RDF syntax, or via a conventional JavaScript–based syntax. RDF constitutes not only the means of expression, but also the subject matter of programs: the native objects and classes of the language are RDF resources and DAML+OIL classes, respectively. The formalization of computation within RDF allows active content to be integrated seamlessly into RDF repositories, and provides a programming environment which simplifies the manipulation of RDF when compared to use of a conventional language via an API. The name of the language is "Fabl".*

1. Introduction

Fabl[1] is a programming language which is built within RDF[1]. The constituents of the language - its code, functions, and classes - are formalized as RDF resources, as is the data over which computation takes place. This means that programs reside within the world of RDF content rather than being relegated to a separate realm connected to RDF via an API. The starting point for the formalization is DAML+OIL[2].

The language provides an efficient imperative programming framework for the RDF domain. Programs may be expressed as RDF objects using standard RDF syntax, or via a conventional syntax which might be described as JavaScript[2] enhanced with types and qualified property names. The language is designed to be easy to learn for programmers familiar with the conventional JavaScript/HTML/XML/DOM web–programming model. In fact, the conceptual cleanliness of RDF makes the language and its semantics far simpler than this conventional model. The initial implementation is similar in runtime efficiency to other scripting environments.

As a computational formalism for RDF, the neighboring points of comparison for Fabl are the RDF APIs (eg [3], [4]), in which computation is expressed in conventional ways, but the subject matter of the computation is expressed in RDF. Fabl has several advantages over APIs:

1. Simplicity of programming.
2. Functions and programs can be managed, inspected, manipulated, and annotated in the same manner as any other RDF resources; they are first–class citizens of the RDF world.

3. Fabl's type system exploits the RDF property–centric style. This yields a system of a kind different, and in some ways more expressive and flexible, than those found in the main thread of object–oriented type systems running from Simula through C++, Java, C#, and Curl.
4. Fabl programs are formalized within RDF in a manner that provides an open framework for extension of the language. The implementation of Fabl is, with the exception of a few low level utilities, written in Fabl itself. Further, the process by which programs are analyzed and converted into an efficiently executable form can be extended by addition of new RDF content. This means that extension of Fabl to include new language facilities, such as new control structures, new syntax, or new typing systems built on different principles can all be carried out in the RDF style: by extending the base of RDF files which describe the language.

Although Fabl defines a particular (albeit, extensible) textual format for programs on the one hand, and implements a particular byte–code and virtual machine for interpretation on the other, the core of the design is its formalism for describing imperative computation *as* RDF. This integrates computation into the RDF realm of distributed semantic description, decoupled from any particular source language and from any particular execution technique. Concretely, active entities, from simple spreadsheets to complex simulations, can be formalized in RDF, and made available to any agent that has a use for them, independent of the language (or graphical interface) from which they were created.

Whether or not the particular formalism introduced here is the right one, RDF can and should be used as a vehicle for standardizing computation as well as passive content. If nothing else, Fabl shows the practicality of this idea.

2. Application Scenarios

Close integration of computation with RDF can benefit both sides of the integration. Most trivially, RDF mechanisms can be used to annotate programs - for example by using the Dublin Core[5] to assert information about date, author, and publisher of code. With the development of simple computational ontologies, metadata about code of the sort useful to software engineeers can be asserted in RDF; examples include call trees, traces, and performance information. The openness of RDF, which allows continually evolving vocabularies and tools to be applied to preexisting data, should benefit the realm of programming as much as any other domain.

Beyond annotation, the formalization of functions and code as RDF resources is the first step in integrating algorithmic computation and inference in an RDF setting. The combination of inference and algorithmic computation might be applied to automatic assembly of programs from available components, and to problem solving which mixes inference and algorithmic computation (when a subproblem is inferred to be solvable by an available algorithm, the algorithm is invoked). This direction of work requires more complex computational ontologies which formalize the kinds of statements about computational objects needed to support useful inference.

Going in the other direction, thorough integration of computation with RDF facilitates the development of active RDF content. The initial application to which we are appying Fabl provides an example. We have defined relatively simple ontologies for geography (themed maps, as in GIS), and for events located in a geographical context. This geographical and historical information is depicted by interactive web-delivered maps in the Macromedia Flash format (see our web site[6] for examples). The active aspect of our RDF repository consists

primarily of handlers which generate interactive maps from the underlying geographical and historical information, and which maintain consistency between the data and its depiction as changes are made. The handlers are RDF resources and their relationship to other data is expressed by RDF statements. Regularities (eg all resources in *this* class have *that* handler) are asserted by DAML+OIL restrictions.

This application provides a template for a wide range of possible applications, wherein complex situations are represented in RDF, and where consistency constraints are automatically maintained by associated constraint propogation mechanisms which are at least partly algorithmic (rather than strictly deductive) in nature. The kind of complete integration proposed here is not the only possible approach to this kind of application, but we would argue that first-class status for computational entities in the RDF world removes a layer of indirectness and complexity that would otherwise be necessary.

3. An Example

Consider the the simplest of data structures, a point on the plane with two coordinates, which can be expressed in Java by:

```
public class  Point {
    double xc;
    double yc;
}
```

Here is an extract from a Fabl RDF file at http://purl.oclc.org/net/fabl/examples/geom defining the same structure:

```
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:fabl="http://purl.oclc.org/net/nurl/fabl/"
  xmlns:nurl="http://purl.oclc.org/net/nurl/"
>
<daml:DatatypeProperty rdf:ID="xc"/>
<daml:DatatypeProperty rdf:ID="yc"/>

<rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#xc"/>
      <daml:toClass
rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
      <daml:cardinality>1</daml:cardinality>
    </daml:Restriction>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#yc"/>
      <daml:toClass
rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
      <daml:cardinality>1</daml:cardinality>
    </daml:Restriction>
</rdfs:Class>
```

The Fabl type system makes use of the March, 2001 version of DAML+OIL. The above RDF asserts that every member of **Point** has **xc** and **yc** properties, and that these properties each have exactly one value of type double. All of the examples in this paper use the name space declarations given just above, which will be abreviated in what follows by *[standard–namespace–declarations]*. Here is vector addition for points:

```
<rdf:RDF
[standard–namespace–declarations]
  xmlns:geom="http://purl.oclc.org/net/fabl/examples/geom#"
>

<fabl:code>
geom:Point function plus(geom:Point x,y)
  {
  var geom:Point rs;
  rs = new(geom:Point);
  rs . geom:xc = x.geom:xc + y.geom:xc;
  rs . geom:yc = x.geom:yc + y.geom:yc;
  return rs;
  }
</fabl:code>
</rdf:RDF>
```

The above text is not, of course, legal RDF. Rather, it represents the contents of a file intended for analysis by the Fabl processor, which converts it into RDF triples. The pseudo–tag **<fabl:code>** encloses Fabl source code; everything not enclosed by the tag should be legal RDF.

Note that the syntax resembles that of JavaScript, except that variables and functions are typed. Fabl types are RDF classes, and are named using XML qualified[7] or unqualified names (details below).

Here are the contents of the file http://purl.oclc.org/net/fabl/examples/color:

```
<rdf:RDF
[standard–namespace–declarations]>

<daml:Class rdf:ID="Color"/>
<Color rdf:ID="yellow"/>
<Color rdf:ID="blue"/>
<rdf:Property rdf:ID="colorOf">
  <rdfs:range rdf:resource="#Color"/>
<rdf:/Property>
```

The following fragment assigns a color to an existing **Point**: yellow if its **x** coordinate is positive, and blue otherwise:

```
<rdf:RDF
[standard–namespace–declarations]
  xmlns:geom="http://purl.oclc.org/net/fabl/examples/geom#"
  xmlns:color="http://purl.oclc.org/net/fabl/examples/color#"
>
```

```
<fabl:code>
  fabl:void function setColor(geom:Point x)
    {
    if (x . geom:xc > 0) x.color:colorOf = color:yellow;
    else x.color:colorOf = color:blue;
    }
  </fabl:code>
```

The expression **fabl:void** may only be used in a context where the return type of a function is indicated. It signifies that the function in question does not return a value. Note that **fabl:void** is not a class, and in particular is should not be identified with **daml:Nothing**. A function with return type **daml:Nothing** would indicate that the function returns a value belonging to **daml:Nothing** - an impossibility.

The setColor example illustrates the central difference between an RDF class and its counterparts in the object–oriented programming tradition. An RDF class is an assertion about properties possessed by a resource, which does not preclude the resource from having additional properties not mentioned in the class, nor from belonging to other classes, nor even from aquiring new properties and class memberships as time goes on. The progression of data types in programming languages exhibits growing freedom of type members: C or Pascal types exactly determine the structure of their members; C++ and Java classes determine the structure of members to a degree, but allow extension by subclasses; the RDF model leaves the structure of members free except as explicitly limited by the class definition.

Unless a property has been explicitly constrained to have only one value, Fabl interprets the value of a property selection:

```
x.P
```

as a bag. In the following example, the first function returns the number of colors assigned to an object, and the latter returns its unique color if it has only one, and a nul value otherwise.

```
xsd:int function numColors(daml:Thing x)
{
   return cardinality(x.color:colorOf);
}

color:Color function theColorOf(daml:Thing x)
{
   var BagOf(color:Color) cls;
   cls = x.color:colorOf;
   if (cardinality(cls)==1) return cls[0];
   else return fabl:undefined;
}
```

fabl:undefined is a special identifier which denotes no RDF value, but rather indicates the absence of any RDF value in the contexts where it appears.

4. RDF Computation in Fabl

RDF syntax and semantics can be viewed as having three layers: (1) a layer which assigns concrete syntax (usually XML) to RDF assertions, (2) the data model layer, in which RDF content is represented as a set of triples over URIs and literals, and (3) a semantic model, consisting of the objects and properties to which RDF assertions refer. DAML+OIL specifies

semantics[8] constraining the relationship between the data model and the semantic model.

The proper level of description for computation over RDF is the data model; the state of an RDF computation is a set of triples <**subject,predicate,object**>. This triple set in turn can be construed as a directed labeled graph whose nodes are URIs and literals, and whose arcs are labeled by the URIs of properties.

Fabl is executed by a virtual machine. An invocation of the Fabl VM creates an initial RDF graph which is in effect Fabl's own self description: the graph contains nodes for the basic functions and constants making up the Fabl language. Subsequent activity modifies the RDF graph maintained by the VM, called the "active graph". The Fabl interpreter can accept input from a command shell, or can be configured as a server in a manner appropriate to the application.

The universe of RDF files on the web plays the role of the persistent store for Fabl. The command

```
loadRdf(U)
```

adds the triple set described in the RDF file at URL **U** to the active graph.

The active graph is partitioned into *pages*. The data defining a page includes: (1) the external URL (if any) from which the page was loaded, (2) the set of RDF triples which the page contains, (3) a dictionary which maps the ids appearing in the page (as values assigned to the rdf:ID attribute) to the resources which they identify, and (4) a set of namespace definitions (bindings of URIs to namespace prefixes). Many pages are the internal representations of external RDF pages, but new pages can be created which are not yet stored externally.

```
saveRdf(x,U)
```

saves the page upon which x lies at the file **U**. The current implementation interacts with the external world of RDF via simple loading and saving of pages, but there are interesting additional possibilities involving distributed computation, which are outlined in a later section

A global variable or constant **X** with value **V** is represented by a **daml:UniqueProperty** named **X** whose value on the URI **fabl:global** is **V**. (It doesn't matter what values the property assumes when applied to other resources, nor does **fabl:global** play any other role.) For example, the following fragment defines the global **pi**:

```
  <daml:DatatypeProperty rdf:ID="pi">
    <rdf:type
rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
  </daml:DatatypeProperty>

  <daml:Class rdf:about="http://purl.oclc.org/net/nurl/fabl/global">
    <pi>3.14159265358979323846 </pi>
  </daml:Class>
```

The values of global properties can be referred to directly by name in Fabl. For example, since http://purl.oclc.org/net/fabl/examples/geom includes the lines above defining **pi**, the following fragment illustrates reference to **pi** as a global:

```
  <rdf:RDF
  [standard-namespace-declarations]
    xmlns:geom="http://purl.oclc.org/net/fabl/examples/geom#"
  >
```

```
<fabl:code>
xsd:double function timesPi(xsd:double x){return x * geom:pi}
</fabl:code>
```

As indicated in the initial example above, basic manipulation of the active graph is accomplished via conventional property access syntax: If **P** is the qualified name of a property, and **x** evaluates to an object, then

```
x.P
```

returns a bag of the known values of **P** on **x**, that is, the set of values **V** such that the triple **<x,P,V>** is present in the active graph. However, if **P** is asserted to be univalued - if it was introduced as a UniqueProperty, or has a cardinality restriction to one value - then

```
x.P
```

evaluates to the unique value instead. The assigment

```
x.P = E
```

for an expression **E** adds the triple **<x,P,value(E)>** to the active graph, unless **P** has been asserted to be a univalued, in which case the new triple replaces the previous triple (if any) which assigned a value to **P** on **x**. The command:

```
var Type name;
```

is equivalent to:

```
<daml:UniqueProperty rdf:ID="name">
  <rdfs:range rdf:resource="Type"/>
</daml:UniqueProperty>
```

The function:

```
new(Type)
```

creates a new node **N** in the active graph, and adds the triple **<N,rdf:type,Type>**. Initially, nodes created with the **new** operator lack an associated URI. However, Fabl allows URIs to be accessed and set as if they were properties, via the pseudo–property **uri**.

```
x.uri
```

is the current URI of **x** if it has one, and **fabl:undefined** if not.

```
x.uri = newURI;
```

assigns a new URI to **x**. If **newURI** is already assigned to another node **y** in the active graph, **x** is merged with **y**. The merged node will possess the union of the properties possessed by **x** and **y** prior to the merge.

5. RDF Computation Via an API: A Comparison

The Java code below uses the Jena API[4] to implement the function presented at the beginning of section 3: vector addition of points. This sample is included to give the reader a concrete sense of the difference between Fabl code, which expresses elementary RDF operations directly as basic operations of the language, and code using an API, in which the same elementary operations must be expressed as explicit manipulations of a representation of RDF content in the host language (here, Java). This is the only purpose of the sample, and the details are not relevant to anything that appears later in this paper. Also, the points made here apply equally to other RDF APIs.

```
  import com.hp.hpl.mesa.rdf.jena.mem.ModelMem;
  import com.hp.hpl.mesa.rdf.jena.model.*;
  import com.hp.hpl.mesa.rdf.jena.vocabulary.*;

  // The class GeomResources initializes variables
  // xc, yc, and Point to RDF resources of the right kind.
  public class GeomResources {
      protected        static        final        String        URI        =
"http://purl.oclc.org/net/fabl/examples/geom#";
      public static String getURI(){return URI;}
      public static Property xc = null;
      public static Property yc = null;
      public static Resource Point = null;
      static {
          try {
              xc = new PropertyImpl(URI, "xc");
              yc = new PropertyImpl(URI, "yc");
              Point = new ResourceImpl(URI+"Point");
             } catch (Exception e) {
              System.out.println("exception: " + e);
          }
      }
  }

  public class GeomFunctions {
  // PointPlus is vector addition
      public static Resource PointPlus(Resource x,Resource y) {
             Resource rs = x.getModel().createResource();
             rs.addProperty(RDF.type, GeomResources.Point);
             rs.addProperty(GeomResources.xc,
                x.getProperty(GeomResources.xc).getDouble() +
                y.getProperty(GeomResources.xc).getDouble());
             rs.addProperty(GeomResources.yc,
                x.getProperty(GeomResources.yc).getDouble() +
                y.getProperty(GeomResources.yc).getDouble());
            return rs;
             }
  }
```

The Fabl implementation, we would argue, is easier to understand and easier to code. The difference is not due to any defect of the Jena API, but to the inherent indirectness of the API approach. Further, the direct expression of RDF primitives in Fabl is less than half the story

with regards to ease of use. More significant is the fact that Fabl types *are* DAML+OIL classes, and type checking and polymorphism at the RDF level are implemented within the language. When using an API, type checking at the RDF level is the user's responsibility. For example, Java will not complain at compile time (nor run time) if the method *GeomFunctions.PointPlus* is applied to resources which are not members of **GeomResources.Point**.

6. Nurls

In normal RDF usage, locators (that is URLs) are often used as URIs whether or not the entities they denote exist on the web. However, nothing prevents the use of URIs which are completely unrelated to any web location, for example:

```
<rdf:Description rdf:about= "my_green_sedan">
```

Identifying an entity in a manner which does not make use of a WWW locator has two advantages. First, the question of where to find information about the entity is decoupled from naming the entity, which allows all of the different varieties of information about the entity to evolve without disturbing the manner in which the entity is named. Among other things, this simplifies the versioning of RDF data. Second, use of non–locating URIs frees up the content of the URI for expressing hierarchy information about the entities described.

In the Fabl implementation, the triple

```
<X,fabl:describedBy,U>
```

means that **U** denotes an RDF file which provides information about **X**. (**rdfs:isDefinedBy**[9] has a closely related, but not quite identical intent; descriptions need not always qualify as definitions). **U** is also taken as relevant to any subject **Y** whose URI (regarded as a pathname, with "." and "/" as delimiters) extends that of **X**. For example, if **my_green_sedan** is described by **U**, then so are **my_green_sedan.engine**, and **my_green_sedan/engine** but not **my_green_sedan_attenna**. The Fabl command:

```
getRdf(Y);
```

loads the files known to describe the resource **Y**; that is those files **F** for which the triple **<X,fabl:describedBy,F>** is present in the active graph,and **Y** is an extension of **X**. A typical Fabl initialization sequence involves first loading a configuration file containing **fabl:describedBy** statements which indicate where to find information about basic resources. Then, as additional resources become relevant to computation, invocations of **getRdf** bring the needed data into the active graph. In future, lazy strategies may be implemented in which, for example, **getRdf(X)** is automatically invoked on the first access to a property of **X**. Also, nothing precludes future development of complex discovery technology for finding relevant RDF, rather than relying only on the simple **describedBy** mechanism.

It is desirable that non–locating URIs not conflict with URLs. For Fabl applications, we have reserved the URI http://purl.oclc.org/net/nurl/ as a root URI whose descendants will never serve as locators. This line appears in our standard namespace declaration:

```
xmlns:nurl=”http://purl.olc.org/net/nurl”
```

Nurl stands for "Not a URL". The following fragment tells the Fabl VM where to find the description of the Fabl language, and illustrates the points just made:

```
   <rdf:Description about= "http://purl.oclc.org/net/nurl/fabl">
      <fabl:describedBy
rdf:resource="http://purl.oclc.org/net/fabl/languageV0"/>
   </rdf:Description>
```

To access a future release of the language which resides at .../languageV1, only the configuration file need change; RDF which mentions language primitives via the namespace prefix "fabl:" may be left unchanged.

If **U** is the source URL of a page in the active graph, and the triple **<X,fabl:describedBy,U>** is present in the active graph, then **X** is said to be a *subject* of the page. That is, the page has as its subject matter the part of the hierarchical URI name space rooted at **X**. A page may have more than one subject. When a new triple **<A,P,B>** is created in the course of computation, and the URI of **A** is an extension of the subject of a page, the new triple is allocated to that page. (Slightly more complex rules - not covered here - govern the case where the subjects of pages overlap.)

7. Identifiers

Identifiers in Fabl represent XML qualified or unqualified names. However, since the "." character is reserved for property selection, "." is replaced by "\" when an XML name is transcribed into Fabl. For example:

```
fablex:automobiles\ford
```

is the Fabl identifier which would have been rendered as

```
fablex:automobiles.ford
```

in XML. The interpretation of unqualified names is governed by the *path* and the *home namespace*. The path is a sequence of namespaces. When an unqualified name **U** is encountered, the Fabl interpreter searches through the path for a namespace **N** such that **N:U** represents a node already present in the active graph. When a new unqualified name **U** is encountered, it is interpreted as **H:U**, where **H** is the home namespace. Normally, the "fabl:" and "xsd:" namespaces are included in the path, enabling unqualified reference to Fabl language primitives and XML Schema datatypes[10].

8. Types

Any RDF type is a legal Fabl type.

```
X.rdf:type = T;
```

asserts that **X** belongs to the type **T**; that is it adds the triple **<X,rdf:type,T>** to the active graph. (This statement is legal only if **T** is a daml:Class, not an XML Schema datatype). Of course, a value may have many types.

Fabl also includes its own primitives for constructing new types, that is, for introducing new resources whose type is **rdfs:Class**. The following lines of Fabl introduce the type **Point** which was discussed earlier.

```
class('Point');
var xsd:double geom:xc;
var xsd:double geom:yc;
endClass();
```

The constructors **BagOf(T), ListOf(T), SeqOf(T),** and **Function(O,I_0,...,I_N)** generate parametric types denoting the set of all bags (resp. lists,sequences) with members in type **T**, and of all functions from input types **I_0**,.... **I_N** to output type **O**, respectively.

Except for the parametric types, any Fabl statement which introduces a type is equivalent to a set of DAML+OIL statements about the type. This was illustrated by the definition of **Point** which appeared earlier. Only a part of the DAML+OIL formalism is used for this purpose. A new Fabl class can be introduced by subclassing a **daml:Restriction**. Within the **daml:Restriction**, properties may be restricted either (1) by **daml:hasValue**, or (2) by **daml:toClass** with an optional **daml:maxCardinality** or **daml:cardinality** restriction with value 1. The effect is that properties may be assigned values, or may be assigned types. If a property is assigned a type, it may optionally be restricted to have either exactly one, or at most one value of that type. A new class may also be introduced as a **daml:intersectionOf** existing classes. Any DAML+OIL class may appear as a legal Fabl type, because *any* RDF type at all can so appear, but Fabl syntax will only generate types in the subset just described, and Fabl's type deduction mechanisms will not fully exploit available information in types outside the subset. Coverage of more of DAML+OIL can be implemented in future extensions of Fabl without disturbing the correctness of code written for the current subset.

Here are the details. A Fabl class definition starts with

```
class('classname');
```

and ends with

```
endClass();
```

Within the definition, statements of the form

```
var pathname = expression;
```

called an *assignment* and

```
var [qualifier] [type] pathname;
```

called an *assertion* may appear. The possible values of the optional qualifier are **exists**, **optional**, and **multivalued** (**exists** is the default). A pathname is a sequence of names of properties, separated by dots ("."). and represents sequential selection of the values of properties along the path. The assertion:

```
var [qualifier] type pathname;
```

means that, for all elements **X** of the class being defined, if **v** is a value of **X.pathname**, then **v** belongs to **type**. The qualifier **exists** (resp. **optional**, **multivalued**) means that **X.pathname** must have exactly one value (resp. at most one value, any number of values).

```
var [qualifier] pathname;
```

makes no claim about the type of **X.pathname**, only about the cardinality of the set of values which it assumes (depending on the qualifier).

The assignment

```
var pathname = expression;
```

means that the value of the slot denoted by the pathname is initialized to the value of the expression at the time when the member **X** is created, or when the class is installed (see below). Here are examples:

```
class('Rectangle');
var Point geom:center;
var geom:width;  //already declared to be a xsd:double in geom:
var geom:height; //already declared to be a xsd:double in geom:
endClass();

class('RedObject');
var color:color = color:red;
endClass();

class('RedRectangleCenteredOnXaxis');
var Rectangle && RedObject this;
var geom:center.geom:xc = 0.0;
endClass();
```

In the last example, The && operator denotes conjunction, and the pathname **this** refers to members of the class being defined, so that

```
var class this;
```

means that the class within whose definition the statement appears is a subclass of **class**.

The translation of Fabl class definitions into DAML+OIL RDF is straightforward. Assertions translate into **toClass** restrictions, and their qualifiers to **cardinality** or **maxCardinality** restrictions. Assignments translate into **hasValue** restrictions. The only minor complication is that, when pathnames of length greater than one appear, helper classes are automatically generated which express the constraints on intermediate values in path traversal (details ommitted).

9. Dynamic Installation of Classes

Recall that

```
x.rdf:type = C;
```

asserts that **x** belongs to daml:Class **C**. Such statements can be executed at any time, thereby dynamically adding class memberships. The effect of the statement is not just to add

the triple asserting class membership, but also to apply the constraints which **C** imposes on its members. Consider, for example:

```
var rect = new(Rectangle);

rect.rdf:type = RedRectangleCenteredOnXaxis;
```

Recall that **RedRectangleCenteredOnXaxis** asserts constant values for slots **geom:center.geom:xc** and **color:color**. Consequently, after the assertion that **x** belongs to this class, the following triples are added to the graph:

```
<x,rdf:type,RedRectangleCenteredOnXaxis>
<x,color:colorOf,color:red>,
<x,geom:center,center–uri>,
<center–uri,rdf:type,geom:Point>
<center–uri,geom:xc,0.0>
```

Here, **center–uri** represents an anonymous node which has been created to represent the value of the geom:center property of **x**.

10. Implementation of Parametric Types

Here is the definition of BagOf:

```
class('BagOf');
var daml:Class this;
var memberType;
endClass();

daml:Class function BagOf(rdfs:Class tp)
{
  var BagOf rs;
  rs = new(BagOf);
  rs . memberType = tp;
  rs . uri = 'nurl:fablParametricTypes/' + 'BagOf(' + uriEncode(tp.uri) +
')';
  return rs;
}
```

This definition appears within the Fabl language definition, where **memberType** has already been declared to be a **UniqueProperty** of type **rdfs:Class**. The operator **uriEncode** encodes reserved characters (such as ":" and "/") as described in the URI standard[11]. Note that **BagOf** implements a one–to–one map from the URIs of types **T** to the URIs of **BagOf(T)**. The implementation of the other parametric types **ListOF**, **SeqOf**, and **Function** are analogous. Fabl programmers can introduce their own parametric types using the same strategy.

11. Types of Fabl Expressions

Fabl is not just a language in which types may be created and manipulated, but a *typed* language in the more usual sense that each Fabl expression **E** is assigned an **rdfs:Class**. Of

course, any particular Fabl *value* (ie node in the active graph) may have arbitrarily many types, but a Fabl *expression* is assigned one of the types which the values of the expression is expected to assume. Types of function applications are deduced in the usual way. If a Fabl function **f** is defined by

```
O function f(I0 a0,... IN aN)
{
...
}
```

then the type of **f(i$_0$,...i$_N$)** is **O** if **i$_0$,...i$_N$** have types **I$_0$ ... I$_N$**. Range assertions are exploited in type deduction concerning property selections. If the triple

```
<P,rdfs:range,T>
```

is present in the active graph for property **P**, the expression

```
x.P;
```

is given type **BagOf(T)**, unless **P** is asserted to be univalued, in which case the type of **x.P** is **T**. (If more than one range type is assigned to **P**, this is equivalent to assigning the conjunction of the range types.)

```
E ~ T
```

performs a type cast of the expression **E** to type **T**. Type casts are checked at runtime: if the value of **E** does not lie in **T** when **E~T** is executed, an error is thrown. Simple coercion rules are also implemented; for example ints coerce to doubles, and conjunctions coerce to their conjuncts.

12. Functions and Methods

A function definition:

```
O function fname (I0 a0,... IN aN))
{
...
}
```

adds a function to the active graph under the decorated name

```
'f'+hash(uri_encode(I0.uri),...uri_encode(IN.uri),fname)+'_'+fname;
```

The purpose of decoration is to support polymorphism by assigning different URIs to functions whose input types differ. If the function definition appears within the scope of a class definition, the function is added beneath the URI of the class, and is invoked in the usual manner of methods: **<object>.fname(...)**. If preceded by the optional keyword **final** a method cannot be overridden. The effect of a Java abstract method is obtained by including a property of functional type in a class definition. Overriding of methods takes place as a side effect of class installation when the class being installed assigns values to functional properties. This simple treatment of method overriding is more flexible than conventional treatments; for

example, dynamic installation of classes may change the set of methods installed in an object at any time, not only at object-creation time as in Java or C++. These points are illustrated by examples just below. The Fabl expression:

```
f[I0,...In]
```

denotes the variant of **f** with the given input types. For example,**twice[SeqOf(xsd:int)]** denotes the variant of **twice** which takes sequences of ints as input. The Fabl operator:

```
supplyArguments(functionalValue,a0,...aN)
```

returns the function which results from fixing the first **N** arguments to **functionalValue** at the values of a_0,...a_N. Now, consider the following code:

```
class('Displayable');
var Function(fabl:void) Displayable\display;
...
endClass();
```

Note that by giving **Displayable\display** as the name of the functional property, we have allocated a URI for that property in the hierarchy beneath the URI for **Displayable**. This technique can be used in any context where a property which pertains to only one class is wanted. Consider also a concrete variant which displays rectangles:

```
fabl:void function display(Rectangle r)
{
....
}
```

Then, with

```
class('DisplayableRectangle');
var Displayable && Rectangle this;
var Displayable\display = supplyArguments(display[Rectangle],this);
endClass();
```

a class is defined which is a subclass of both **Rectangle** and **Displayable**, and which assigns concrete functions to the corresponding functional properties in the latter class. This is similar to what happens when a C++ or Java class contains a virtual method which is implemented by a method defined in a subclass. As noted earlier, the wiring of virtual methods to their implementations can only take place at object creation time in Java or C++, and cannot be undone thereafter, whereas Fabl allows wiring of functional properties to their implementations to take place at any time during a computation, via, for example

```
someRectanglePreviouslyUndisplayable.rdf:type = DisplayableRectangle;
```

Fabl supports assertion of constraints as part of class definitions - constraints which are applied to members at class installation time, and maintained thereafter by a constraint propagation mechanism. The constraint facility is beyond the scope of this paper.

13. Code as RDF

The foregoing discussion has described how Fabl data and types are rendered as sets of RDF triples. The remaining variety of Fabl entity which needs expression in RDF is *code*.

Code is represented by elements of the class **fabl:Xob** (**Xob** = "eXecutable object"). **Xob** has subclasses for representing the atoms of code (global variables, local variables, and constants), and for the supported control structures (blocks, if–else, loops, etc). Here is the class **Xob**:

```
class('Xob');
//atomic Xob classes such as Xlocal do not require flattening
var optional Function(Xob,Xob) Xob\flatten;
var rdfs:Class Xob\valueType;
endClass();
```

Subclasses of **Xob** include:

```
class('Xconstant'); //Constant appearing in code
var Xob this;
var Xconstant\value;
endClass();

class('Xlocal'); //Local variable
var Xob this;
var xsd:string Xlocal\name;
endClass();

class('Xif');
var Xob this;
var Xob Xif\condition;
var Xob Xif\true;
var optional Xob Xif\false;
endClass();

class('Xapply'); //application of a function to arguments
var Xob this;
var AnyFunction Xapply\function;
var SeqOf(Xob) Xapply\arguments;
endClass();
```

(The type **AnyFunction** represents the union of all of the function types **Function(O,I_0...I_N))** The Fabl statement

```
if (test(x)) action(2);
```

translates to the Xob given by this RDF:

```
<fabl:Xif>
   <fabl:Xif.condition>
      <fabl:Xapply>
          <fabl:Xapply.function rdf:resource="#f001a0e6f_test"/>
          <fabl:Xapply.arguments>
              <rdf:seq>
                  <rdf:li>
                      <fabl:Xlocal>
                          <fabl:Xlocal.name>x</fabl:Xlocal.name>
                      </fabl:Xlocal>
                  </rdf:li>
              </rdf:seq>
          </fabl:Xapply.arguments>
      </fabl:Xapply>
   </fabl:Xif.condition>

<fabl:Xif.true>
      <fabl:Xapply>
          <fabl:Xapply.function rdf:resource="#f001a0e6f_action"/>
          <fabl:Xapply.arguments>
              <rdf:seq>
                  <rdf:li>
                      <fabl:Xconstant Xconstant.value=2/>
                  </rdf:li>
              </rdf:seq>
          </fabl:Xapply.arguments>
      </fabl:Xapply>
   </fabl:Xif.true>
<fabl:Xif>
```

f001a0e6f_action is the decorated name of the variant of **action** which takes an **xsd:int** as input. Verbose as this is, it omits the **Xob** properties. Correcting this omission for the **Xlocal** would add the following lines in the scope of the **Xlocal** element:

```
<rdf:type rdf:resource = "http://purl.oclc.org/net/nurl/fabl/Xob"/>
<fabl:Xob.valueType rdf:resource =
"http://www.w3.org/2000/10/XMLSchema:int"/>
```

(The **Xob\flatten** property does not appear because **Xlocals** do not require flattening). A full exposition of the set of all Xob classes is beyond the scope of this paper, but the above examples should indicate the simple and direct approach taken. The class

```
class('Xfunction');
var xsd:string Xfunction\name;
var rdfs:Class Xfunction\returnType;
var SeqOf(Xlocal) Xfunction\parameters;
var SeqOf(Xlocal) Xfunction\localVariables;
var Xob Xfunction\code;
var SeqOf(xsd:byte) Xfunction\byteCode;
endClass();
```

defines an implementation of a function. When a Fabl function is defined, the code is analyzed, producing an **Xfunction** as result. This **Xfunction** is assigned as the value of the decorated name of the function.

The following steps are involved in translating the source code of a Fabl function or command into an **Xfunction**:

```
Source code [Parser] ->
Parse tree   [Analyzer] ->
Type–analyzed form (Xob) [Flattener]->
Flattened form (Xob) [Assembler] ->
Byte Code (executed by the Fabl virtual machine)
```

All of these steps are implemented in Fabl. The parse tree is a hierarchical list structure in the Lisp tradition whose leaves are tokens; a token in turn is a literal annotated by its syntactic category. A *flat* Xob is one in which all control structures have been unwound, resulting in a flat block of code whose only control primitives are conditional and unconditional jumps. Separating out flattening as a separate step in analysis supports extensibility by new control structures, as will be seen in a moment.

The analysis step is table driven: it is implemented by an extensible collection of *constructors* for individual tokens. The constructor property of a token is a function of type **Function(Xob,daml:List)** which, when supplied with a parse tree whose operator is the token, returns the analysis of that tree. Here is the code for the constructor for **if**. The parse of an **if** statement is a list of the form (if <**condition**> <**action**>).

```
Xob function if_tf(daml:List x)
  {
  var Xob cnd,ift,Xif rs;
  cnd = analyze(x[1]); //the condition
  if (cnd.Xob\valueType!=xsd:boolean) error('Test in IF not  boolean');
  ift = analyze(x[2]);
  rs = new(Xif);
  rs . Xif\condition = cnd;
  rs . Xif\true = ift; //no value need be assigned for  Xif\false
  return rs;
  }
```

x[N] selects the Nth element of the list. Then, the statement

```
ifToken.constructor = if_tf[daml:List];
```

assigns this function as the constructor for the **if** token. More than one constructor may be assigned to a token; each is tried in turn until one succeeds.

The **Xif** class, like other non–primitive control structures, includes a method for flattening away occurences of the class into a pattern of jumps and gotos (details omitted). Constructors and flattening methods rely on a library of utilities for manipulating **Xobs**, such as the function **metaApply**, which constructs an application of a function to arguments, and **metaCast** which yields a **Xob** with a different type, but representing the same computation, as its argument.

This simple architecture implements the whole of the Fabl language. The crucial aspect of the architecture is that it is fully open to extension within RDF. New control structures, type constructors, parametrically polymophic operators, annotations for purposes such as aspect–oriented programming[12], and other varieties of language features can all be introduced by loading RDF files containing their descriptions. The core Fabl implementation itself comes into being when the default configuration file loads the relevant RDF; a different configuration file drawing on a different supply of RDF would yield another variant of the language. This is the sense in which the implementation provides an open framework for describing computation in RDF, rather than a fixed language.

Finally, note once again that **Xobs** provide a formalism for representing computation in RDF which does not depend for its definition on any particular source language nor on any particular method for execution. That is, it formalizes computation *within* RDF, as promised by the title of the paper, and can yield the benefits sketched in the introduction.

14. Implementation

The practicality of an RDF–based computational formalism is a central issue for this paper, so size and performance data for our initial implementation are relevant.

The implementation consists of a small kernel written in C. The size of the kernel as a WinTel executable is 120 kilobytes. The kernel includes the byte–code interpreter, a generation–scavenging garbage collector, and a loader for our binary format for RDF. The remainder of the implementation consists of Fabl's RDF self description, which consumes 700 kilobytes in our RDF binary format. A compressed self–installing version of the implementation, which includes the Fabl self description, consumes 310 kilobytes. Startup time (that is, load of the Fabl self description) is about one third of a second on a 400MHZ Pentium II. Primitive benchmarks show performance similar to scripting languages such as JavaScript (as implemented in Internet Explorer 5.0 by Jscript) and Python. However, further work on performance should yield much better results, since the language is strongly typed, and amenable to many of the same performance techniques as Java.

The full value of formalizing computation within RDF will be realized only by an open standard. We regard Fabl as a proof–of–concept for such a formalization. In the context of a standards effort, we would be willing to contribute as Open Source whatever part of Fabl's implementation is found to be relevant.

15. Future Work

The current Fabl implementation treats the external RDF world as a store of RDF triple sets, which are activated explicitly via **loadRdf** or **getRdf**. However, an interesting direction for future work is the definition of a remote invocation mechanism for RDF–based computation. Here is an outline of one possibility.

Values of the **fabl:describedBy** property might include URLs which denote servers as well as RDF pages. In this case, **getRdf(U)** would not load any RDF. Instead, a connection would be made to the server (or servers) **S** designated by the value(s) of **fabl:describedBy** on **U**. In this scenario, the responsibility of **S** is to evaluate properties, globals, and functions in the URI hierarchy rooted at **U**. Whenever a property **E.P**, a global **G**, or a function application **F(x)** is evaluated in the client **C**, and the URI of **E**, **G**, or **F** is an extension of **U**, a request is forwarded to the server **S**, which performs the needed computation, and returns the result. The communication protocol would itself be RDF–based, along the lines proposed on the www–rdf–interest mailing list[13]. Such an approach would provide simple and transparent access to distributed computational resources to the programmer, while retaining full decoupling of description of computation in RDF from choices about source language and implementation.

16. Other XML Descriptions of Computation

Imperative computational constructs appear in several XML languages. Two prominent examples are SMIL[14] and XSLT[15], in which, for example, conditional execution of statements is represented by the **<switch>** and **<xsl:if>** tags, respectively. The aims of these formalizations are limited to specialized varieties of computation which the languages target. Scripting languages encoded in XML include XML Script[16] and XFA Script[17].

Footnotes

1. Fabl™ is a trademark of The Behavior Engine Company, and is pronounced "fable".
2. The standardization of JavaScript is ECMAscript[18]

References

[1] W3C RDF Model and Syntax Working Group. Resource Description Framework (RDF) Model and Syntax Specification, *http://www.w3.org/TR/REC–rdf–syntax/*, February 1999
[2] Ian Horrocks, Frank van Harmelen, Peter Patel–Schneider, eds. DAML+OIL (March 2001), *http://www.daml.org/2001/03/daml+oil–index*, March 2001
[3] Chris Waterson. RDF: back–end architecture, *http://www.mozilla.org/rdf/back–end–architecture.html*, August 1999
[4] Brian McBride. Jena - A Java API for RDF, *http://www–uk.hpl.hp.com/people/bwm/rdf/jena/index.htm*, May 2001 (last update)
[5] Tim Bray, Dave Hollander, Andrew Layman, eds. Namespaces in XML, *http://www.w3.org/TR/REC–xml–names/*, January, 1999
[6] The Dublin Core Metadata Initiative, *http://dublincore.org*, July 2001 (last update)
[7] The Behavior Engine Company, *http://www.behaviorengine.com*, July 2001 (last update)
[8] Ian Horrocks, Frank van Harmelen, Peter Patel–Schneider, eds. A Model–Theoretic Semantics for DAML+OIL (March 2001), *http://www.daml.org/2001/03/model–theoretic–semantics.html*, March 2001
[9] Dan Brickley, R. V. Guha, eds. Resource Description Framework (RDF) Schema Specification 1.0, *http://www.w3.org/TR/2000/CR–rdf–schema–20000327/*, March 2000
[10]Paul V. Biron, Ashok Malhotra, eds. XML Schema Part 2: Datatypes, *http://www.w3.org/TR/2001/REC–xmlschema–2–20010502/*, May 2001
[11] T. Berners–Lee. Uniform Resource Identifiers (URI): Generic Syntax, *http://www.ietf.org/rfc/rfc2396.txt*, August 1998
[12] Gregor Kiczales, John Lamping, Anurag Mendhekar and Chris Maeda, Cristina Lopes, Jean–Marc Loingtier and John Irwin. Aspect–Oriented Programming , 11th European Conference on Object–Oriented Programming, LNCS, vol. 1241, Springer Verlag, 1997
[13] Ken MacLeod, and respondents. Toying with an idea: RDF Protocol, *http://lists.w3.org/Archives/Public/www–rdf–interest/2001Mar/0196.html*, March 2001
[14] Jeff Ayars et al, eds. Synchronized Multimedia Integration Language (SMIL 2.0) Specification, *http://www.w3.org/TR/2001/WD–smil20–20010301/*, March 2001
[15] James Clark, ed. XSL Transformations (XSLT), *http://www.w3.org/TR/xslt*, November 1999
[16] DecisionSoft Limited, XML Script, *http://www.xmlscript.org/* May 2001
[17] XML For All, Inc. XFA Script, *http://www.xmlforall.com/cgi/xfa?XFAScript*, May 2001
[18] ECMA. Standard ECMA–262 – ECMAScript Language Specification, *http://www.ecma.ch/ecma1/stand/ecma–262.htm*, December 1999

Design Rationale of RuleML: A Markup Language for Semantic Web Rules

Harold Boley[1], Said Tabet[2] and Gerd Wagner[3]
1. DFKI GmbH, Erwin-Schrödinger-Straße D-67663 Kaiserslautern Germany
2. Nisus, Inc. 180, Turnpike Road Westboro, MA 01581 USA
3. Faculty of Technology Management Eindhoven University of Technology P.O. Box 513 5600 MB Eindhoven The Netherlands
boley@dfki.de, stabet@nisusinc.com, G.Wagner@tm.tue.nl

Abstract. This paper lays out the design rationale of RuleML, a rule markup language for the Semantic Web. We give an overview of the RuleML Initiative as a Web ontology effort. Subsequently, the modular syntax and semantics of RuleML and the current RuleML 0.8 DTDs are presented (focusing on the Datalog and URI sublanguages). Then we discuss negation handling, priorities/evidences, as well as agents and RuleML. We next proceed to RuleML implementations via XSLT and rule engines. In our conclusions, we continue to explore the bigger picture of ontologies and discuss some requirements for a future RuleML. An appendix shows our Semantic Web scenario in the insurance industry.

1. Introduction

Rules have traditionally been used in theoretical computer science, compiler technology, databases, logic programming, and AI. The Semantic Web tries to represent information in the World Wide Web such that it can be used by machines not just for display purposes, but for automation, integration, and reuse across applications; it has recently advanced to a W3C Activity. *Rule Markup* for the Semantic Web has been a hot topic since rules were identified as one of its Design Issues.

However, Semantic Web rules have been less systematically studied than the corresponding ontology (actually, taxonomy) markup. The Rule Markup Initiative tries to fill the gap by exploring rule systems (e.g., extended Horn logics) suitable for the Web, their (XML and RDF) syntax, semantics, tractability/efficiency, and transformation and compilation. Both derivation rules (which may be evaluated bottom-up as in deductive databases, top-down as in logic programming, or by tabled resolution as in XSB) (10) and reaction rules (also called "ECA" -- "event-condition-action" -- or "trigger" rules), as well as possible combinations, are being considered.

In the context of the Semantic Web, rules may be built on F-logic for RDF inference, as pioneered by SiLRI (4). This work has recently been extended for rules with expressive bodies (full FOL syntax) in TRIPLE (5). Rules may also be used to enhance the content of Web pages and XML documents in various ways. E.g., derivation rules allow the dynamic inclusion of derived facts, while reaction rules allow the specification of behavior in response to browser events.

RuleML started on the basis of pre-existing rule markup languages and has already inspired further rule-markup projects. As examples, we just sketch our RFML, URML, and AORML languages here, but refer readers to http://www.dfki.unikl.de/ruleml/#Participants for the complete picture:

- RFML (Relational-Functional Markup Language) is a (Web-)output format for relational-functional knowledge bases and computations implemented as part of the

Relfun system. The (Web-)input translation of RFML markup into Relfun's Prolog-like syntax is implemented via an XSLT stylesheet.

- URML was initially a project to Webize the ART and ARTScript Rule Language (11). URML is pushing the effort further to integrate Object Oriented Rule-based programming with XML and provide a basis for the implementation of Web objects and their manipulation in rules.
- AORML is a project to define a markup language for agent-oriented business rules in the context of Agent Object Relationship (AOR) models.

Participants in the RuleML Initiative have expressed an urgent need for a standard rule markup language, with translators in and out along with further tools. This need provided the impetus for the RuleML effort.

This paper lays out the design rationale of the Rule Markup Language (RuleML), the Initiative's evolving markup language for the Semantic Web. To accommodate the various (Web) rule-user communities from Knowledge-Based Systems to Intelligent Agents to E-Commerce, a modular hierarchy of sublanguages will be discussed. Rule extensions will concern first-class URIs, Web-suited negations, labelings, certainties/priorities, and packages. The Initiative also examines where current description methods and implementation techniques (e.g., XML DTDs vs. Schemas and C vs. Java-based rule engines) are sufficient for such rule markup and where they would need revisions and extensions.

This paper further attempts to contribute to some open issues of Notation 3 (N3) and DAML-Rules in relation to RuleML. Finally, by studying issues of combining rules and taxonomies via sorted logics, description logics, or frame systems, the paper also touches on the US-European proposal DAML+OIL.

2. The RuleML Initiative as a Web Ontology Effort

The RuleML Initiative started in August 2000 during the Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000). It has brought together expert teams from several countries, including leaders in Knowledge Representation and Markup Languages, from both academia and industry. The RuleML Initiative is developing an open, vendor neutral XML/RDF-based rule language. This will allow for the exchange of rules between various systems including distributed software components on the Web, heterogeneous client-server systems found within large corporations, etc. The RuleML language offers XML syntax for rules Knowledge Representation, interoperable among major commercial and non-commercial rules systems.

Among our industrial participants are rules engine vendors, Web technology vendors, XML/RDF tools vendors and also technology users such as financial corporations, telecom companies and some of the major Web portals and ASPs. The RuleML Initiative is collaborating with numerous related efforts such as the complementary Java Rules Engine API specification, the W3C RDF working group, the DAML group, W3C P3P Activity, PMML, and many others. This collaboration will enable RuleML to share mechanisms and provide a rules language to existing and emerging industry standards such as the Semantic Web and RDF, P3P, CC/PP and EDI (Electronic Data Interchange). The scenario in Appendix 1 exemplifies some inferential and metadata uses of RuleML for the Semantic Web.

Since RuleML participants organized a Birds Of a Feather (BOF) session at W3C's Technical Plenary and WG Meeting Event in February/March 2001, the Initiative has been discussing with W3C about possibilities of a working group devoted to Web rules (axioms) or to a combination of Web-ontology efforts as expressed by the 'equation' **ontology** = **taxonomy** + **axioms**. This would create the chance of a uniform ontology language with a description-logic taxonomy and Horn-logic-like rules.

In particular, large-scale RuleML rulebase exchange will require a taxonomy of the relations defined in the rulebase, where a relation with its arguments becomes a class with its slots. Participants in a rulebase exchange could then align each other's relation hierarchies to detect incompatibilities prior to merging and firing their rule definitions.

Conversely, large-scale DAML+OIL taxonomies will require a rule system to derive/use certain implicit information that is not captured by the taxonomy alone. The required rules could be marked up according to the suitable RuleML expressive class. DAML+OIL taxonomies and RuleML axioms should be expressed in compatible ways, ideally in one unified language. To achieve this, the current RuleML 0.8 and DAML+OIL markups could be co-developed in the Web Ontology Group towards a common version 1.0.

The fact that combined ontology systems quickly become undecidable is not a big issue since the higher RuleML expressive classes, e.g. Horn logic, are already undecidable. A big issue of the collaboration between DAML+OIL and RuleML, however, is the development of an *interleaved* layered system whose decidable taxonomy expressive classes interact well with the decidable or undecidable axiom expressive classes.

Initially, the possible combinations of taxonomies and axioms should be systematically compared w.r.t. criteria such as naturalness vs. formality, expressiveness vs. efficiency, DL terms as types for Horn variables vs. DL terms as Horn premises (or even conclusions), etc. On the taxonomy side, this comparison should span the range from order-sorted logics (which can be regarded as a degenerate description logic without slots, i.e. only the class lattice) to expressive decidable description logics such as *ALLNR*. On the axioms side, we should study the range from versions of Datalog, to Horn logic, to full first-order logic, and conservative extensions (e.g., restricted higher-order syntax). The question then is which of these respective subclasses go together well w.r.t. our criteria.

For example, it is well-known that order-sorted logics go together well with Horn logic and even with full first-order logic, as, e.g., shown by solutions to Schubert's Steamroller Problem such as (3): the combination reduces the search space. On the other hand, as shown by (9), only versions of Datalog seem to go together well with expressive description logics such as *ALLNR*: the combination enlarges the search space. If we allow free variations of both the taxonomy and axioms expressive classes, there are also many possible combinations in between. However, if a user community can state their requirements w.r.t. expressiveness of the taxonomy, the axioms, or both, it will be easier to fix the remaining degrees of freedom.

When building real Web ontologies it seems wise to start with less expressive classes on both the taxonomy and axioms sides, since a builder community cannot anticipate the requirements of future user communities. The ontological content should be packaged in an as lightweight ontology language as possible to make it available to a maximum number of users. The RuleML Initiative tried to prepare such a methodology through the bottom-up construction of a system of sublanguages from RDF-like triples to labeled Horn logic with equations plus URI individuals and relations. This could be complemented by a bottom-up taxonomy-language (re-)construction, and brought together through joint work on ontology layering.

3. The Modular Syntax and Semantics of RuleML

The modular RuleML design is described in this section. RuleML encompasses a hierarchy of rules, from reaction rules (event-condition-action rules), via integrity-constraint rules (consistency-maintenance rules) and derivation rules (implicational-inference rules), to facts (premiseless derivation rules). Till now, we have been mostly working on derivation rules and facts (cf. appendix 2).

The RuleML hierarchy of rules constitutes a partial order rooted in reaction rules. Its second main layer consists of, next to each other, integrity-constraint rules and derivation rules. The third layer just specializes derivation rules to facts. Thus, the global RuleML picture looks as shown in Figure 1.

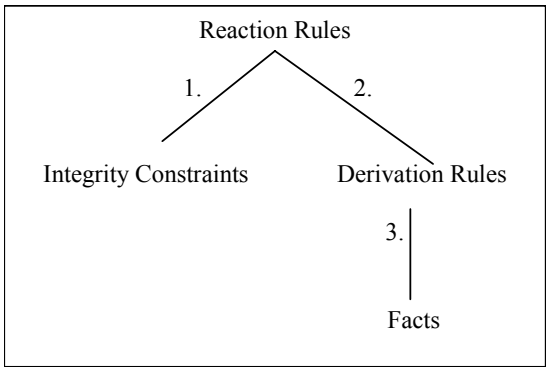


Figure 1: The RuleML hierarchy top-level.

Let us discuss the hierarchy's numbered specialization links in turn. (For a more fine-grained discussion of derivation rules, facts, and their further specialization to RDF triples see Figure 2.)

- Integrity constraints are considered as "denials" or special reaction rules whose only possible kind of action is to signal inconsistency when certain conditions are fulfilled.
- Derivation rules are considered as special reaction rules whose action happens to only add or 'assert' a conclusion when certain conditions (premises) are fulfilled. This asserting of conclusions can be regarded as a purely declarative step, as used for model generation and fixpoint semantics. Such rules can thus also be applied backward for proving a conclusion from premises.
- Facts are considered as special derivation rules that happen to have an empty (hence, 'true') conjunction of premises.

We can now make more precise our views regarding the application direction for the four rule categories:

- General reaction rules can only be applied in the forward direction in a natural fashion, observing/checking events/conditions and performing an action if and when all events/conditions have been perceived/fulfilled.
- Integrity constraints are usually also forward-oriented, i.e. triggered by updates, mainly for efficiency reasons.

- Derivation rules, on the other hand, can be applied in the forward direction as well as in a backward direction, the latter reducing the proof of a goal (conclusion) to proofs of all its subgoals (premises). Since in different situations different application directions of derivation rules may be optimal (forward, backward, or mixed), RuleML does not prescribe any one of these.
- For facts or 'unit clauses' it makes little sense to talk of an application direction.

While reaction rules, as the all-encompassing rule category, could implement all other ones, in RuleML we are introducing tailored special-purpose syntaxes for each of these categories. The following markup syntax only serves for our preliminary distinction of the four categories (for instance, we plan to permit **and**/**or** nestings besides flat conjunctions as premises):

- Reaction rules: **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head>** *action* **</_head> </rule>**
- Integrity constraints: **<ic> <_body> <and>** *prem1 ... premN* **</and> </_body> </ic>** implemented by **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head> <signal>** *inconsistency* **</signal> </_head> </rule>**
- Derivation rules: **<imp> <_head>** *conc* **</_head> <_body> <and>** *prem1 ... premN* **</and> </_body> </imp>** implemented by **<rule> <_body> <and>** *prem1 ... premN* **</and> </_body> <_head> <assert>** *conc* **</assert> </_head> </rule>**
- Facts: **<fact> <_head>** *conc* **</_head> </fact>** implemented by **<imp> <_head>** *conc* **</_head> <_body> <and> </and> </_body> </imp>**

Let us now elaborate on RuleML's derivation rules. Because of the infinity of possible rule-markup **syntaxes** and the rich previous work on **semantics** of rule-system classes, RuleML has attempted the following **separation of concerns**:

- **The sublanguage hierarchy**. Figure 2 shows the 12 sublanguages that together constitute the modularized basic RuleML definition. All sublanguages except the 'UR' (URL/URI) group correspond to well-known rule systems, where each sublanguage has a corresponding semantic (model- and proof-theoretic) characterization. Current work concerns a more precise URL/URI/URN semantics, as discussed in section The RuleML 0.8 DTDs. Sections Negation Handling in RuleML and Priorities/Evidences in RuleML prepare modular extensions of this basis for negations and priorities, respectively.
- **The concrete markup**. In recent months, the RuleML 0.7 DTDs have been developed into the RuleML 0.8 DTDs without affecting the above semantics. The new markup uses XML in RDF's 'explicit role-markup' style, relativizing XML's positionality to places where RDF's Seq containers or DAML+OIL lists would be needed. RuleML 0.8 is still being developed in DTDs, but will also be delivered (via translators) in XML Schemas. In the next section it will be illustrated with an earlier RuleML example, upgraded from 0.7 to 0.8.

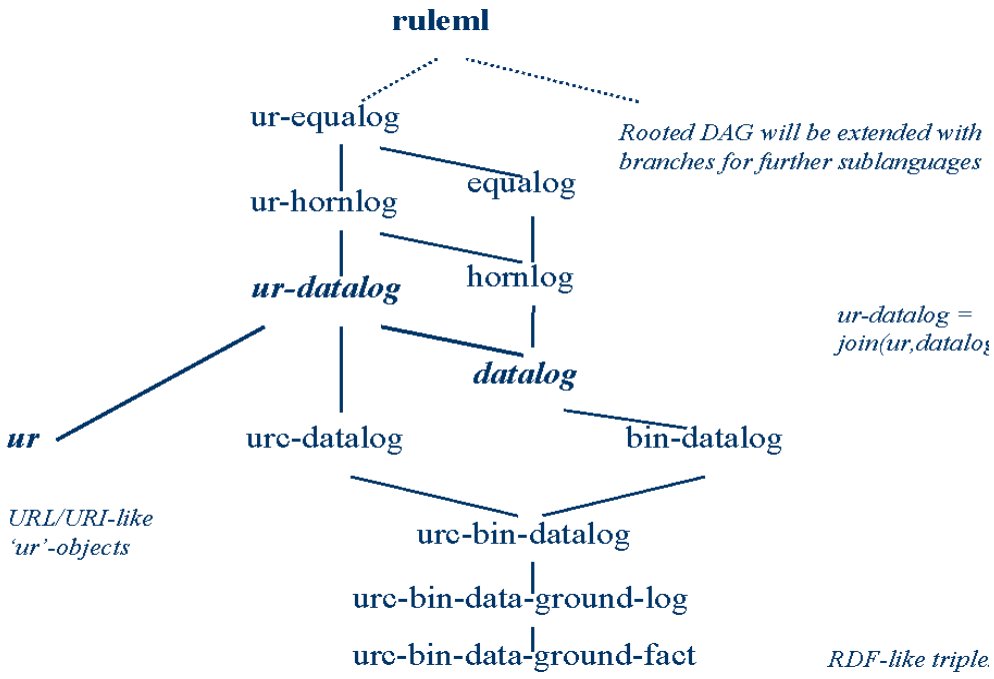


Figure 2: The RuleML hierarchy with 12 derivation-rule sublanguages.

4. The RuleML 0.8 DTDs

The upper layer of the RuleML hierarchy of rules is discussed in section The Modular Syntax and Semantics of RuleML. In that terminology, the system of RuleML DTDs presented here only covers derivation rules, not reaction rules.

This is because we think it is important to start with a subset of simple rules, test and refine our principal strategy using these, and then work 'up' to the more general categories of rules in the hierarchy. For this we choose Datalog, a language corresponding to relational databases (ground facts without complex domains or 'constructors') augmented by views (possibly recursive rules), and work a few steps upwards to further declarative rules from (equational) Horn logic. We also work upwards from a URL/URI language corresponding to simple objects. The join of both of these branches then permits inferences over RDF-like 'resources' and can be re-specialized to RDF triples.

Regarding the concrete markup syntax, we have been experimenting with several DTDs prior to the current, still preliminary, version. The rationale for our current tags is as follows.

- Rather than leaving conjunction implicit, an explicit tag pair **<and>** ... **</and>** with a sequence of N conjuncts is used (this would preferably be a set of conjuncts), preparing the unavoidable explicit markup of other boolean connectives (mainly **<or>** ... **</or>**) and their nesting.
- As a result of previous discussions, RuleML now uses an XML-RDF-unified data model with "Order-Labeled (OrdLab) Trees" (exemplified in appendix 3) as its notational base (2).
- In particular, we conventionally mark up RDF-like predicates, here called 'roles', by "_"-prefixed tags in XML (if all class-like 'type' tags would start with an upper-case

letter, then 'role' tags could also be distinguished, Java-like, by having them start with a lower-case letter, as in The FRODO rdf2java Tool).

- Using an **atom** (for a single premise) or an **and** (for a conjunction of premises) in the role of the body and an **atom**ic conclusion in the role of the head, rules aggregate two commutative roles; in particular, our Horn-like implication rules equivalently become **<imp>** **<_body>** **<and>** *prem1* ... *premN* **</and>** **</_body>** **<_head>** *conc* **</_head>** **</imp>** or become **<imp>** **<_head>** *conc* **</_head>** **<_body>** **<and>** *prem1* ... *premN* **</and>** **</_body>** **</imp>** (thus unifying KIF's "implication" and "reverse implication" syntaxes).
- The main advantage of roles is that of feature-term or **object-centered modeling**: If some extra information is to be added to an element such as a priority factor to the **imp** element, then it is easy to attach, RDF-like, a new **_priority** role with a **float**-type value; on the other hand the insertion, XML-like, of the **float**-type value directly into the child sequence would (be harder to read and) cause all subsequent children to assume a new position in the element (a problem for processing via XSLT etc.).
- In the new data model an element can have "mixed content" in the new sense of having both 'role' and 'type' children (see the **atom** examples below whose content consists of one _opr-role child and _1, _2, ... var-type children): while the 'type' children form an ordered sequence as in XML, without need for RDF's Sequence container, (1) the 'role' children are commutative as in RDF (treating an ordered sequence as a unit, as if it was reified into a Sequence container).

Appendix 2 contains a preliminary DTD for a Datalog subset of RuleML 0.8. Appendix 3 shows a simple example rule base that conforms to that DTD.

As indicated in Figure 2, besides the sublanguages towering above the Datalog DTD, there is another major RuleML branch consisting of the sublanguages on top of 'UR'-object (URL/URI) DTD. In RuleML we try to build on existing W3C work whenever possible. Hence, Uniform Resource Identifiers (URIs) are used to locate, describe and access resources and services such as classes, objects, software agents, Web components, Web services, etc. The representation of objects as URIs in RuleML will also facilitate the integration with related work on ontologies. Web objects and services use a URL/URI as their unique object identifier (cf. SHOE, RDF, URML) and the point of access to the Web (and in some cases standalone or intranet) resource or software agent. URLs/URIs can be embedded in facts, rule conditions and rule actions.

The RuleML language thus offers support for URIs in its system of DTDs starting from the 'UR' sublanguage. For example, in UR-Datalog, names can be assigned to individuals and relations using content markup and/or an URI attribute. The content markup need not be unique while the URI attribute is unique. The modular design of RuleML will allow us to extend URIs to a number of other addressing schemes.

As a simple Datalog example consider the facts in appendix 3, which use content markup to name, perhaps not uniquely, an individual book. Alternatively, in UR-Datalog the first of these facts, say, can use a URI under an **href** attribute of the empty **ind** element as follows:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <ind>John</ind>
      <ind>Mary</ind>
      <ind href="http://www.ibiblio.org/xml/books/bible2"/>
    </atom>
  </_head>
</fact>
```

Moreover, the second of these facts, say, can now combine the original content markup with the URI attribute as follows:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>keep</rel></_opr>
      <ind>Mary</ind>
      <ind href="http://www.ibiblio.org/xml/books/bible2">XMLBible</ind>
    </atom>
  </_head>
</fact>
```

It should be noted that, content markup not being unique, a given URI can be combined with different content markups in different elements. Thus, the second fact, say, could also use the same URI with this time an extended PCDATA **New XMLBible**. Conversely, of course, two different URIs can be combined with the same content markup.

5. Negation Handling in RuleML

In natural language, and in practical knowledge representation systems, such as the IBM business rule system CommonRules (6) that is based on the formalism of *extended logic programs*, there are two kinds of negation: a *weak* negation expressing *non-truth* (in the sense of "I don't like snow"), and a *strong* negation expressing explicit *falsity* (in the sense of "I dislike snow"). In RuleML, the weak negation connective is denoted by '*not*' and the strong negation connective by '*neg*'. In the case of a complete predicate, such as being an odd number, both negations collapse: 'not odd(x)' is equivalent to 'neg odd(x)', or in other words, the non-truth of the atom 'odd(x)' amounts to its falsity. In the case of an incomplete predicate, such as 'like', we only have that the strong negation implies the weak negation: 'neg like(I,snow)' implies 'not like(I,snow)', but not conversely. Also, while the double negation form 'neg not' collapses (according to partial logic, see [Wag98]), the double negation form 'not neg' does not collapse: not disliking snow does not amount to liking snow.

Using two kinds of negation in derivation rules has been proposed independently in (7) and (12). Rules with weak negation, or with other non-persistent connectives, lead to nonmonotonic inference. It is well-known that the semantics of nonmonotonic knowledge systems is not based on all models of a knowledge base but solely on the set of all *intended* models. E.g., for relational databases, which can be viewed as the most fundamental case of a knowledge system, the intended models are the minimal ones. The model-theoretic semantics of nonmonotonic rules is based on the concept of *stable (generated)* models in

classical and partial logic (see 7, 8 and 9). Notice that classical logic can be viewed as the degenerate case of partial logic when all predicates are total.

Under the preferential semantics of stable (generated) models, the weak negation 'not' corresponds to *negation-as-failure* in Prolog and to the *EXCEPT* operator in SQL in the following way: a query expression "give me all objects x such that 'p(x) and not q(x)'" corresponds to the SQL expression 'P EXCEPT Q' where P and Q denote the tables that represent the extensions of the predicates p and q. Since SQL tables were not intended to be able to represent incomplete predicates, SQL does not contain a strong negation operator.

Because in many computational domains predicates are assumed to be complete (according to the Closed-World Assumption), 'not' is used more frequently than 'neg'. An example of a rule that defines a derived attribute of a certain class in a UML class model is the following: A car is available for rental if it is physically present, is not assigned to any rental order, is not scheduled for service, and does not require service. This rule defines the derived Boolean attribute 'isAvailable' of the class 'RentalCar' by means of the stored Boolean attributes 'isPresent', 'requiresService', 'isScheduledForService', and an association 'isAssignedTo' between cars and rental orders, here called '`isAssignedToRentalOrder`'. The association is shown more explicitly in the UML class model of Figure 3.

```
<imp>
  <_head>
    <atom>
      <_opr><rel>isAvailable</rel></_opr>
      <var>Car</var>
    </atom>
  </_head>
  <_body>
     <and>
      <atom>
        <_opr><rel>isPresent</rel></_opr>
        <var>Car</var>
      </atom>
      <not>
        <atom>
          <_opr><rel>isAssignedToRentalOrder</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
      <not>
        <atom>
          <_opr><rel>isScheduledForService</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
      <not>
        <atom>
          <_opr><rel>requiresService</rel></_opr>
          <var>Car</var>
        </atom>
      </not>
    </and>
  </_body>
</imp>
```

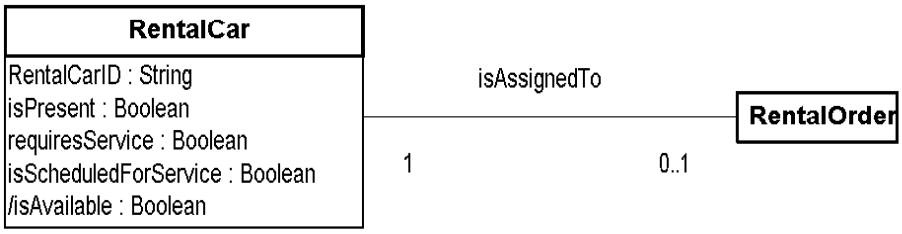


Figure 3: A UML model of the class *RentalCar* with the derived Boolean attribute */isAvailable*.

The strong negation is an "open world" negation, since in an open world such as the Web, the non-truth (or failure) of a statement does not imply its falsity. By combining weak and strong negation, one can express default rules (in the sense of Reiter's default logic) in a natural way. An example of this is the rule "a document that is not classified as being official has normally to be treated as an unofficial document". Such a rule could, for instance, supplement an ontology about enterprise documents and help answering queries about unofficial documents. Let us assume that EEEBizz classifies documents by means of a 'full'/'partial'/'open'-valued *Approval* property, while EEEComm classifies documents with the help of a 'yes'/'no'-valued *Released* property. Then, we may want to use a rule that allows to conclude a strongly negated atom on the basis of either of two weakly negated atoms (the **`or`** in the **`_body`** could be eliminated via separate rules for the disjuncts):

```
<imp>
  <_head>
    <neg>
      <atom>
        <_opr><rel>isOfficialDocument</rel></_opr>
        <var>DocumentName</var>
      </atom>
    </neg>
  </_head>
  <_body><or>
      <not>
        <atom>
          <_opr>
               <rel href="http://www.eeebizz.com/rdf sch#Approval"/>
          </_opr>
          <var>DocumentName</var>
          <ind>full</ind>
        </atom>
      </not>
      <not>
        <atom>
          <_opr>
               <rel href="http://www.eeecomm.net/rdf-voc#Released"/>
          </_opr>
          <var>DocumentName</var>
          <ind>yes</ind>
        </atom>
      </not>
    </or></_body>
</imp>
```

Notice that this rule allows to conclude that a document is unofficial unless the contrary is known. Therefore, it would provide the conclusion that a certain document is unofficial even if it suggests to be official (at its own URI) but is not classified properly (at the metadata's URI). This rule cannot be applied if there is an explicit 'full Approval'

classification and an explicit 'yes, Released' classification of the document (according to the respective definitions of EEEBizz and EEEComm).

For example, suppose the metadata consist only of the following 'full Approval' fact, an RDF triple (according to the URC-bin-data-ground-fact DTD of RuleML, cf. Figure 2) about a joint-mission document:

```
<fact>
  <_head>
    <atom>
      <_opr><rel href="http://www.eeebizz.com/rdf-sch#Approval"/></_opr>
      <ind href="http://www.eeebico.org/docs/joint-mission.html"/>
      <ind>full</ind>
    </atom>
  </_head>
</fact>
```

The first disjunct is false since its 'Approval' atom unifies with the fact (via the binding of **<var>DocumentName</var>** to **<ind href="http://www.eeebico.org/docs/joint-mission.html"/>**); but the second disjunct is true for lack of a corresponding 'yes, Released' fact; so the default rule classifies the document as unofficial.

6. Priorities/Evidences in RuleML

The following is an example using an auto insurance scenario. This example involves two conflicting rules, shown below. The first rule, which applies to drivers under 25 years of age, states that after the accident, the premium will increase by 40%. On the other hand, in the second rule, because the customer is on the family plan, his or her premium will not increase after the first accident. This example is treated in more detail in appendix 1.

The first rule, applicable to drivers under 25:

```
<imp>
  <_rlab><ind>beginners</ind></_rlab>
  <_spriority><ind>0.75</ind></_spriority>
  <_head>
        .......
  </_head>
  <_body>
    <and>
        ...
      <atom>
        <_opr><rel>customerUnder25</rel></_opr>
        <var>customer</var>
      </atom>
    </and>
  </_body>
</imp>
```

The second rule, applicable to drivers with a family plan:

```
<imp>
  <_rlab><ind>family</ind>  </_rlab>
  <_spriority><ind>0.9</ind> </_spriority>
  <_head>
        .......
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>FamilyAutoPlan</rel></_opr>
        <var>customer</var>
        <var>familyauto</var>
      </atom>
    </and>
  </_body>
</imp>
```

Both research prototypes and commercial rules engines offer a facility for controlling rule execution and conflict resolution. In RuleML, one can define either quantitative priorities declaring a numerical *Priority* property for rules or qualitative priorities using *Overrides* facts over rule labels.

A quantitative priority is a numerical value indicating the salience (or the evidence) of a rule. We consider supporting both static and dynamic salience. A static priority value can be represented by a constant or a variable. A dynamic salience is represented using a variable or a function or relation call: the numerical value is calculated at runtime from the current binding environment.

Qualitative priorities are represented using facts comparing rule labels. This approach is influenced by the rules conflict handling in BRML, based on partially-ordered prioritization information (6). Qualitative priorities using the *Overrides* fact can be generated from numerical saliences. For example, in the auto insurance example above, since rule labeled 'family' (salience 0.9) is higher priority than rule labeled 'beginners' (salience 0.75), we can generate the following qualitative priority fact: ***Overrides(family, beginners)***, which means that rule family will always win if it enters in a conflict with rule beginners.

7. Agents and RuleML

Biological and artificial systems that interact with their (natural or virtual) environment on the basis of their *mental state*, and exhibit some degree of autonomy, are called "agents". The most basic mental components of an agent are its *perceptions* of events (in the form of incoming messages) and its *beliefs* (or knowledge). Further important components are

- *memory* about past events and actions,
- *commitments* towards other agents to perform certain actions,
- *claims* against other agents,
- *goals* in the form of state conditions to be achieved by means of planning and plan execution, and
- *intentions* in the form of action plans that have been chosen to be executed.

A sophisticated software agent may be specified by

- an RDFS-based taxononmy for defining the schema of its mental state,
- a set of RDF facts for specifying its factual (extensional) knowledge,
- a set of RuleML integrity constraints for excluding non-admissible mental states,
- a set of RuleML derivation rules for specifying its terminological and heuristic (intensional) knowledge, and
- a set of RuleML reaction rules for specifying its behavior in response to communication and environment events.

Thus, it will be possible to completely specify a software agent using RDF/RDFS and RuleML. Executing such an agent specification requires a combination of a knowledge subsystem (including an inference and an update operation), a perception (or incoming message handling) subsystem and an action (or outgoing message handling) subsystem.

Michael Sintek has recently implemented a much simpler first example of a RuleML querying agent. This is a servlet (running in Tomcat) that receives RuleML rulebases in an

RDF-based RuleML syntax (since it uses The FRODO rdf2java Tool) together with some queries, evaluates them with XSB Prolog (in auto-tabling mode, which should be equivalent to bottom-up evaluation), and returns the result as an HTML page containing the bindings as facts of instantiated queries. A future version must, of course, return a RuleML file. It can be tried at this URL: Click on 'example' and paste the RDF RuleML popping up into the input window (note that pasting XML/RDF cannot be directly done in IE, only in Netscape; use "view source" in IE). Alternatively, you can use the Prolog parser and RDF translator to generate the RDF RuleML. Since it cannot be guaranteed that the above URLs will always work (server reboots etc.), this picture shows the agent in action.

8. RuleML Implementations via XSLT and Rule Engines

XSLT can itself be regarded as a rule-based programming language operating on XML elements. These elements can also be other rules expressed in XML. The RuleML Initiative has been implementing the translation between various rules systems using XSLT stylesheets. The first XSLT stylesheet from RuleML to another system demonstrated the translation of RuleML 0.7 to RDF; it can be seen as a preparation of our transition towards the current more RDF-oriented RuleML 0.8.

One of the most popular (reaction) rule engines currently available free for non-commercial use is JESS (Java Expert System Shell). Jess is implemented in the Java language. It was originally inspired by the CLIPS expert system shell, but has grown into a complete, distinct rule-based tool of its own. CLIPS is a development environment for rule-based and object oriented expert systems. CLIPS is being used by government agencies, research laboratories and universities as well as a number of companies around the world.

Following the release of RuleML 0.8, we will provide an XSLT style sheet that produces Jess code. A style sheet already exists for RuleML 0.7, compatible with Jess 60a5.

The example below shows a RuleML 0.8 rule originally authored using RuleML 0.7 and translated into Jess using an XSLT stylesheet. This kind of process can be automated easily in a Web-based platform using existing XML and XSLT tools and APIs. The same rule is translated into Prolog. This demonstrates the flexibility and the power of the rules exchange mechanism offered in RuleML.

The Rule written in RuleML:

```
<rulebase label="myRules">
  <imp>
    <_head><atom>
        <rel>likes</rel>
        <ind>John</ind>
        <var>x</var>
    </atom></_head>
    <_body><atom>
        <rel>likes</rel>
        <var>x</var>
        <ind>wine</ind>
      </atom></_body>
  </imp>
</rulebase>
```

The transformation to Jess gives the following Jess (and CLIPS) rule:

```
        (defrule myRules-1
                 "This rule has been generated from RuleML"
                 (likes ?x wine)
             =>
                 (likes John ?x))
```

and the transformation to Prolog returns the rule:

```
likes(John, X) :- likes(X, Wine).
```

With GEDCOM, Mike Dean created the first operational RuleML (0.7) rulebase, where rules on family relationships (child, spouse, etc.) are run via XSLT translators to the XSB, JESS, and n3/cwm engines. Besides indirectly, via translators, RuleML implementations should also be done directly, via rule engines.

With Mandarax RuleML, Jens Dietrich has implemented the first complete input-processing-output environment for RuleML (0.8). For a RuleML 0.8 engine we also cooperate with the CommonRules, Euler, and TRIPLE projects and hope to also join forces with W3C's N3 and NILE efforts, and with further interested companies.

9. Conclusions

Looking at the bigger picture of "ontologies", we will now discuss three related requirements for future RuleML versions.

1) Following our earlier 'taxonomy-plus-axioms' notion of "ontology", RuleML, together with DAML-Rules and Euler, can be seen as the "axioms part" working on the "taxonomy part" developed by some other effort such as DAML+OIL. Derivation rules are normally used in the context of an information model, such as a UML class model, an RDFS-based taxonomy (as used in DAML+OIL ontologies), or a predicate logic signature. The underlying information model defines a language for expressing logical statements that can play the role of an assertion, of a query, or of a condition. It should be possible to include a RuleML rulebase (or a reference to a RuleML document) within an XML-based version of an information model (such as a UMI document or a RDFS-based taxononmy). Vice versa, it should be possible to include (a reference to) such an information model within the XML-based RuleML rulebase. Ideally, a 'taxonomy-plus-axioms' ontology should include both parts on the same level, as pioneered by SHOE and N3.

Implied requirement for RuleML: A RuleML rulebase can either be embedded in an information model, or its top-level element ("rulebase") can have an attribute that specifies its context by refering to a respective XML document.

2) We could link to UML classes via RuleML variables: **<var>** could have an attribute giving the class constraining it. Also, a DAML+OIL taxonomy could be linked in such a "sorted logic" manner. We could additionally allow to plug in some other atom-defining formalism as an option. The "atoms" used in the premise and conclusion of a derivation rule in the context of a UML class model would then be expressed in OCL. The "atoms" used in the premise and conclusion of a derivation rule in the context of a DAML+OIL taxononmy would then be expressed in DAML+OIL RDF.

Implied requirement for RuleML: A separation of concerns: the proper rule language is more concerned with sentential connectives and rule keywords, than with the language of "atoms". The language of "atoms" can be called the content language of a RuleML rulebase. It consists of two layered sublanguages: 1) the predefined constructs of the chosen metamodel (like UML or RDFS), and 2) the terms defined by the chosen model/taxononmy.

3) Derivation rules operate on facts that are typically represented in a database, or in an XML or RDF document.

Implied requirement for RuleML: It should be possible to include a RuleML rulebase (or a reference to a RuleML document) within an XML or RDF document. Technically, it is easy to mix RuleML and other XML namespaces (like for, say, MathML), incl. RDF(S) namespaces. For this we assume a **ruleml:** namespace prefix.

Acknowledgements

Benjamin Grosof, MIT Sloan School of Management, continues to make significant contributions to RuleML, in particular to its DTDs. Michael Sintek, DFKI, provided a lot of help and devised the first RuleML querying agent. We acknowledge the remarks of the SWWS reviewers, which have helped us to improve this paper. Thanks go also to the EU for funding Harold Boley's research on collaborative Web technologies within the Clockwork project. The RuleML Initiative has been supported in part by Nisus, Inc. and DFKI who funded Said Tabet's and Harold Boley's active participation and the inception of the RuleML effort.

References

[1] Harold Boley. Relationships Between Logic Programming and RDF. *Proc. 1st Pacific Rim International Workshop on Intelligent Information Agents (PRIIA 2000)*, University of Melbourne, Australia, 2000; LNAI volume to be published.

[2] Harold Boley. A Web Data Model Unifying XML and RDF. *Working Draft*, DFKI Kaiserslautern, July 2001.

[3] A.G. Cohn. On the Solution of Schubert's Steamroller in Many Sorted Logic. *IJCAI-85*, pages 1169--1174, 1985.

[4] Stefan Decker and Dan Brickley and Janne Saarela and Juergen Angele. A Query and Inference Service for RDF. *QL'98 - The Query Languages Workshop*, World Wide Web Consortium, 1998.

[5] Stefan Decker and Michael Sintek. Triple. *RDF Interest Group: Face to face meeting*, World Wide Web Consortium, 2001.

[6] B.N. Grosof. Prioritized Conflict Handling for Logic Programs. *Proc. of the Int. Symposium on Logic Programming (ILPS-97)*, edited by Jan Maluszynski, MIT Press, Cambridge, MA, USA, 1997.

[7] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. *Proc. of Int. Conf. on Logic Programming (ICLP'90)*, MIT Press, 1990.

[8] H. Herre and G. Wagner. Stable Models are Generated by a Stable Chain. *J. of Logic Programming* 30:2 (1997), pages 165-177.

[9] A. Levy and M.-C. Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence Journal* 104, September 1998.

[10] K. Sagonas and T. Swift. An Abstract Machine for Tabled Execution of Fixed-Order Stratified Logic Programs. *ACM Transactions on Programming Languages and Systems* 20:3, pages 586-634, May 1998.

[11] S. Tabet, P. Bhogaraju and D. Ash. Using XML as a Language Interface for AI Applications. *Proceedings of the Symposium on the Application of Artificial Intelligence in Industry*, pages 133-142, Sixth Pacific Rim International Conference on Artificial Intelligence, Melbourne, Australia, August, 2000.

[12] G. Wagner. A Database Needs Two Kinds of Negation. In B. Thalheim and H.-D. Gerhardt (eds.), *Proc. of the 3rd Symp. on Mathematical Fundamentals of Database and Knowledge Base Systems (MFDBS'91)*, LNCS 495, pages 357-371, Springer-Verlag, 1991.

[13] G. Wagner. *Foundations of Knowledge Systems - with Applications to Databases and Agents*. Kluwer Acadamic Publishers, 1998.

Appendix 1: A Semantic Web Scenario in the Insurance Industry

In this appendix, we provide a Semantic Web scenario applying RuleML in a common pragmatic situation. After a car accident, one of the questions people are facing is: *how much will my premiums increase and how does this accident affect my insurance policy?* Not all insurance companies follow the same rules or apply the same formula. In the USA this results in premium increases that can vary from hundreds of dollars to over a thousand. Many companies follow the Insurance Services Office (ISO) standard of increasing your premium by 40 percent of their "base rate" after your first at-fault accident. A base rate is the average amount of all claims paid, plus the insurance company's processing fee. For example, if your company's base rate is $600, your premium after the accident will go up by $240.

In our scenario, Olivia is a teenager who unfortunately has just had her first car accident. She is insured on her mother's premium family insurance plan. This situation involves two conflicting rules, as formalized in RuleML below. The first rule, which applies to drivers under 25 years of age, states that after the accident, Olivia's premium will increase by 40%. On the other hand, the second rule, applying to drivers on a family plan, states that her premium will not increase at all after her first accident.

The first rule, applicable to drivers under 25:

```
<imp>
  <_rlab><ind>beginners</ind></_rlab>
  <_spriority><ind>0.75</ind></_spriority>
  <_head>
    <atom>
      <_opr><rel>calculatePremium</rel></_opr>
      <var>customer</var>
      <ind>40</ind>
    </atom>
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>InsurancePolicy</rel></_opr>
        <var>customer</var>
        <var>insurance</var>
      </atom>
      <atom>
        <_opr><rel>lifeEvent</rel></_opr>
        <var>customer</var>
        <ind>accident</ind>
        <var>report</var>
      </atom>
      <atom>
        <_opr><rel>customerUnder25</rel></_opr>
        <var>customer</var>
      </atom>
    </and>
  </_body>
</imp>
```

The second rule, applicable to drivers with a family plan:

```
<imp>
  <_rlab><ind>family</ind></_rlab>
  <_spriority><ind>0.9</ind></_spriority>
  <_head>
    <atom>
      <_opr><rel>calculatePremium</rel></_opr>
      <var>customer</var>
      <ind>0</ind>
    </atom>
  </_head>
  <_body>
    <and>
      <atom>
        <_opr><rel>InsurancePolicy</rel></_opr>
        <var>customer</var>
        <var>insurance</var>
      </atom>
      <atom>
        <_opr><rel>lifeEvent</rel></_opr>
        <var>customer</var>
        <ind>accident</ind>
        <var>report</var>
      </atom>
      <atom>
        <_opr><rel>FamilyAutoPlan</rel></_opr>
        <var>customer</var>
        <var>familyauto</var>
      </atom>
    </and>
  </_body>
</imp>
```

Let us now turn to formalizing the relevant facts.

Olivia is under 25:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>customerUnder25</rel></_opr>
      <ind>Olivia</ind>
    </atom>
  </_head>
</fact>
```

The following RDF-like RuleML facts permit to prove further premises of the above rules and also provide metadata descriptions for the required documents referenced and retrieved by URIs.

Olivia has an insurance policy and this document has link **`.../IMA-0835`**:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>InsurancePolicy</rel></_opr>
      <ind>Olivia</ind>
      <ind href="http://www.BostonInsurance.com/policy/IMA-0835"/>
    </atom>
  </_head>
</fact>
```

Olivia is in a family auto plan and this document has link **.../FMA-0142**:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>FamilyAutoPlan</rel></_opr>
      <ind>Olivia</ind>
      <ind href="http://www.BostonInsurance.com/plan/FMA-0142"/>
    </atom>
  </_head>
</fact>
```

Olivia's accident report is available at TrafficReport.biz:

```
<fact>
  <_head>
    <atom>
      <_opr><rel>lifeEvent</rel></_opr>
      <ind>Olivia</ind>
      <ind>accident</ind>
      <ind href="http://www.TrafficReport.biz/MA/report0712"/>
    </atom>
  </_head>
</fact>
```

The 'metafact' below is used to resolve the conflict between rule **beginners** and rule **family** (cf. section Priorities/Evidences in RuleML):

```
<fact>
  <_head>
    <atom>
      <_opr><rel>Overrides</rel></_opr>
      <ind>family</ind>
      <ind>beginners</ind>
    </atom>
  </_head>
</fact>
```

The rulebase presented in this example illustrates the use of Web-based documents in rules for matching and inferencing. In this example, we also show how priorities can be applied to rules. The **Overrides** fact above will allow rule **family** to fire as a higher priority rule and save Olivia a good amount of money: her premium will not increase.

Appendix 2: DTD for a Datalog Subset of RuleML

```
<!-- An XML DTD for a Datalog RuleML Sublanguage: Monolith Version -->
<!-- Last Modification: 2001-07-07 -->

<!-- ELEMENT Declarations -->

<!-- 'rulebase' root element uses 'imp' rules and 'fact' assertions on top-level -->
<!ELEMENT rulebase ((imp | fact)*)>

<!-- 'imp' rules are usable as general implications on the top-level -->
<!-- 'imp' element uses a conclusion role _head before a premise role _body, or -->
<!--  uses a premise role _body before a conclusion role _head -->
<!ELEMENT imp ((_head, _body) | (_body, _head))>

<!-- 'fact' assertions are usable as degenerate rules on the top-level -->
<!-- 'fact' element uses just a conclusion role _head -->
<!-- "<fact>_head</fact>" stands for "_head is implied by true" -->
<!ELEMENT fact (_head) >

<!-- _head role is usable within 'imp' rules and 'fact' assertions -->
<!-- _body role is usable within 'imp' rules -->
<!-- _head uses an atomic formula -->
<!-- _body uses an atomic formula or an 'and' -->
<!ELEMENT _head (atom)>
<!ELEMENT _body (atom | and)>

<!-- an 'and' is usable within _body's -->
<!-- 'and' uses zero or more atomic formulas -->
<!-- "<and>atom</and>" is equivalent to "atom"-->
<!-- "<and></and>" is equivalent to "true"-->
<!ELEMENT and (atom*)>

<!-- atomic formulas are usable within _head's, _body's, and 'and's -->
<!-- atom element uses an: -->
<!-- _opr ("operator of relations") role followed by zero or more arguments, or -->
<!-- one or more argument followed by an _opr role -->
<!-- the arguments may be ind(ividual)s or var(iable)s -->
<!ELEMENT atom ((_opr, (ind | var)*) | ((ind | var)+, _opr))>

<!-- _opr is usable within atoms -->
<!-- _opr uses rel(ation) symbol -->
<!ELEMENT _opr (rel)>

<!-- there is one kind of fixed argument -->
<!-- individual constant, as in predicate logic -->
<!ELEMENT ind  (#PCDATA)>

<!-- there is one kind of variable argument -->
<!-- logical variable, as in logic programming -->
<!ELEMENT var  (#PCDATA)>

<!-- there are only fixed (first-order) relations -->
<!-- relation or predicate symbol -->

<!ELEMENT rel  (#PCDATA)>
```

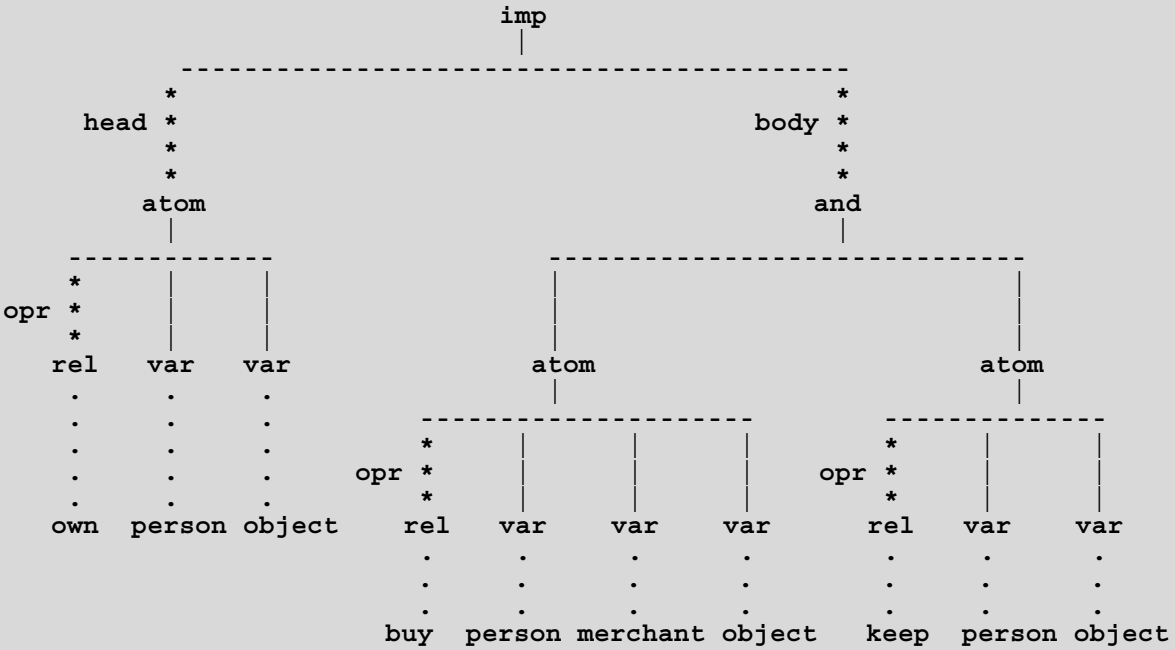
Appendix 3: Example RuleML Document: A Rulebase `own.ruleml`

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE rulebase SYSTEM "http://www.dfki.de/ruleml/dtd/0.8/ruleml-datalog-monolith.dtd">
<rulebase>

<!-- start XML comment ...
This example rulebase contains four rules. The first and second rules are implications; the third and
fourth ones are facts.

The first rule implies that a person owns an object if that person buys the object from a merchant and
the person keeps the object.

As an OrdLab Tree:
```




```
... end XML comment -->

<imp>
 <_head>
  <atom>
   <_opr><rel>own</rel></_opr>
   <var>person</var>
   <var>object</var>
  </atom>
 </_head>
 <_body>
  <!-- explicit 'and' -->
  <and>
   <atom>
    <_opr><rel>buy</rel></_opr>
    <var>person</var>
    <var>merchant</var>
    <var>object</var>
   </atom>
   <atom>
    <_opr><rel>keep</rel></_opr>
    <var>person</var>
    <var>object</var>
   </atom>
  </and>
```

```
  </_body>
</imp>


<!-- The second rule implies that a person buys an object from a merchant
if the merchant sells the object to the person. -->

<imp>
  <_head>
    <atom>
      <_opr><rel>buy</rel></_opr>
      <var>person</var>
      <var>merchant</var>
      <var>object</var>
    </atom>
  </_head>
  <_body>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <var>merchant</var>
      <var>person</var>
      <var>object</var>
    </atom>
  </_body>
</imp>

<!-- The third rule is a fact that asserts that John sells XMLBible to Mary. -->

<fact>
  <_head>
    <atom>
      <_opr><rel>sell</rel></_opr>
      <ind>John</ind>
      <ind>Mary</ind>
      <ind>XMLBible</ind>
    </atom>
  </_head>
</fact>

<!-- The fourth rule is a fact that asserts that Mary keeps XMLBible.

Observe that this fact is binary - i.e., there are two arguments for the relation. RDF viewed as a logical
knowledge representation is, likewise, binary, although its arguments have type restrictions,
e.g., the first must be a resource (basically, a URI). Some of the DTD's on the RuleML website handle
URL's/URI's (UR's); see especially urc-datalog.dtd for inferencing with RDF-like facts

-->

<fact>
  <_head>
    <atom>
      <_opr><rel>keep</rel></_opr>
      <ind>Mary</ind>
      <ind>XMLBible</ind>
    </atom>
  </_head>
</fact>
</rulebase>
```

Enabling Semantic Web Programming by Integrating RDF and Common Lisp

Ora Lassila
Nokia Research Center, 5 Wayside Road, Burlington, Massachusetts, USA

Abstract: This paper introduces "Wilbur", an RDF and DAML toolkit implemented in Common Lisp. Wilbur exposes the RDF data model as a frame-based representation system; an object-oriented view of frames is adopted, and RDF data is integrated with the host language by addressing issues of input/output, data structure compatibility, and error signaling. Through seamless integration we have achieved a programming system well suited for building "Semantic Web" applications.

1. Introduction

Common Lisp [24] is a programming language that has enjoyed great popularity in the AI community. Despite its somewhat waning use, it can still be considered one of the most expressive mainstream programming languages. Because of its somewhat unique integration of rich data structures with the language itself, Common Lisp offers the interesting possibility of integrating RDF [18, 19] and DAML [10, 11] data with a programming language, therefore making it easier to build software that takes advantage of the "Semantic Web" [3].

This paper will discuss "Wilbur", a Common Lisp -based open source toolkit for RDF and DAML. Wilbur includes an API (Application Programming Interface) which allows the underlying RDF data to be treated as a frame system, essentially providing an object-oriented view of the data. The relationship between frame-based representation, object-oriented modeling, and RDF is straightforward [20], but an even more interesting aspect is the synergistic potential of integrating a programming language with a frame system [15]. Many frame systems have offered some type of programming support such as access-oriented behavior [e.g., 13, pp.30-32] or some other type of "slot daemons" (for example, both CRL and KEE allowed a Lisp function to be invoked when certain operations were being performed on a slot). Tight integration, however, would in practice have to involve not only integration of the frame system's and the programming language's type systems, but also leveraging the programming language's native programming model and facilities (such as method invocation).

2. RDF Toolkits

RDF data consists of nodes and attached attribute/value pairs. Nodes can be any Web resources, including other RDF nodes. Attributes are named properties of nodes, and their values are either atomic (text strings) or other nodes. The essence of RDF is this model of nodes, properties and their values. In addition to the node-centric view the RDF model can be seen as

directed, labeled graphs (DLGs). The nodes are the vertices of a graph, and the properties name the edges. Therefore, if X has a property Y with the value Z, we can think of X and Z linked by an edge labeled Y, pointing from X to Z.

To make construction of "RDF-savvy" software easier, a number of RDF toolkits have recently appeared, offering functionality that goes beyond mere parsing. Examples of these toolkits are Redland [2], Jena [21], and the ICS-FORTH RDFSuite [1]. These toolkits are typically implemented in either Java or C/C++.

"Wilbur" is Nokia Research Center's open source toolkit for RDF and DAML, written in Common Lisp. Like other RDF toolkits, it offers an API for manipulating RDF data (graphs, nodes, etc.) as well as parsing functionality (parsers not only for XML-encoded RDF and DAML but also for "plain" XML [5] since one written in Common Lisp did not exist when the Wilbur project was started[1]; it also offers a simple HTTP client API for accessing remote URLs for the same reason). Wilbur also offers a frame system API on top of the RDF data API, including a simple query language. Wilbur strives for tight integration of RDF data with the intrinsic features of Common Lisp.

Generally, Wilbur implements the RDF data model by providing four abstract interfaces (and their concrete implementations):

1. The class `node` represents nodes of an RDF graph. Each node may have a URI (Universal Resource Identifier) string associated with it, in which case we consider the node to be *named*; nodes without a URI are called *anonymous* (the reader is referred to the discussion of URIs and their printed representation below).
2. A mapping from URI strings to nodes is provided by the class `dictionary`. The system uses a single default dictionary where all named nodes are placed. The unique mapping from URI strings to `node` instances allows us to implement strict *read/print correspondence* for nodes (described below).
3. The class `triple` represents labeled arcs of an RDF graph. A triple consists of a *subject* (a `node` instance), a *predicate* (also a `node` instance), and an *object* (either a `node` instance or a string, although in the current implementation any Common Lisp object can be used); each triple also has an associated *source* (also a `node` instance), designating the file or HTTP URL from which the triple was originally parsed.
4. Collections of triples are stored in databases (instances of class `db`). The upper level API of the system assumes a single default database, but also exposes a lower-level API where the database can be specified explicitly (allowing software to be constructed which makes use of multiple databases). Simple query functionality is provided for se-

[1] Wilbur's XML parser (written in Common Lisp) has an interface similar to SAX 1 [22]. The parser was written with RDF's needs in mind and does not, for example, support DTDs (except for entity declarations).

lecting triples from a database, similar to the "find" interface of the Stanford RDF API [23] (not to be confused with the Wilbur frame query language described later).

For debugging purposes, the object inspector of the Macintosh Common Lisp was extended to allow easy browsing of RDF graphs.[2]

3. Integration Issues

Our two previous frame systems, BEEF [12, 16] and PORK [17], both addressed the issue of integrating object-oriented programming with frame-based representation. BEEF (which predated practical implementations of the Common Lisp Object System) added object-oriented programming features to a frame system, whereas PORK approached the issue from the opposite direction by taking an object-oriented programming language and adding features of frame-based representation to it; PORK used the Common Lisp *metaobject protocol* [14] to extend the Common Lisp Object System (CLOS).

Wilbur, as a frame system API overlaid on RDF, takes a lower-level approach to integration, by allowing manipulation of RDF graphs. Future development may still address programming issues taking either the "BEEF-approach" (adding programming features to a frame system) or the "PORK-approach" (adding frame features to a programming language). In Wilbur, the RDF/CLOS integration focuses on the following areas:

- ease of use of Common Lisp data structures with RDF,
- issues of input and output of RDF data in a "Common Lisp -friendly" manner, and
- the use of the Common Lisp *condition mechanism* for signaling unexpected situations.

3.1. Reading and Printing RDF Data

To be able to use RDF data seamlessly in an interactive Common Lisp environment, this data must have a printed representation which can be read back into a Common Lisp system. Common Lisp defines this quality, known as *read/print correspondence* [24, p.509], as follows:

> "Ideally, one could print a LISP object and then read the printed representation back in, and so obtain the same identical object. In practice this is difficult and for some purposes not even desirable. Instead, reading a printed representation produces an object that is (with obscure technical exceptions) `equal`[3] to the originally printed object."

The former approach is called "strict read/print correspondence" and the latter "non-strict"; many Common Lisp data structures (such as lists and strings) are non-strict, whereas some (such as symbols) are strict. Wilbur provides *strict* read/print correspondence for nodes.

---

[2] Similar to BBN's DAML Viewer [7]

[3] `equal` is a Common Lisp predicate for structural similarity.

URIs are used internally throughout Wilbur: they give unique identity to nodes. In order to avoid having to write (and read) full URIs, which typically are rather long, the system provides an abbreviated syntax, based on the idea of namespace-qualified names in XML [4]. For example, if we introduce a mapping for the prefix "foo" as follows:

```
"foo" → "http://foo.com/schema#"
```

then we have

```
"foo:bar" → "http://foo.com/schema#bar"
```

Although the XML namespace specification does not specifically define concatenating the expanded form of the prefix with the name part, Wilbur adopts the RDF convention of turning each qualified name into a single (concatenated) URI string.

Wilbur uses the Common Lisp *read macro* mechanism to incorporate the expansion of abbreviated URIs into the *reader* (i.e., the Common Lisp parser). Any expression of the form `!foo:bar` is turned into an instance of Wilbur's `node` class and placed into a dictionary which maps URI strings to `node` instances. This allows references to nodes to be embedded in Common Lisp source files, thus enabling one to embed RDF Data in compiled (binary) files. Wilbur uses the notion of a "forward reference" to a node in cases where the abbreviated URI could not be resolved. When a missing prefix-to-URI mapping is introduced, the system updates the affected nodes by resolving the URIs. This approach is similar to the forward reference model of PORK which allowed one to easily construct circular data structures without having to worry about the order in which named objects were introduced [17].

For printing data structures, Common Lisp defines [24, p.510] that

> "When `print` produces a printed representation, it must choose arbitrarily from among many printed representations. It attempts to choose one that is readable."

The `print-object` method for the Wilbur `node` class uses any existing prefix-to-URI mapping to determine a possible abbreviated form of a node's URI, and subsequently produces a printed representation which can be read in if necessary.

The Wilbur toolkit has two separate parsers, one conforming to the RDF Model and Syntax specification [19] and another conforming to the DAML+OIL reference description [11]. The RDF parser supports all features[4] of the specification, including reification of complete descriptions, reification of individual statements, and the attribute namespace ambiguity. The parser is "near-streaming" and is internally based on a state machine where SAX-like parsing events serve as transition inputs.

[4] Except "rdf:aboutEachPrefix" which probably no-one supports.

The DAML parser (class `daml-parser`) is implemented as an extension of the RDF parser (i.e., as a subclass of `rdf-parser`) and adds support for the DAML collection syntax specified using `rdf:parseType="daml:collection"`.

3.2. Integrating Data Structures

The Wilbur frame API itself is quite simple, basically offering functions for creating frames, for adding values to a slot, for deleting values from a slot, and for reading a slot's values. Frames in Wilbur form graphs when slot values are other frames. Wilbur introduces a query language for selecting subgraphs from these graphs (in other words selecting sets of nodes from RDF graphs). Query expressions are patterns expressed as regular expressions with arc labels (slots, i.e., RDF properties) as atoms, using the following operators and "pseudo-labels":

- **Sequence**: the operator `:seq` matches a sequence of n steps in the graph, consisting of subexpressions e_1,e_2,...,e_n; the operator `:seq+` is similar except any sequence e_1,e_2,...,e_k for k in [1...n] will match.
- **Disjunction**: the operator `:or` matches any one of n subexpressions e_1,e_2,...,e_n.
- **Repetition**: the operator `:rep*` matches the transitive closure of subexpression e; the operator `:rep+` is the same as `(:seq e (:rep* e))`.
- **Inverse**: satisfaction of `(:inv e)` requires the path defined by the subexpression e to be matched in reverse direction.
- **Container membership**: the atom `:members` will match any of the `rdf:_1`, `rdf:_2`, `rdf:_3`, etc. container membership properties.
- **Wildcard**: the atom `:any` will match any label.

The Wilbur query language is similar to the BEEF path grammar [16] which, in turn, was a simplification of the CRL path grammar [8, 9]. Given a "root" node (i.e., a search start point) and a path (a query expression), Wilbur provides functions for retrieving either the first reachable node or all reachable nodes, and for determining whether a path exists between two specified nodes. These functions make it easy to turn RDF graphs into Common Lisp list structures. For example, given a DAML collection (constructed as a "dotted-pair" list using the properties `daml:first` and `daml:rest`), the following query expression will turn it into a Common Lisp list:

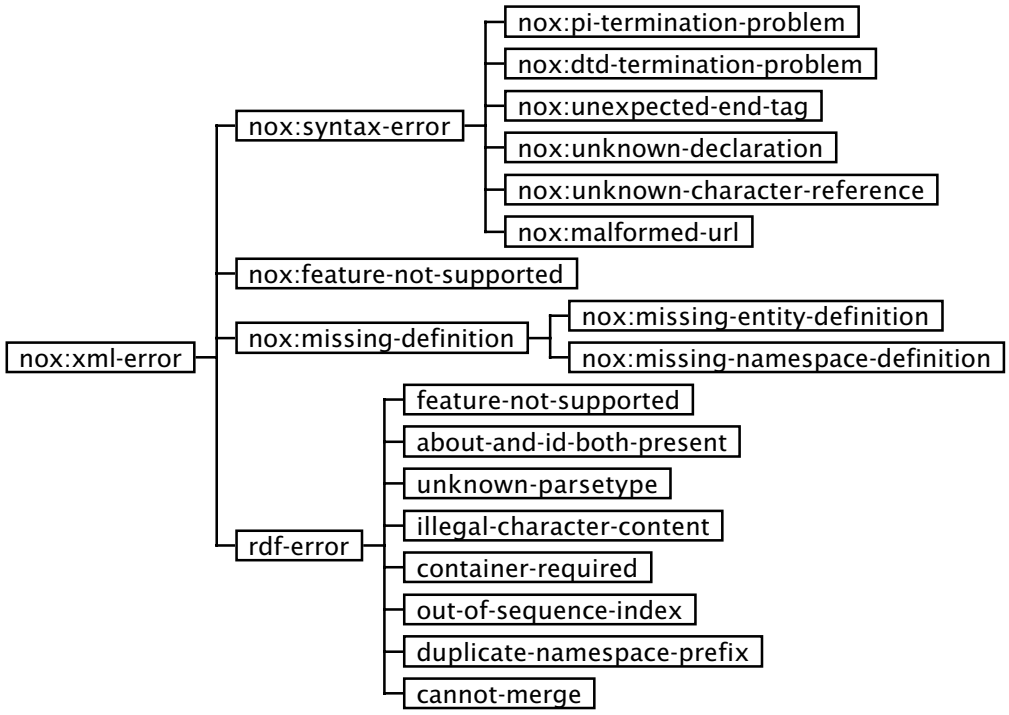
```
(:seq (:rep* !daml:rest) !daml:first)
```

As mentioned before, the Wilbur DAML parser supports the DAML collection syntax and correctly generates dotted-pair lists.

3.3. Dealing with Unexpected Situations

The Common Lisp *condition system* is a powerful mechanism for raising signals when unexpected situations are encountered. When a condition is signaled, instead of reporting an error,

the calling program may choose to catch the signal and allow the execution to continue on from the point where the signal was raised (or caught). Wilbur defines a rich taxonomy of classes for various types of unexpected conditions, and takes full advantage of the condition system's ability to "ignore" errors. The following figure illustrates this taxonomy (note that condition classes in the "nox" package are generated by the XML parser):



As a general rule, all errors of the XML parser are signaled as "non-continuable" (i.e., they abort parsing) whereas all errors of the RDF and DAML parsers are signaled as "continuable" (using the Common Lisp function `cerror`) and allow parsing to continue if the user or the calling program so chooses. The rich taxonomy allows fine-grained mapping of errors to remedial behaviors.

4. Future Work

Several additional features of the toolkit are currently at an experimental stage. These include an RDF *serializer*, capable of producing textual XML from triple databases, and a *schema validator*, capable of checking triple database consistency against the constraints defined by the RDF Schema specification [6].

Both the serializer and the validator make extensive use of the query language. For example, in order to find out whether a slot value (here denoted by *x*) satisfies the (disjunctive) range constraints of a property (here denoted by *p*), the following query can be executed:

```
(relatedp x
          '(:seq !rdf:type
                 (:rep* !rdfs:subClassOf)
                 (:inv !rdfs:range)
                 (:rep* (:inv !rdfs:subPropertyOf)))
          p)
```

Note that the call `(relatedp A B C)` determines whether node C can be reached from node A via path B.

In addition to RDF 1.0 and DAML+OIL, Wilbur will have "plug-in" parsers for the "RDF-like" DMoz Open Directory format and for the alternate RDF syntax "N3".

Other future work will focus on DAML and supporting requirements of the DAML community (for example, we are working on an OKBC interface to the Wilbur frame system), as well as supporting changes introduced by the W3C RDF Core Working Group for the next version of RDF.

5. Conclusions

The Wilbur toolkit attempts to create a programming environment for RDF and DAML by closely integrating some of the representational features with the programming features provided by Common Lisp and CLOS. Issues in integrating input and output of RDF data are addressed, as well as compatibility of RDF and Common Lisp data structures. A query language is introduced to make it easier to select parts of RDF graphs and convert them to Common Lisp data structures.

Exposing RDF as a frame system and allowing programmers to use the full power of Common Lisp makes it easier to create "Semantic Web" applications. Using the frame paradigm also makes it easier to understand RDF (and data models expressed using RDF).

References

[1] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle: "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases", in: S.Staab et al (eds.): "Proceedings of the Second International Workshop on the Semantic Web", May 2001

[2] David Beckett: "The Design and Implementation of the Redland RDF Application Framework", in: Proceedings of the Tenth International World Wide Web Conference, WWW10, May 2001

[3] Tim Berners-Lee, James Hendler, and Ora Lassila: "The Semantic Web", Scientific American, May 2001

[4] Tim Bray, Dave Hollander, and Andrew Layman: "Namespaces in XML", W3C Recommendation, World Wide Web Consortium, January 1999

[5] Tim Bray, Jean Paoli, C.M.Sperberg-McQueen, and Eve Maler: "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, World Wide Web Consortium, October 2000

[6] Dan Brickley & R.V.Guha: "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation, World Wide Web Consortium, March 2000

[7] Mike Dean & Kelly Barber: "DAML Viewer", www.daml.org/viewer/

[8] Mark S. Fox: “Knowledge Representation for Decision Support”, in: L.B.Methlie & R.H.Sprague (eds.): “Knowledge Representation for Decision Support Systems”, Elsevier, 1985

[9] Mark S. Fox, J.Wright, and D.Adam: “Experiences with SRL: An analysis of a frame-based knowledge representation”, in: Expert Database Systems, Benjamin/Cummings, 1985

[10] James Hendler & Deborah L. McGuinness: “DARPA Agent Markup Language”, IEEE Intelligent Systems 15(6):72-73

[11] Frank van Harmelen, Peter F. Patel-Schneider and Ian Horrocks (eds.): "Reference description of the DAML+OIL (March 2001) ontology markup language", working document of the DARPA Agent Markup Language program, March 2001

[12] Juha Hynynen & Ora Lassila: “On the Use of Object-Oriented Paradigm in a Distributed Problem Solver”, AI Communications 2(3):142-151, 1989

[13] Peter D. Karp: “The design space of frame knowledge representation systems”, Technical Report 520, SRI International AI Center, 1992

[14] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow: “The Art of the Metaobject Protocol”, MIT Press, 1991

[15] Ora Lassila: “Frames or Objects, or Both?”, Workshop Notes from the Eight National Conference on Artificial Intelligence (AAAI-90): Object-Oriented Programming in AI, American Association for Artificial Intelligence, July 1990 (also Report HTKK-TKO-B67, Department of Computer Science, Helsinki University of Technology, 1990)

[16] Ora Lassila: “BEEF Reference Manual - A Programmer's Guide to the BEEF Frame System”, Second Version, Report HTKK-TKO-C46, Department of Computer Science, Helsinki University of Technology, 1991

[17] Ora Lassila: "PORK Object System Programmer's Guide", Report CMU-RI-TR-95-12, The Robotics Institute, Carnegie Mellon University, 1995

[18] Ora Lassila: “Web Metadata: A Matter of Semantics”, IEEE Internet Computing 2(4):30-37

[19] Ora Lassila & Ralph R. Sw ick: "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, World Wide Web Consortium, February 1999

[20] Ora Lassila & Deborah L. McGuinness: "The Role of Frame-Based Representation on the Semantic Web", Report KSL-01-02, Knowledge Systems Laboratory, Stanford University, 2001

[21] Brian McBride: “Jena: Implementing the RDF Model and Syntax Specification”, in: Steffen Staab et al (eds.): “Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001”, May 2001

[22] David Megginson: “SAX 1.0: The Simple API for XML”, www.megginson.com/SAX/SAX1/

[23] Sergey Melnik: "RDF API Draft", working document, Stanford University, 1999

[24] Guy L. Steele, Jr: “Common Lisp - the Language, 2nd ed.”, Digital Press, 1990

Acknowledgements

The author would like to thank the following individuals for their advice during the Wilbur project and during the preparation of this article: Jessica Jenkins, Marcia Lassila and Louis Theran, as well as the three anonymous reviewers whose suggestions proved invaluable.

Although portable to any Common Lisp platform, the Wilbur toolkit was developed entirely using Digitool’s “Macintosh Common Lisp” (which the author considers to be a fantastic software development environment).

Wilbur is an open source software project. More information about the project is available at http://purl.org/NET/wilbur/.

DAML-S: Semantic Markup For Web Services

The DAML Services Coalition:
Anupriya Ankolekar[1], Mark Burstein[2], Jerry R. Hobbs[3], Ora Lassila[4],
David L. Martin[3], Sheila A. McIlraith[5], Srini Narayanan[3], Massimo Paolucci[1],
Terry Payne[1], Katia Sycara[1], Honglei Zeng[5,6]

Abstract.
The Semantic Web should enable greater access not only to content but also to services on the Web. Users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties. As part of the DARPA Agent Markup Language program, we have begun to develop an ontology of services, called DAML-S, that will make these functionalities possible. In this paper we describe the overall structure of the ontology, the service profile for advertising services, and the process model for the detailed description of the operation of services. We also compare DAML-S with several industry efforts to define standards for characterizing services on the Web.

1 Introduction: Services on the Semantic Web

Efforts toward the creation of the Semantic Web are gaining momentum [2]. Soon it will be possible to access Web resources by content rather than just by keywords. A significant force in this movement is the development of DAML—the DARPA Agent Markup Language [10]. DAML enables the creation of ontologies for any domain and the instantiation of these ontologies in the description of specific Web sites.

Among the most important Web resources are those that provide services. By "service" we mean Web sites that do not merely provide static information but allow one to effect some action or change in the world, such as the sale of a product or the control of a physical device. The Semantic Web should enable users to locate, select, employ, compose, and monitor Web-based services automatically.

To make use of a Web service, a software agent needs a computer-interpretable description of the service, and the means by which it is accessed. An important goal for DAML, then, is to establish a framework within which these descriptions are made and shared. Web sites should be able to employ a set of basic classes and properties for declaring and describing services, and the ontology structuring mechanisms of DAML provide the appropriate framework within which to do this.

This paper describes a collaborative effort by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, and SRI International to define just such an ontology. We call this language DAML-S. We first motivate our effort with some sample tasks. In the central part of the paper we describe the upper ontology for services that we have developed, including the ontologies for profiles, processes, and time, and thoughts toward a future ontology of process control. We then compare DAML-S with a number of recent industrial efforts to standardize a markup language for services.

[1] The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA
[2] BBN Technologies, Cambridge, Massachusetts
[3] Artificial Intelligence Center, SRI International, Menlo Park, California
[4] Nokia Research Center, Burlington, Massachusetts
[5] Knowledge Systems Laboratory, Stanford University, Stanford, California
[6] Authors' names are in alphabetical order.

2 Some Motivating Tasks

Services can be simple or primitive in the sense that they invoke only a single Web-accessible computer program, sensor, or device that does not rely upon another Web service, and there is no ongoing interaction between the user and the service, beyond a simple response. For example, a service that returns a postal code or the longitude and latitude when given an address would be in this category. Alternately, services can be complex, composed of multiple primitive services, often requiring an interaction or conversation between the user and the services, so that the user can make choices and provide information conditionally. One's interaction with www.amazon.com to buy a book is like this; the user searches for books by various criteria, perhaps reads reviews, may or may not decide to buy, and gives credit card and mailing information. DAML-S is meant to support both categories of services, but complex services have provided the primary motivations for the features of the language. The following four sample tasks will give the reader an idea of the kinds of tasks we expect DAML-S to enable [13, 14].

1. **Automatic Web service discovery.** Automatic Web service discovery involves the automatic location of Web services that provide a particular service and that adhere to requested constraints. For example, the user may want to find a service that sells airline tickets between two given cities and accepts a particular credit card. Currently, this task must be performed by a human who might use a search engine to find a service, read the Web page, and execute the service manually, to determine if it satisfies the constraints. With DAML-S markup of services, the information necessary for Web service discovery could be specified as computer-interpretable semantic markup at the service Web sites, and a service registry or ontology-enhanced search engine could be used to locate the services automatically. Alternatively, a server could proactively advertise itself in DAML-S with a service registry, also called middle agent [4, 24, 12], so that requesters can find it when they query the registry. Thus, DAML-S must provide declarative advertisements of service properties and capabilities that can be used for automatic service discovery.

2. **Automatic Web service invocation.** Automatic Web service invocation involves the automatic execution of an identified Web service by a computer program or agent. For example, the user could request the purchase of an airline ticket from a particular site on a particular flight. Currently, a user must go to the Web site offering that service, fill out a form, and click on a button to execute the service. Alternately the user might send an HTTP request directly to the service with the appropriate parameters in HTML. In either case, a human in the loop is necessary. Execution of a Web service can be thought of as a collection of function calls. DAML-S markup of Web services provides a declarative, computer-interpretable API for executing these function calls. A software agent should be able to interpret the markup to understand what input is necessary to the service call, what information will be returned, and how to execute the service automatically. Thus, DAML-S should provide declarative APIs for Web services that are necessary for automated Web service execution.

3. **Automatic Web service composition and interoperation.** This task involves the automatic selection, composition and interoperation of Web services to perform some task, given a high-level description of an objective. For example, the user may want to make all the travel arrangements for a trip to a conference. Currently, the user must select the Web services, specify the composition manually, and make sure that any software needed for the interoperation is custom-created. With DAML-S markup of Web services, the information necessary to select and compose services will be encoded at the service Web sites. Software can be written to manipulate these representations, together with a specification of the objectives of the task, to achieve the task automatically. Thus, DAML-S must provide declarative specifications of the prerequisites and consequences of individual service use that are necessary for automatic service composition and interoperation.

4. **Automatic Web service execution monitoring.** Individual services and, even more, compositions of services, will often require some time to execute completely. Users may want to know during this period what the status of their request is, or their plans may have changed requiring alterations in the actions the software agent takes. For example, users may want to make sure their hotel reservation has already been made. For these purposes, it would be good to have the ability to find out where in the process the request is and whether any unanticipated glitches have appeared. Thus, DAML-S should provide descriptors for the execution of services. This part of DAML-S is a goal of ours, but it has not yet been defined.

Any Web-accessible program/sensor/device that is *declared* as a service will be regarded as a service. DAML-S does not preclude declaring simple, static Web pages to be services. But our primary motivation in defining DAML-S has been to support more complex tasks like those described above.

3 An Upper Ontology for Services

The class *Service* stands at the top of a taxonomy of services, and its properties are the properties normally associated with all kinds of services. The upper ontology for services is silent as to what the particular subclasses of *Service* should be, or even the conceptual basis for structuring this taxonomy, but it is expected that the taxonomy will be structured according to functional and domain differences and market needs. For example, one might imagine a broad subclass, *B2C-transaction*, which would encompass services for purchasing items from retail Web sites, tracking purchase status, establishing and maintaining accounts with the sites, and so on.

Our structuring of the ontology of services is motivated by the need to provide three essential types of knowledge about a service (shown in figure 1), each characterized by the question it answers:

- *What does the service require of the user(s), or other agents, and provide for them?* The answer to this question is given in the "profile[7]." Thus, the class *Service presents* a *ServiceProfile*
- *How does it work?* The answer to this question is given in the "model." Thus, the class *Service* is *describedBy* a *ServiceModel*
- *How is it used?* The answer to this question is given in the "grounding." Thus, the class *Service supports* a *ServiceGrounding*

The properties *presents*, *describedBy*, and *supports* are properties of *Service*. The classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding* are the respective ranges of those properties. We expect that each descendant class of *Service*, such as *B2C-transaction*, will *present* a descendant class of *ServiceProfile*, be *describedBy* a descendant class of *ServiceModel*, and *support* a descendant class of *ServiceGrounding*. The details of profiles, models, and groundings may vary widely from one type of service to another—that is, from one descendant class of *Service* to another. But each of these three classes provides an essential type of information about the service, as characterized in the rest of the paper.

The service profile tells "what the service does"; that is, it gives the type of information needed by a service-seeking agent to determine whether the service meets its needs (typically such things as input and output types, preconditions and postconditions, and binding patterns). In future versions, we will use logical rules or their equivalent in such a specification for expressing interactions among parameters. For instance, a rule might say that if a particular input argument is bound in a certain way, certain other input arguments may not be needed, or may be provided by the service itself. As DAML and DAML-S

[7] A service profile has also been called service capability advertisement [20].

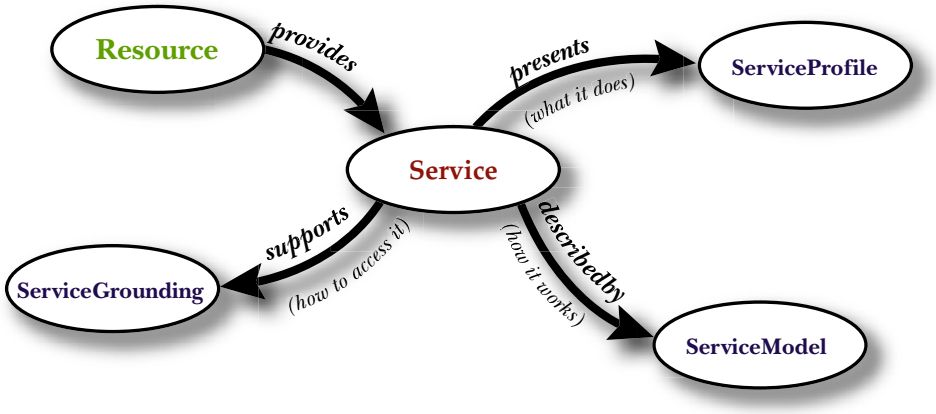


Figure 1: Top level of the service ontology

and their applications evolve, logical rules and inferential approaches enabled by them are likely to play an increasingly important role in models and groundings, as well as in profiles. See [5] for additional examples.

The service model tells "how the service works"; that is, it describes what happens when the service is carried out. For non-trivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; (4) to monitor the execution of the service. For non-trivial services, the first two tasks require a model of action and process, the last two involve, in addition, an execution model.

A service grounding ("grounding" for short) specifies the details of how an agent can access a service. Typically a grounding will specify a communications protocol (e.g., RPC, HTTP-FORM, CORBA IDL, SOAP, Java RMI, OAA ACL [12]), and service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the *ServiceModel*, an unambiguous way of exchanging data elements of that type with the service (that is, the marshaling/serialization techniques employed). The likelihood is that a relatively small set of groundings will come to be widely used in conjunction with DAML services. Groundings will be specified at various well-known URIs.

Generally speaking, the *ServiceProfile* provides the information needed for an agent to discover a service. Taken together, the *ServiceModel* and *ServiceGrounding* objects associated with a service provide enough information for an agent to make use of a service.

The upper ontology for services deliberately does not specify any cardinalities for the properties *presents*, *describedBy*, and *supports*. Although, in principle, a service needs all three properties to be fully characterized, it is possible to imagine situations in which a partial characterization could be useful. Hence, there is no specification of a minimum cardinality. Further, it should certainly be possible for a service to offer multiple profiles, multiple models, and/or multiple groundings. Hence, there is no specification of a maximum cardinality.

In general, there need not exist a one-to-one correspondence between profiles, models, and/or groundings. The only constraint among these three characterizations that might appropriately be expressed at the upper level ontology is that for each model, there must be at least one supporting grounding.

In the following two sections we discuss the service profile and the service model in greater detail

(Service groundings are not discussed further, but will be covered in greater depth in a subsequent publication.)

4 Service Profiles

A service profile provides a high-level description of a service and its provider [21, 20]; it is used to request or advertise services with discovery/location registries. Service profiles consist of three types of information: a human readable *description* of the service; a specification of the *functionalities* that are provided by the service; and a host of *functional attributes* which provide additional information and requirements about the service that assist when reasoning about several services with similar capabilities. Service functionalities are represented as a transformation from the inputs required by the service to the outputs produced. For example, a news reporting service would advertise itself as a service that, given a date, will return the news reported on that date. Functional attributes specify additional information about the service, such as what guarantees of response time or accuracy it provides, or the cost of the service.

While service providers use the service profile to advertise their services, service requesters use the profile to specify what services they need and what they expect from such a service. For instance, a requester may look for a news service that reports stock quotes with no delay with respect to the market. The role of the registries is to match the request against the profiles advertised by other services and identify which services provide the best match.

Implicitly, the service profiles specify the intended purpose of the service, because they specify only those functionalities that are publicly provided. A book-selling service may involve two different functionalities: it allows other services to browse its site to find books of interest, and it allows them to buy the books they found. The book-seller has the choice of advertising just the book-buying service or both the browsing functionality and the buying functionality. In the latter case the service makes public that it can provide browsing services, and it allows everybody to browse its registry without buying a book. In contrast, by advertising only the book-selling functionality, but not the browsing, the agent discourages browsing by requesters that do not intend to buy. The decision as to which functionalities to advertise determines how the service will be used: a requester that intends to browse but not to buy would select a service that advertises both buying and browsing capabilities, but not one that advertises buying only.

The service profile contains only the information that allows registries to decide which advertisements are matched by a request. To this extent, the information in the profile is a summary of the information in the process model and service grounding. Where, as in the above example, the service does not advertise some of its functionalities, they will not be part of the service profile. But they *are* part of the service model to the extent that they are needed for *achieving* the advertised services. For example, looking for a book is an essential prerequisite for buying it, so it would be specified in the process model, but not necessarily in the profile. Similarly, information about shipping may appear within the process model but not the profile.

4.1 Description

Information about the service, such as its provenance or a text summary, is provided within the profile. This is primarily for use by human users, although these properties are considered when locating requested services.

4.2 Functionality Description

An essential component of the profile is the specification of what the service provides and the specification of the conditions that have to be satisfied for a successful result. In addition, the profile specifies what

conditions result from the service including the expected and unexpected results of the service activity.

The service is represented by `input` and `output` properties of the profile. The input property specifies the information that the service requires to proceed with the computation. For example, a book-selling service could require the credit-card number and bibliographical information of the book to sell. The outputs specify the result of the operation of the service. For the book-selling agent the output could be a receipt that acknowledges the sale.

```
<rdf:Property rdf:ID="input">
  <rdfs:comment>
    Property describing the inputs of a service in the Service Profile
  </rdfs:comment>
  <rdfs:domain rdf:resource="#ServiceProfile"/>
  <rdfs:subPropertyOf rdf:resource="#parameter"/>
</rdf:Property>
```

While inputs and outputs represent the service, they are not the only things affected by the operations of the service. For example, to complete the sale the book-selling service requires that the credit card is valid and not overdrawn or expired. In addition, the result of the sale is not only that the buyer owns the book (as specified by the outputs), but that the book is physically transferred from the the warehouse of the seller to the house of the buyer. These conditions are specified by `precondition` and `effect` properties of the profile. Preconditions present one or more logical conditions that should be satisfied prior to the service being requested. These conditions should have associated explicit effects that may occur as a result of the service being performed. Effects are events that are caused by the successful execution of a service.

```
<rdf:Property rdf:ID="precondition">
  <rdfs:domain rdf:resource="#ServiceProfile"/>
  <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>
```

The service profile also provides a specific type of precondition called an `accessCondition`, that is expected to be true for the service to succeed, but is not modified by the activity of the service. Access conditions are used when the access to the service is restricted to only some users: as, for example, services that are restricted to users affiliated to some organization. For instance, to access a classified news service a user needs to have some level of clearance, details about it would be specified as an `accessCondition`.

Finally, the profile allows the specification of what `domainResources` are affected by the use of the service. These domain resources may include computational resources such as bandwidth or disk space as well as more material resources consumed when the service controls some machinery. This type of resource may include fuel, or materials modified by the machine.

4.3 Functional Attributes

In the previous section we introduced the functional description of services. Yet there are other aspects of services that the users should be aware of. While a service may be accessed from anywhere on the Internet, it may only be applicable to a specific audience. For instance, although it is possible to order food for delivery from a Pittsburgh-based restaurant Web site in general, one cannot reasonably expect to do this from California. Functional attributes address the problem that there are properties that can be used to describe a service other than as a functional process. These properties are described below.

geographicRadius The geographic radius refers to the geographic scope of the service. This may be at the global or national scale (e.g. for e-commerce) or at a local scale (e.g. pizza delivery).

degreeOfQuality This property provides qualifications for the service. For example, the following two sub-properties are examples of different degrees of quality, and could be defined within some additional ontology.

serviceParameter An expandable list of properties that may accompany a profile description.

communicationThru This property provides a high-level summary of how a service may communicate, such as what agent communication language (ACL) is used (e.g., FIPA, KQML, SOAP). This summarizes the descriptions provided by the service grounding and are used when matching services; but is not intended to replace the detail provided by the service grounding.

serviceType The service type refers to a high-level classification of the service, for example B2B, B2C etc.

serviceCategory The service category refers to an ontology of services that may be on offer. High-level services could include Products as well as Problem-Solving Capabilities, Commercial Services, Information and so on.

qualityGuarantees These are guarantees that the service promises to deliver, such as guaranteeing to provide the lowest possible interest rate, or a response within 3 minutes, etc.

qualityRating The quality rating property represents an expandable list of rating properties that may accompany a service profile. These ratings refer to industry accepted ratings, such as the Dun and Bradstreet Rating for businesses, or the Star rating for Hotels. For example:

```
<!-- Dun and Bradstreet Rating -->
<rdf:Property rdf:ID="dAndBRating">
  <rdfs:subPropertyOf rdf:resource="#qualityRating" />
</rdf:Property>
```

As a result of the service profile, the user, be it a human, a program or another service, would be able to identify what the service provides, what conditions result from the service and whether the service is available, accessible and how it compares with other functionally equivalent services.

5 Modeling Services as Processes

A more detailed perspective on services is that a service can be viewed as a *process*. We have defined a particular subclass of *ServiceModel*, the *ProcessModel* (as shown in figure 2), which draws upon well-established work in a variety of fields, such as AI planning and workflow automation, and which we believe will support the representational needs of a very broad array of services on the Web.

The two chief components of a process model are the *process model*, which describes a service in terms of its component actions or processes, and enables planning, composition and agent/service interoperation; and the *process control model*, which allows agents to monitor the execution of a service request. We will refer to the first part as the Process Ontology and the second as the Process Control Ontology. Only the former has been defined in the current version of DAML-S, but below we briefly describe our intentions with regard to the latter. We have defined a simple ontology of time, described below; in subsequent versions this will be elaborated. We also expect in a future version to provide an ontology of resources.

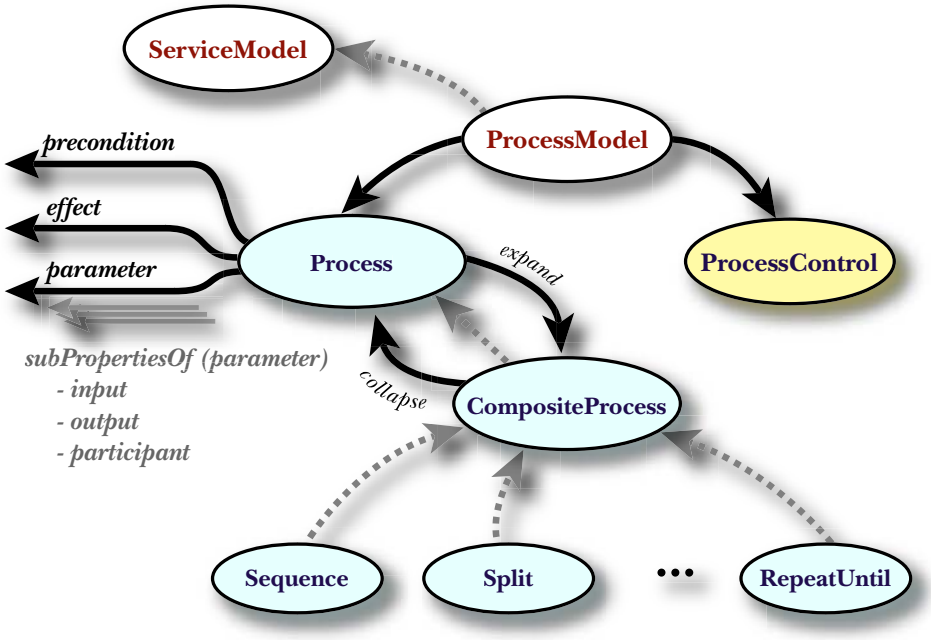


Figure 2: Top level of process modeling ontology

5.1 The Process Ontology

We expect our process ontology to serve as the basis for specifying a wide array of services. In developing the ontology, we drew from a variety of sources, including work in AI on standardizations of planning languages [9], work in programming languages and distributed systems [16, 15], emerging standards in process modeling and workflow technology such as the NIST's Process Specification Language (PSL) [19] and the Workflow Management Coalition effort (http://www.aiim.org/wfmc), work on modeling verb semantics and event structure [17], previous work on action-inspired Web service markup [14], work in AI on modeling complex actions [11], and work in agent communication languages [12, 8].

The primary kind of entity in the Process Ontology is, unsurprisingly, a "process".[8] A process can have any number of inputs, representing the information that is, under some conditions, required for the execution of the process. It can have any number of outputs, the information that the process provides, conditionally, after its execution. Participants and other parameters may be specified; for example, the participants may include the roles in the event frame, such as the agents, patient, and instrument, whereas other parameters, especially for physical devices, might be rates, forces, and knob-settings. There can be any number of preconditions, which must all hold in order for the process to be invoked. Finally, the process can have any number of effects.

A process can often be viewed either as a primitive, undecomposable process or as a composite process, decomposable into other primitive or composite processes. Either perspective may be the more useful in some given context. Thus, a top-level PROCESS class has, as its sole subclass, COMPOSITEPROCESS, which in turn is subclassed by a variety of control structures.

[8] This term was chosen over the terms "event" and "action", in part because it is more suggestive of internal structure than "event" and because it does not necessarily presume an agent executing the process and thus is more general than "action". Ultimately, however, the choice is arbitrary. It is modeled after computational procedures or planning operators.

More precisely, in DAML-S:

- **Process**

```
<rdfs:Class rdf:ID="Process">
  <rdfs:comment> Top-level class for describing how a service works
  </rdfs:comment>
</rdfs:Class>
```

Class PROCESS has related properties *parameter*, *input*, *output*, *participant*, *precondition*, and (conditional) *effect*. Each of these properties ranges over a DAML object, which, at the upper ontology level, is not restricted at all. The properties *input*, *output*, and *participant* are categorized as subproperties of *parameter*. Subclasses of PROCESS for specific domains can use DAML language elements to indicate more specific range restrictions, as well as cardinality restrictions for each of these properties.

The following is an example of a property definition:

```
<rdf:Property rdf:ID="parameter">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource=""http://www.daml.org/2001/03/daml+oil#Thing"/>
</rdf:Property>
```

In addition to its action-related properties, a PROCESS has a number of bookkeeping properties such as *name(rdf:literal), address (URI), documentsread (URI), documentsupdated (URI),* and so on.

- **CompositeProcess**

```
<daml:Class rdf:ID="CompositeProcess">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Process"/>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#components"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>

<rdf:Property rdf:ID="components">
  <rdfs:comment>
    Holds the specific arrangement of subprocesses.
  </rdfs:comment>
  <rdfs:domain rdf:resource="#CompositeProcess"/>
</rdf:Property>
```

Composite processes are processes that have additional properties called *components* to indicate the ordering and conditional execution of the subprocesses from which they are composed. For instance, the composite process, SEQUENCE, has a *components* property that ranges over a PROCESSLIST (a list whose items are restricted to be simple or composite processes). In the process "upper ontology", we have attempted to come up with a minimal set of process classes that can be specialized to describe a variety of Web services. This minimal set consists of Sequence, Split, Split + Join, Choice, Unordered, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until.

Note that while a composite process is a process, and thus has slots for preconditions and effects, there may be no easy way to compute these values for an arbitrary composite process, given its component sub-processes.

There are two fundamental relations between processes and composite processes. The EXPAND relation associates a Process with the CompositeProcess describing its component subprocesses, while its inverse, the COLLAPSE relation represents the association of the CompositeProcess to its atomic Process form. Expanding is intended to provide a "glassbox" and collapsing a "blackbox" view of the process. The expanded version is likely to be used for service composition (both off-line and runtime) and the collapsed version for service execution.

The minimal set of composition templates (subclasses of *CompositeProcess*) is as follows:

Sequence : A list of Processes to be done in order. We use a DAML restriction to restrict the components of a Sequence process to be a List of subprocesses (simple and/or composite).

```
<rdfs:Class rdf:ID="Sequence">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <rdfs:Class> rdf:about="#Process" </rdfs:Class>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#components"/>
      <daml:toClass rdf:resource="#ProcessList"/>
    </daml:Restriction>
  <daml:intersectionOf>
</rdfs:Class>
```

Split : The components of a *Split* process are a bag of sub-processes to be executed concurrently. No further specification about waiting or synchronization is made at this level.

```
<rdfs:Class rdf:ID="Split">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <rdfs:Class> rdf:about ="#Process" </rdfs:Class>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#components"/>
      <daml:toClass rdf:resource="#ProcessBag"/>
    </daml:Restriction>
  <daml:intersectionOf>
</rdfs:Class>
```

Split is similar to other ontologies' use of Fork, Concurrent, or Parallel. We use the DAML *sameClassAs* feature to accommodate the different standards for specifying this.

Unordered : Here a bag of processes can be executed in any order. No further constraints are specified. All processes must be executed.

Split+Join : Here the process consists of concurrent execution of a bunch of sub-processes with barrier synchronization. With Split and Split and Join, we can define processes that have partial synchronization (e.g., split all and join some sub-bag).

Choice : Choice is a composite process with additional properties "chosen" and "chooseFrom". These properties can be used both for process and execution control (e.g., choose from "chooseFrom" and do "chosen" in sequence, or choose from "chooseFrom" and do "chosen" in parallel) as well for constructing new subclasses like "choose at least n from m", "choose exactly n from m", "choose at most n from m" [9], and so on.

Condition : Conditions are composite processes with an output property (conditionValue) whose range is a binary value. Conditions usually correspond to test actions, but they may be world states, resource levels, timeouts or other things affecting the evolution of processes.

[9] This can be obtained by restricting the size of the Process Bag that corresponds to the "components" of the chosen and chooseFrom subprocesses using cardinality, min-cardinality, max-cardinality to get choose(n, m)($0 \leq n \leq |components(chooseFrom)|, 0 < m \leq |components(chosen)|$).

If-Then-Else : The If-Then-Else class is a composite process that has properties "ifCondition", "then" and "else" holding different aspects of the If-Then-Else composite process. Its semantics is intended as "Test If-condition; if True do *Then*, if False do *Else*."

```
<rdf:Property rdf:ID="ifCondition">
  <rdfs:comment> The if condition of an if-then-else </rdfs:comment>
  <rdfs:domain rdf:resource="#If-Then-Else"/>
  <rdfs:range> rdf:resource ="#Condition" </rdfs:range>
</rdf:Property>

<rdf:Property rdf:ID="then">
  <rdfs:domain rdf:resource="#If-Then-Else"/>
  <rdfs:range rdf:resource="#CompositeProcess"/>
</rdf:Property>

<rdf:Property rdf:ID="else">
  <rdfs:domain rdf:resource="#If-Then-Else"/>
  <rdfs:range rdf:resource="#CompositeProcess"/>
</rdf:Property>
```

Iterate : Iterate is a composite process whose next process property has the same value as the current process. Repeat is defined as a synonym of the iterate class. The repeat/iterate process makes no assumption about how many iterations are made or when to initiate, terminate or resume. The initiation, termination or maintainance condition could be specified with a whileCondition or an untilCondition as below.[10]

Repeat-Until : The Repeat-Until class is similar to the Repeat-While class in that specializes the If-Then-Else class where the "ifCondition" is the same as the untilCondition and different from the Repeat-While class in that the "else" (compared to "then") property is the repeated process. Thus the process repeats till the untilCondition becomes true.

5.2 Process Control Ontology

A process instantiation represents a complex process that is executing in the world. To monitor and control the execution of a process, an agent needs a model to interpret process instantiations with three characteristics:

1. It should provide the mapping rules for the various input state properties (inputs, preconditions) to the corresponding output state properties.

2. It should provide a model of the temporal or state dependencies described by the sequence, split, split+join, etc constructs.

3. It should provide representations for messages about the execution state of atomic and composite processes sufficient to do execution monitoring. This allows an agent to keep track of the status of executions, including successful, failed and interrupted processes, and to respond to each appropriately.

We have not defined a process control ontology in the current version of DAML-S, but we plan to in a future version.

[10] Another possible extension is to ability to define counters and use their values as termination conditions. This could be part of an extended process control and execution monitoring ontology.

5.3 Time

For the initial version of DAML-S we have defined a very simple upper ontology for time. There are two classes of entities—*instants* and *intervals.* Each is a subclass of *temporal-entity.*

There are three relations that may obtain between an instant and an interval, defined as DAML-S properties:

1. The *Start-of* property whose domain is the Interval class and whose range is an Instant.

2. The *End-of* property whose domain is the Interval class and whose range is an Instant.

3. The *Inside* property whose domain is the Interval class and whose range is an Instant.

No assumption is made that intervals *consist of* instants.

There are two possible relations that may obtain between a process and one of the temporal objects. A process may be in an *at-time* relation to an instant or in a *during* relation to an interval. Whether a particular process is viewed as instantaneous or as occuring over an interval is a granularity decision that may vary according to the context of use. These relations are defined in DAML-S as properties of processes.

1. The *At-time* property: its domain is the Process class and its range is an Instant.

2. The *During* property: its domain is the Process class and its range is an Interval.

Viewed as intervals, processes could have properties such as startTime and endTime which are synonymous (daml:samePropertyAs) with the Start-Of and End-Of relation that obtains between intervals and instants.

One further relation can hold between two temporal entities—the *before* relation. The intended semantics is that for an instant or interval to be before another instant or interval, there can be no overlap or abutment between the former and the latter. In DAML-S the *Before* property whose domain is the Temporal-entity class and whose range is a Temporal-entity.

Different communities have different ways of representing the times and durations of states and events (processes). For example, states and events can both have durations, and at least events can be instantaneous; or events can only be instantaneous and only states can have durations. Events that one might consider as having duration (e.g., heating water) are modeled as a state of the system that is initiated and terminated by instantaneous events. That is, there is the instantaneous event of the start of the heating at the start of an interval, that transitions the system into a state in which the water is heating. The state continues until another instantaneous event occurs—the stopping of the event at the end of the interval. These two perspectives on events are straightforwardly interdefinable in terms of the ontology we have provided. Thus, DAML-S supports both.

The various relations between intervals defined in Allen's temporal interval calculus [1] can be defined in a straightforward fashion in terms of *before* and identity on the start and end points. For example, two intervals meet when the end of one is identical to the start of the other. Thus, in the near future, when DAML is augmented with the capability of defining logical rules, it will be easy to incorporate the interval calculus into DAML-S. In addition, in future versions of DAML-S we will define primitives for measuring durations and for specifying clock and calendar time.

6 Example Walk-Through

To illustrate the concepts described in this paper, we have developed an example of a fictitious book-buying service offered by the Web service provider, Congo Inc. Congo has a suite of programs that they are making accessible on the Web. Congo wishes to compose these individual programs into Web services that it offers to its users. We focus here on the Web service of buying a book, CongoBuy. In the DAML-S release, we present a walk-through that steps through the process of creating DAML-S markup for Congo[11].

We take the perspective of the typical Web service provider and consider three automation tasks that a Web service provider might wish to enable with DAML-S: 1) automatic Web service discovery, 2) automatic Web service invocation, and 3) automatic Web service composition and interoperation. For the purposes of this paper, we limit our discussion to the second and third tasks.

6.1 Web Service Invocation

To automate Web Service Invocation, DAML-S markup must tell a program how to automatically construct an (http) call to execute or invoke a Web service, and what output(s) may be returned from the service. To enable such functionality, the process ontology in DAML-S provides markup to describe individual and composite Web-accessible programs as either simple or composite processes.

6.1.1 Define the Service as a Process

Congo Inc. provides the CongoBuy Web service to its customers. We view the CongoBuy Web service as a Process, i.e., it is a subclass of the class Process in the process ontology.

```
<rdfs:Class rdf:ID="CongoBuy">
  <rdfs:subClassOf rdf:resource=
    "http://www.daml.org/services/daml-s/2001/05/Process.daml#Process"/>
</rdfs:Class>
```

Although the CongoBuy service is actually a predetermined composition of several of Congo's Web-accessible programs, it is useful to initially view it as a black-box process. The black-box process, CongoBuy has a variety of invocation-relevant properties, including input, (conditional) output and parameter. For example, input to the CongoBuy book-buying service includes the name of the book (bookName), the customer's credit card number, and their account number and password. If the service being described is simple in that it is not the composition of other services or programs, then the service inputs are simply the set of inputs that must be provided in the service invocation. The outputs are the outputs returned from the service invocation. Note that these outputs may be conditional. For example the output of a book-buying service will vary depending upon whether the book is in or out of stock.

In contrast, if the service is composed of other services, as is the case with CongoBuy, then the rationale for specification of the inputs, outputs and parameters is more difficult, and the utility of these properties is limited. In the simplest case, the inputs and outputs of the black-box process can be defined to be the composition of all the possible inputs and all the possible (conditional) outputs of the simple services that the black-box process may invoke, taking every possible path through the composition of simple services. Note however that this is not a very exacting specification. In particular, the collection of outputs may be contradictory (e.g., one path of CongoBuy may lead to confirmation of a purchase, while another may lead to confirmation of no purchase). The conditions under which inputs and outputs arise are encoded exactly in the expand of this black-box process, and can be retrieved from the expanded process. The inputs, outputs and parameters for the black-box process are designed to be a useful shorthand. Thus, it

[11] The Congo example can be found at http://www.daml.org/services/daml-s/2001/05/Congo.daml.

could be argued that the inputs and outputs should describe the most likely inputs and outputs through the system. However, in some cases, even this is difficult to define. For now, DAML-S leaves this decision up to the Web service provider.

The following is an example of one input to CongoBuy. Note that it is a subproperty of the property input of Process, from the process model.

```
<rdf:Property rdf:ID="bookName">
  <rdfs:subPropertyOf rdf:resource=
    "http://www.daml.org/services/daml-s/2001/05/Process.daml#input"/>
  <rdfs:domain rdf:resource="#CongoBuy"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>
```

An output can similarly be defined as a subproperty of the property output of Process. In a real book-buying service, this output would likely be conditioned on the book being in stock, or the customer's credit card being valid, but to simplify our example, we assume Congo has an infinite supply of books, and infinite generosity.

```
<rdf:Property rdf:ID="eReceiptOutput">
  <rdfs:subPropertyOf rdf:resource=
    "http://www.daml.org/services/daml-s/2001/05/Process.daml#output"/>
  <rdfs:range rdf:resource="#EReceipt"/>
</rdf:Property>
```

In addition to input and output properties, each service has parameter properties. A parameter is something that affects the outcome of the process, but which is not an input provided by the invoker of the process. It may be known by the service, or retrieved by the service from elsewhere. For example, the fact that the customer's credit card is valid, is a parameter in our CongoBuy process, and is relevant when considering the use of the CongoBuy, but it is not an input or output of CongoBuy.

```
<rdf:Property rdf:ID="creditCardValidity">
  <rdfs:subPropertyOf rdf:resource=
    "http://www.daml.org/services/daml-s/2001/05/Process.daml#parameter"/>
  <rdfs:range rdf:resource="#ValidityType"/>
</rdf:Property>
```

6.1.2 Define the Process as a Composition of Processes

Given the variability in the specification of inputs, outputs and parameters, it is generally insufficient to simply specify a service as a black-box process, if the objective is to automate service invocation. We must expand the black-box service to describe its composite processes. This is achieved by first defining the individual processes and then defining their composition as a composite process.

Define the Individual Processes

We first define each of the simple services in CongoBuy, i.e., LocateBook, PutInCart, etc.[12]

```
<rdfs:Class rdf:ID="LocateBook">
  <rdfs:subClassOf rdf:resource="#CongoBuy"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="PutInCart">
  <rdfs:subClassOf rdf:resource="#CongoBuy"/>
</rdfs:Class>

<rdf:Property rdf:ID="bookSelected">
  <rdfs:subPropertyOf rdf:resource=
    "http://www.daml.org/services/daml-s/2001/05/Process.daml#input"/>
  <rdfs:domain rdf:resource="#PutInCart"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>
```

Define the Composition of the Individual Processes

The composition of each of our simple services can be defined by using the composition constructs created in the process ontology, i.e., Sequence, Split, Split + Join, Unordered, Condition, If-Then-Else, Repeat-While, Repeat-Until. We first create an expand class and then construct the overall expand class recursively in a top- down manner.

```
<process:expand>
  <rdfs:Class> rdfs:about ="#CongoBuy"</rdfs:Class>
  <rdfs:Class> rdfs:about ="#ExpandedCongoBuy"</rdfs:Class>
</process:expand>
```

Each process has a property called components (itself a bag of processes). The processes in the bag may be other simple or composite processes. As such, they recursively define the composition of simple processes that defines the black-box process CongoBuy.

The expanded CongoBuy process (ExpandedCongoBuy) is comprised of a sequence of two processes, a simple process that locates a book (LocateBook), and a complex process that buys the book (CongoBuyBook). We define them as follows[13]:

```
<rdfs:Class rdf:ID="ExpandedCongoBuy">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about=
      "http://www.daml.org/services/daml-s/2001/05/Process.daml#Sequence"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource=
        "http://www.daml.org/services/daml-s/2001/05/Process.daml#components"/>
      <daml:toClass>
        <daml:Class>
          <daml:intersectionOf rdf:parseType="daml:collection">
            <daml:Restriction>
              <daml:onProperty rdf:resource=
        "http://www.daml.org/services/daml-s/2001/05/Process.daml#firstItem"/>
              <daml:toClass rdf:resource ="#LocateBook"/>
            </daml:Restriction>
            <daml:Restriction>
              <daml:onProperty rdf:resource=
        "http://www.daml.org/services/daml-s/2001/05/Process.daml#secondItem"/>
              <daml:toClass rdf:resource ="#CongoBuyBook"/>
            </daml:Restriction>
          </daml:intersectionOf>
        </daml:Class>
      </daml:toClass>
    </daml:Restriction>
  </daml:intersectionOf>
</rdfs:Class>
```

[12] See http://www.daml.org/services/daml-s/2001/05/Congo.daml. Additional DAML code is needed here to specify the relationship between the bookName property of CongoBuy and the bookSelected property of PutInCart. As of this writing, discussions are underway to determine the best way to indicate this relationship in DAML+OIL.

In the full Congo.daml example, CongoBuyBook is a composite process that is further decomposed, eventually terminating in a composition of simple processes. With this markup we complete our markup to enable automated service invocation.

6.1.3 Automated Service Composition and Interoperation

The DAML-S markup required to automate service composition and interoperation builds directly on the markup for service invocation. In order to automate service composition and interoperation, we must also encode the effects a service has upon the world, and the preconditions for performing that service. For example, when a human being goes to www.congo.com and successfully executes the CongoBuy service, the human knows that they have purchased a book, that their credit card will be debited, and that they will receive a book at the address they provided. Such consequences of Web service execution are not part of the input/output markup we created for automating service invocation.

The process ontology provides precondition and effect properties of a process to encode this information. As with our markup for automated service invocation, we define preconditions and effects both for the black-box process CongoBuy and for each of the simple processes that define its composition, and as with defining inputs and outputs, it is easiest to define the preconditions and effects for each of the simple processes first, and then to aggregate them into preconditions and effects for CongoBuy. The markup is analogous to the markup for input and (conditional) output, but is with respect to the properties precondition and (conditional) effect, instead.

7 Related Efforts

Industry efforts to develop standards for electronic commerce, and in particular for the description of Web-based services currently revolve around UDDI, WSDL, and ebXML [23]. There have also been company-specific initiatives to define architectures for e-commerce, most notably E-speak from Hewlett-Packard.

Nevertheless, we believe that DAML-S provides functionality that the other efforts do not. In comparison to the DAML-S characterization of services, the industry standards mostly focus on presenting a *ServiceProfile* and a *ServiceGrounding* of services (to use DAML-S terminology). *ServiceGroundings* are supported by all the standards. However, they are limited with respect to DAML-S profiles in that they cannot express logical statements, e.g. preconditions and postconditions, or rules to describe dependencies between the profile elements. Input and output types are supported to varying extents. Furthermore, DAML-S supports the description of certain functional attributes of services, which are not covered in the other standards, such as qualityGuarantees and serviceType.

With respect to the four tasks of automatic Web service discovery, automatic Web service invocation, automatic Web service interoperation and composition, and automatic Web service execution monitoring that DAML-S is meant to support, the standards primarily enable the first and the second tasks to a certain extent. These standards are still evolving and it is unclear at present to what extent composition will be addressed. At the moment, the standards do not consider the *ServiceModel* of a service and thus, they also do not support execution monitoring, as defined in this paper.

In the following sections, we look in greater detail at each of these technologies in turn and compare them to DAML-S.

[13] `firstItem` and `secondItem` are easily defined.

7.1 UDDI

UDDI (Universal Description, Discovery and Integration) is an initiative proposed by Microsoft, IBM and Ariba to develop a standard for an online registry, and to enable the publishing and dynamic discovery of Web services offered by businesses [22]. UDDI allows programmers and other representatives of a business to locate potential business partners and form business relationships on the basis of the services they provide. It thus facilitates the creation of new business relationships.

The primary target of UDDI seems to be integration and at least semi-automation of business transactions in B2B e-commerce applications. It provides a registry for registering businesses and the services they offer. These are described according to an XML schema defined by the UDDI specification. A Web service provider registers its advertisements along with keywords for categorisation. A Web services user retrieves advertisements out of the registry based on keyword search. The UDDI search mechanism relies on pre-defined categorisation through keywords and does not refer to the semantic content of the advertisements. The registry is supposed to function in a fashion similar to white pages or yellow pages, where businesses can be looked up by name or by a standard service taxonomy as is already used within the industry. UDDI attempts to cover all kinds of services offered by businesses, including those that are offered by phone or e-mail and similar means; in principle, DAML-S could do this, but it has not been our focus.

Technically speaking, each business description in UDDI consists of a businessEntity element, akin to a White Pages element describing the contact information for a business. A businessEntity describes a business by name, a key value, categorisation, services offered (businessService elements) and contact information for the business. A businessService element describes a service using a name, key value, categorisation and multiple "bindingTemplate" elements. This can be considered to be analogous to a Yellow Pages element that categorises a business. A bindingTemplate element in turn describes the kind of access the service requires (phone, mailto, http, ftp, fax etc.), key values and tModelInstances. tModelInstances are used to describe the protocols, interchange formats that the service comprehends, that is, the technical information required to access the service. It is also used to describe the "namespaces" for the classifications used in categorisation. Many of the elements are optional, including most of the ones that would be required for matchmaking or service composition purposes.

UDDI aims to facilitate the discovery of potential business partners and the discovery of services and their groundings that are offered by known business partners. This may or may not be done automatically. When this discovery occurs, programmers affiliated with the business partners program their own systems to interact with the services discovered. This is also the model generally followed by ebXML. DAML-S enables more flexible discovery by allowing searches to take place on almost any attribute of the ServiceProfile. UDDI, in contrast, allows technical searches only on tModelKeys, references to tModelInstances, which represent full specifications of a kind of service.

UDDI does not support semantic descriptions of services. Thus, depending on the functionality offered by the content language, although agents can search the UDDI registry and retrieve service descriptions, a human needs to be involved in the loop to make sense of the descriptions, and to program the access interface.

Currently, UDDI does not provide or specify content languages for advertisement. Although WSDL is most closely associated with UDDI as a content language, the specification refers to ebXML and XML/edi also as potential candidates. Content languages could be a possible bridge between UDDI and DAML-S. DAML-S is also a suitable candidate for a content language and in this sense, DAML-S and UDDI are complementary. A higher-level service or standard defined on top of UDDI could take advantage of the additional richness of content DAML-S has to offer within the UDDI registries.

7.2 WSDL

WSDL (Web Services Description Language) is an XML format, closely associated with UDDI as the language for describing interfaces to business services registered with a UDDI database. Thus, it is closer to DAML-S in terms of functionality than UDDI. Like DAML-S, it attempts to separate services, defined in abstract terms, from the concrete data formats and protocols used for implementation, and defines bindings between the abstract description and its specific realization [3]. However, the abstraction of services is at a lower level than in DAML-S.

Services are defined as sets of ports, i.e. network addresses associated with certain protocols and data format specifications. The abstract nature of a service arises from the abstract nature of the messages and operations mapped to a port and define its port type. Port types are reusable and can be bound to multiple ports [18]. There are four basic types of operations in WSDL: a one-way, a (two-way) request-response, a (two-way) solicit-response and a (one-way) notification message. A message itself is defined abstractly as a request, a response or even a parameter of a request or response and its type, as defined in a type system like XSD. They can be broken into parts to define the logical break-down of a message.

Messages and operations are defined abstractly and are thus reusable and extensible and correspond roughly to the DAML-S ServiceProfile. The service element itself incorporates both a ServiceProfile and ServiceGrounding information. WSDL service descriptions are not as expressive as DAML-S profiles. Preconditions, postconditions and effects of service access cannot be expressed within WSDL.

Like UDDI, WSDL does not support semantic description of services. WSDL focuses on the grounding of services and although it has a concept of input and output types as defined by XSD, it does not support the definition of logical constraints between its input and output parameters. Thus its support for discovery and invocation of services is less versatile than that of DAML-S.

7.3 E-speak

Hewlett-Packard is collaborating with the UDDI consortium to bring E-speak technology to the UDDI standard. E-speak and UDDI have similar goals in that they both facilitate the advertisement and discovery of services. E-speak is also comparable to WSDL in that it supports the description of service and data types [6]. It has a matching service that compares service requests with service descriptions, primarily on the basis of input-output and service type matching.

E-speak describes services (known as “Resources”) as a set of attributes within several “Vocabularies”. Vocabularies are sets of attributes common to a logical group of services. E-speak matches lookup requests against service descriptions with respect to these attributes. Attributes take common value types such as String, Int, Boolean and Double. There is a base vocabulary which defines basic attributes such as Name, Type (of value String only), Description, Keywords and Version. Currently, there is no semantic meaning attached to any of the attributes. Any matching which takes place is done over the service description attributes which does not distinguish between any further subtypes. DAML-S had a much richer set of attributes; in DAML-S terminology, the input/output parameters, effects and additional functional attributes. In addition, dependencies between attributes and logical constraints on them are not expressible within E-speak.

Unlike UDDI, which was intended to be an open standard from the beginning, e-speak scores relatively low on interoperability. It requires that an e-speak engine be run on all participating client machines. Furthermore, although e-speak is designed to be a full platform for Web services and could potentially expose a execution monitoring interface, service processes remain a black-box for the e-speak platform and consequently no execution monitoring can be done.

7.4 ebXML

ebXML, being developed primarily by OASIS and the United Nations, approaches the problem from a workflow perspective. ebXML uses two views to describe business interactions, a Business Operational View (BOV) and a Functional Service View (FSV) [7] [23]. The BOV deals with the semantics of business data transactions, which include operational conventions, agreements, mutual obligations and the like between businesses. The FSV deals with the supporting services: their capabilities, interfaces and protocols. Although ebXML does not concentrate on only Web services, the focus of this view is essentially the same as that of the current DAML-S effort.

It has the concept of a Collaboration Protocol Profile (CPP) "which allows a Trading Partner to express their supported Business Processes and Business Service Interface requirements [such that they are understood] by other ebXML compliant Trading Partners", in effect a specification of the services offered by the Trading Partner. A Business Process is a set of business document exchanges between the Trading Partners. CPPs contain industry classification, contact information, supported Business Processes, interface requirements etc. They are registered within an ebXML registry, in which there is discovery of other Trading Partners and the Business Processes they support. In this respect, UDDI has some similarities with ebXML. However, ebXML's scope does not extend to the manner in which the business documents are specified. This is left to the Trading Partners to agree upon a priori by the creation of a Collaboration Protocol Agreement.

In conclusion, the kind of functionality, interoperability and dynamic matchmaking capabilities provided by DAML-S is only partially supported, as the standards are currently positioned, by WSDL and UDDI. UDDI may become more sophisticated as it incorporates e-speak-like functionalities, but it will not allow automatic service interoperability until it incorporates the information provided by DAML-S.

8 Summary and Current Status

DAML-S is an attempt to provide an ontology, within the framework of the DARPA Agent Markup Language, for describing Web services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints. We have released an initial version of DAML-S. It can be found at the URL: http://www.daml.org/services/daml-s

We expect to enhance it in the future in ways that we have indicated in the paper, and in response to users' experience with it. We believe it will help make the Semantic Web a place where people can not only find out information but also get things done.

Acknowledgments

The authors have profited from discussions about this work with Ron Fadel, Richard Fikes, Jessica Jenkins, James Hendler, Mark Neighbors, Tran Cao Son, and Richard Waldinger. The research was funded by the Defense Advanced Research Projects Agency as part of the DARPA Agent Markup Language (DAML) program under Air Force Research Laboratory contract F30602-00-C-0168 to SRI International, F30602-00-2-0579-P00001 to Stanford University, and F30601-00-2-0592 to Carnegie Mellon University. Additional funding was provided by Nokia Research Center.

References

[1] J. F. Allen and H. A. Kautz. A model of naive temporal reasoning. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 251–268. Ablex Publishing Corp., 1985.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[4] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *IJCAI97*, 1997.

[5] G. Denker, J. Hobbs, D. Martin, S. Narayanan, and R. Waldinger. Accessing information and services on the daml-enabled web. In *Proc. Second Int'l Workshop Semantic Web (SemWeb'2001)*, 2001.

[6] E-Speak. E-Speak Architectural Specification Release A.0. http://www.e-speak.hp.com/media/a0/architecturea0.pdf, 2001.

[7] ebXML. ebXML Web Site. http://www.ebXML.org/, 2000.

[8] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

[9] M. Ghallab et. al. Pddl-the planning domain definition language v. 2. Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[10] J. Hendler and D. L. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, 15(6):72–73, 2001.

[11] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A Logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, April-June 1997.

[12] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 13(1-2):92–128, 1999.

[13] S. McIlraith, T. C. Son, and H. Zeng. Mobilizing the web with daml-enabled web service. In *Proc. Second Int'l Workshop Semantic Web (SemWeb'2001)*, 2001.

[14] S. McIlraith, T. C. Son, and H. Zeng. Semantic web service. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[15] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[16] R. Milner. Communicating with Mobile Agents: The pi-Calculus. Cambridge University Press, Cambridge, 1999.

[17] S. Narayanan. Reasoning about actions in narrative understanding. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI'1999)*, pages 350–357. Morgan Kaufman Press, San Francisco, 1999.

[18] U. Ogbuji. Using WSDL in SOAP applications: An introduction to WSDL for SOAP programmers. http://www-106.ibm.com/developerworks/library/ws-soap/?dwzone=ws, 2001.

[19] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, and J. Lee. The Process Specification Language (PSL): Overview and version 1.0 specification. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD., 2000.

[20] K. Sycara and M. Klusch. Brokering and matchmaking for coordination of agent societies: A survey. In A. e. a. Omicini, editor, *Coordination of Internet Agents*. Springer, 2001.

[21] K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*, 28(1):47–53, 1999.

[22] UDDI. The UDDI Technical White Paper. http://www.uddi.org/, 2000.

[23] D. Webber and A. Dutton. Understanding ebXML, UDDI and XML/edi. http://www.xml.org/feature_articles/2000_1107_miller.shtml, 2000.

[24] H.-C. Wong and K. Sycara. A Taxonomy of Middle-agents for the Internet. In *ICMAS'2000*, 2000.

Searching for Services on the Semantic Web Using Process Ontologies

Mark Klein
Center for Coordination Science
Massachusetts Institute of Technology
m_klein@mit.edu

Abraham Bernstein
Stern School of Business
New York University
bernstein@stern.nyu.edu

Abstract. The ability to rapidly locate useful on-line services (e.g. software applications, software components, process models, or service organizations), as opposed to simply useful documents, is becoming increasingly critical in many domains. As the sheer number of such services increases it will become increasingly more important to provide tools that allow people (and software) to quickly find the services they need, while minimizing the burden for those who wish to list their services with these search engines. This can be viewed as a critical enabler of the 'friction-free' markets of the 'new economy'. Current service retrieval technology is, however, seriously deficient in this regard. The *information retrieval* community has focused on the retrieval of documents, not services per se, and has as a result emphasized keyword-based approaches. Those approaches achieve fairly high recall but low precision. The *software agents* and *distributed computing communities* have developed simple 'frame-based' approaches for 'matchmaking' between tasks and on-line services increasing precision at the substantial cost of requiring all services to be modeled as frames and only supporting perfect matches. This paper proposes a novel, ontology-based approach that employs the characteristics of a process-taxonomy to increase recall without sacrificing precision and computational complexity of the service retrieval process.

1 The Challenge

Increasingly, the Semantic Web will be called upon to provide access not just to static *documents* that collect useful *information*, but also to *services* that provide useful *behavior*. Potential examples of such services abound:

- *Software applications* (e.g. for engineering, finance, meeting planning, or word processing) that can invoked remotely by people or software
- *Software components* that can be downloaded for use when creating new applications
- *Process models* that describe how to achieve some goal (e.g. eCommerce business models, material transformation processes, etc)
- *Individuals* or *organizations* who can perform particular functions, e.g. as currently brokered using such web sites as guru.com, elance.com and freeagent.com.

As the sheer number of such services increase it will become increasingly important to provide tools that allow people (and software) to quickly find the services they need, while

minimizing the burden for those who wish to list their services with these search engines [1]. This paper describes a set of ideas, based on the sophisticated use of process ontologies, for creating improved service retrieval technologies.

2 Contributions and Limitations of Current Technology

Current service retrieval approaches have serious limitations with respect to meeting the challenges described above. They either perform relatively poorly or make unrealistic demands of those who wish to index or retrieve services. We review these approaches below.

Service retrieval technology has emerged from several communities. The information retrieval community has focused on the retrieval of documents, not services per se, and has as a result emphasized keyword-based approaches. The software agents and distributed computing communities have developed simple 'frame-based' approaches for 'matchmaking' between tasks and on-line services. The software engineering community has developed by far the richest set of techniques for service retrieval [2]. We can get a good idea of the relative merits of these approaches by placing them in a *precision/recall* space (Figure 1):

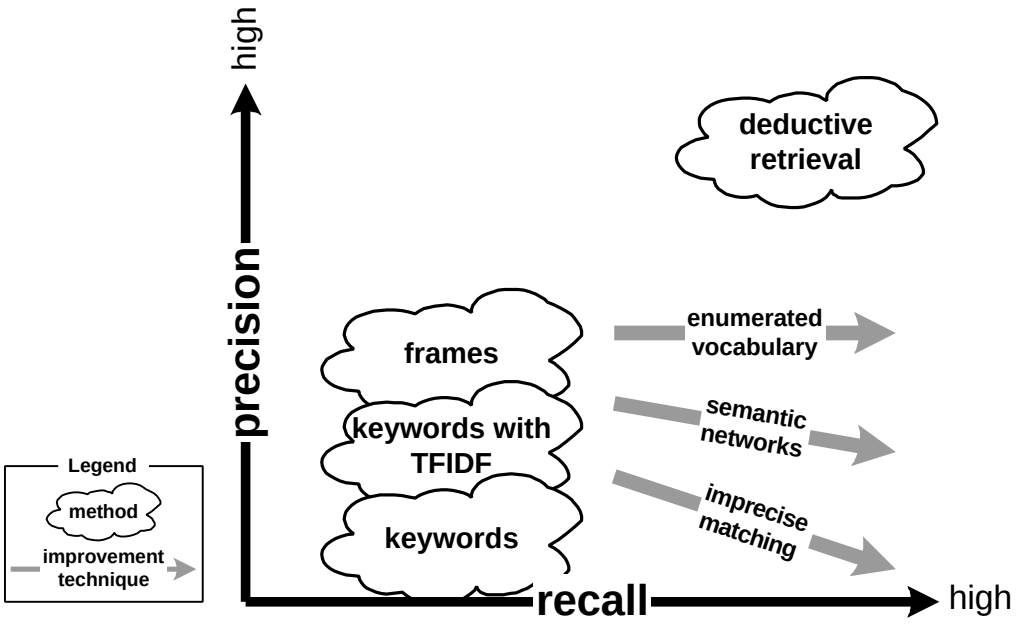


Figure 1: The state of the art in service retrieval.

Recall is the extent to which a search engine retrieves *all* of the items that one is interested in (i.e. avoiding false negatives) while *precision* is the extent to which the tool retrieves *only* the items that one is interested in (i.e. avoiding false positives).

Most search engines look for items (e.g. web pages) that contain the keywords in the query. More sophisticated variants (based on the technique known as TFIDF) look for items in which the searched-for keywords are more common than usual, thereby increasing precision [3]. Typically, no manual effort is needed to list items with such search engines, and queries can be specified without needing to know a specialized query language. Keyword-based approaches are, however, notoriously prone to both low precision and imperfect recall. Many completely

irrelevant items may include the keywords in the query, leading to low precision. It is also possible that the query keywords are semantically equivalent but syntactically different from the words in the searched items, leading to reduced recall. Imagine , for example, that we are searching for a service that can offer a loan to cover a $100,000 house addition. Entering the keywords "loan house 100000" into google.com (a keyword-based search service), for example, returns 2,390 documents. While the first two results are promising (a loan calculator and a mortgage calculator somewhat connected to a loan granting organization), the third hit points to a report of a campaign against arms trade, and the fourth shows a junior high-school math-project on how to calculate mortgages. If we enter the same query in altavista.com (which uses TFIDF) we get 24,168,519 'hits'. While the first few hits talk about loans, they do not all provide a loan service. Most of them point to classes that discuss loan calculation techniques and provide some sort of a mortgage calculator.

It is of course somewhat misleading to use web-search engines to assess the likely performance of a keyword-based service retrieval engine. The web contains much more then services. Many documents (like the loan calculation classes mentioned above) have nothing to do with the provision of services. Nevertheless we can take the poor precision of those queries as an indicator of what would happen in a system that relies solely on these techniques for retrieving services. A mortgage calculator is a useful instrument in itself. It does not, however, provide the service of creditworthiness analysis or loan provision. We can, therefore, assume that systems using these techniques would still show low precision.

Several techniques have been developed to address these problems. One is to require that items and queries be described using the same, pre-enumerated, vocabulary [4]. This increases the probability that the same terms will be used in the query and desired items, thereby increasing recall. Another approach is to use semantic nets (e.g. WordNet [5]) that capture the semantic relationships between words (e.g. synonym, antonym, hypernym and hyponym) to increase the breadth of the query and thereby increase recall [6] but this potentially can reduce precision and can result in suboptimal recall because these networks focus on purely linguistic relationships. Search engines like google.com [7] prioritize retrieved documents according to whether they are linked to documents that also contain the searched-for keywords, as a way of increasing precision. Finally, most text-based search engines allow for imprecise matching (e.g. retrieving items that contain some but not all of the query keywords), potentially increasing recall but again at the cost of reduced precision.

We can see then that keyword-based approaches can achieve fairly high recall but at the cost of low precision. The key underlying problem is that keywords are a poor way to capture the semantics of a query or item. If this semantics could be captured more accurately then precision would increase. *Frame*-based approaches [8] [9] [10] [11] [12] have emerged as a way of doing this. A frame consists of attribute value pairs describing the properties of an item. Figure 2 for example shows a frame for an integer averaging service:

Both items and queries are described using frames: matches represent items whose (textual) property values match those in the query. All the commercial service search technologies we are aware of (e.g. Jini, eSpeak, Salutation, UDDI [13]) use the frame-based approach, typically with an at least partially pre-enumerated vocabulary of service types and properties. The more sophisticated search tools emerging from the research community (e.g. LARKS [14]) also make limited use of semantic nets, e.g. returning a match if the input type of a service is equal to *or* a generalization of the input type specified in the query. Frame-based approaches thus do increase precision at the (fairly modest) cost of requiring that all services be modeled as frames.

Description	a service to find the average of a list of integers
Input	integers
Output	real
Execution Time	number of inputs * 0.1 msec

Figure 2: A frame-based description of an integer sorting service.

The frame-based approach is taken one step further in the *deductive retrieval* approach [15] [16] [17] wherein service properties (e.g. inputs, outputs, function, and performance) are expressed *formally* using logic (Figure 3):

Name:	set-insert
Syntax:	set-insert(Elem, Old, New)
Input-types:	(Elem:Any), (Old:SET)
Output-types:	(New: SET)
Semantics:	
Precond:	$\neg member(Elem, Old)$
	$member(Elem, New)$
Postcond:	$\wedge \forall x(member(x, Old) \rightarrow member(x, New)$
	$\wedge \forall y(member(y, New) \rightarrow (member(y, old) \vee y = Elem)$

Figure 3: A service description for deductive retrieval [15]

Retrieval then consists of finding the items that can be *proved* to achieve the functionality described in the query. If we assume a non-redundant pre-enumerated vocabulary of logical predicates and a complete formalization of all relevant service and query properties, then deductive retrieval can in theory achieve both perfect precision and perfect recall. This approach, however, faces two very serious practical difficulties. First of all, it can be prohibitively difficult to model the semantics of non-trivial queries and services using formal logic. Even the simple set-insert function shown above in Figure 3is non-trivial to formalize correctly: imagine trying to formally model the behavior of Microsoft Word or an accounting package! The second difficulty is that the proof process implicit in this kind of search can have a high computational complexity, making it extremely slow [15]. Our belief is that these limitations, especially the first one, make deductive retrieval unrealistic as a scalable general purpose service search approach.

Other approaches do exist, but they apply only to specialized applications. One is execution-based retrieval, wherein software components are selected by comparing their actual I/O behavior with the desired I/O behavior. This is approach is suitable only for contexts where observing a few selected samples of I/O behavior are sufficient to prune the service set [18] [19] [20].

3 Our Approach: Exploiting Process Ontologies

Our challenge, as we have seen, can be framed as being able to capture enough service and query semantics to substantively increase precision without reducing recall or making it unrealistically difficult for people to express these semantics. *Our central claim is that these goals can be achieved through the sophisticated use of process ontologies*. We begin by capturing the function(s) of a service as a process model. The service model is then indexed (to facilitate its subsequent retrieval) by placing it and all its components (subtasks and so on) into the appropriate sections of the ontology. Queries are also expressed as (partial) process models. The matching algorithm then finds all the services whose process models match that of the query, using the semantic relationships encoded in the process ontology. Our approach can thus be viewed as having the following functional architecture:

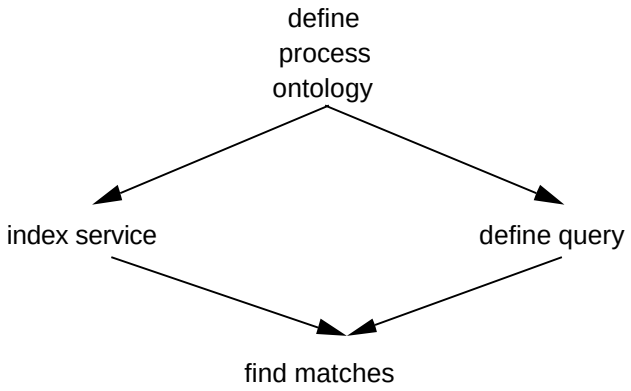


Figure 4: Functional architecture of our proposed service retrieval technology.

We will consider each element of the functional architecture in the sections below.

3.1 *Define Process Ontology*

Our approach differs from previous efforts in that it is based on highly expressive process models arranged into a fully-typed process ontology. The key concepts underlying this ontology are an extension of those developed by the MIT Process Handbook project. The Handbook is a process knowledge repository which has been under development at the Center for Coordination Science (CCS) for the past eight years [21] [22]. The Handbook is under active use and development by a highly distributed group of more than 40 scientists, teachers, students and sponsors for such diverse purposes as adding new process descriptions, teaching classes, and business process re-design. We believe the current Handbook process ontology represents an excellent starting point for indexing on-line services because it is focused on business processes which is what a high proportion of on-line services are likely to address.

The Handbook takes advantage of several simple but powerful concepts to capture and organize process knowledge: *attributes, ports, decomposition, dependencies, exceptions* and *specialization.*

> *Process Attributes:* Like most process modeling techniques, the Handbook allows processes to be annotated with attributes that capture such information as a textual

description, typical performance values (e.g. how long a process takes to execute), as well as pre-, post- and during- conditions.

Decomposition: Also like most process modeling techniques, the Handbook uses the notion of *decomposition*: a process is modeled as a collection of activities that can in turn be broken down ("decomposed") into subactivities.

Ports: Ports describe the I/O-behavior of an activity. They describe the types of resources the activity uses and produces, and are as a result important for assessing the match between a service specification and a query.

Dependencies: Another key concept we use is that coordination can be viewed as the management of *dependencies* between activities [21]. Every dependency can include an associated *coordination mechanism*, which is simply the process that manages the *resource* flow and thereby coordinates the activities connected by the dependency. Task inputs and outputs are represented as *ports* on those tasks. A key advantage of representing processes using these concepts is that they allow us to highlight the 'core' activities of a process and abstract away details about how they coordinate with each other, allowing more compact service descriptions without sacrificing significant content.

Exceptions: Processes typically have characteristic ways they can fail and, in at least some cases, associated schemes for anticipating and avoiding or detecting and resolving them. This is captured in our approach by annotating processes with their characteristic 'exceptions', and mapping these exceptions to processes describing how these exceptions can be handled [23].

Specialization: The final key concept is that processes and all their key elements (ports, resources, attributes, and exceptions) appear in type taxonomies, with very generic classes at one extreme and increasingly *specialized* ones at the other, so process models are fully typed. The taxonomies place items with similar semantics (e.g. processes with similar purposes) close to each other, the way books with similar subjects appear close to each other in a library: Processes that vary along some identifiable dimension can be grouped into *bundles*; where processes can appear in more than one bundle. Bundles subsume the notion of 'faceted' classification [4], well-known in the software component retrieval community, because bundles, unlike facets, can include a whole ontology branch as opposed to simply a flat set of keywords, allowing varying abstraction levels and the use of synonyms.

As shown in Figure 5, an activity is thus defined by specifying its decomposition (i.e., the activities it contains), its interface (as defined by the ports it contains), the dependencies between its sub-activities, and the attributes defined for each of those entities (not shown). Each activity can be linked to the kinds of exceptions it can face, and these exceptions can be linked in turn to the processes if any used to handle (anticipate and avoid, or detect and resolve) them.

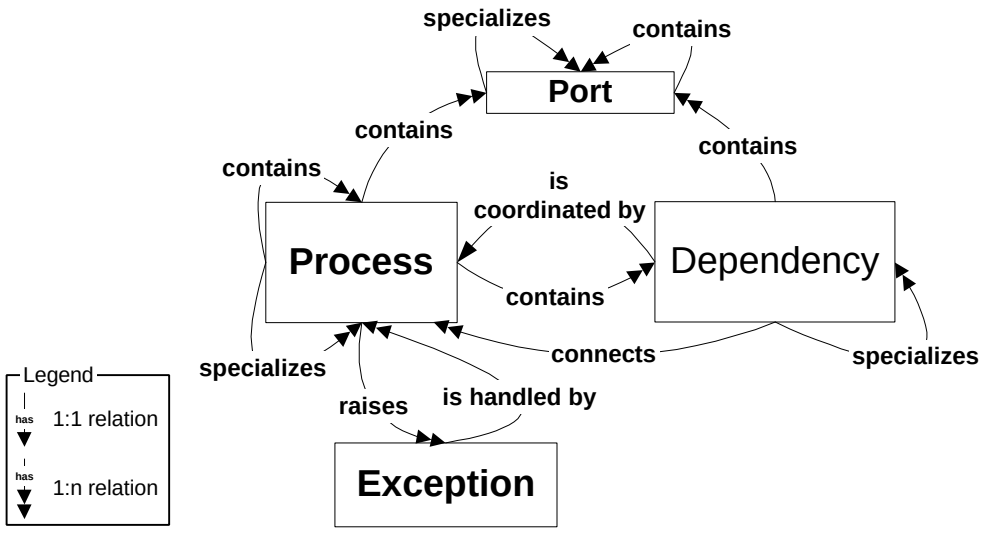


Figure 5: Partial meta-model for our process ontology (attributes not shown).

Every type of entity has its own specialization hierarchy into which it can be placed, making this a fully-typed process description approach. There is thus a specialization hierarchy for processes, resources, exceptions and so on. The "Sell loan" process, for example, is a specialization of the more general "Sell financial service" process. It, furthermore, specializes into more specific processes such as "Sell reserve credit," "Sell Credit Card," and "Sell mortgage" (Figure 6):

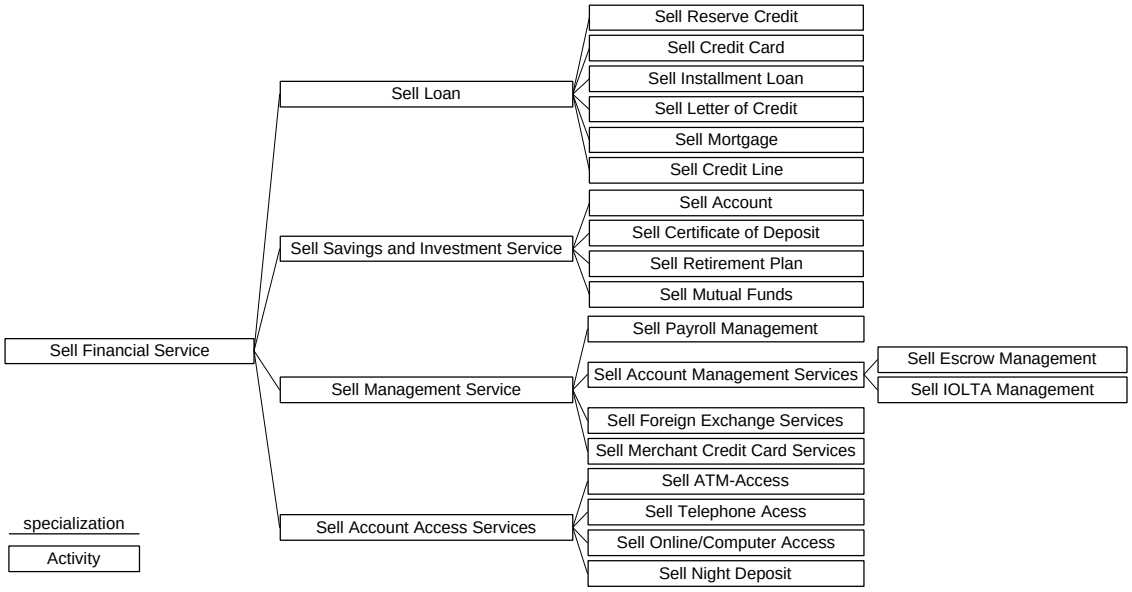


Figure 6: Specialization hierarchy for grant loan

Each of the elements of the "Sell loan" process is also a member of the overall specialization hierarchy. "Analyze credit-worthiness," for example, is a specialization of the more general

"Perform financial analysis" and specializes to more specific processes such as "Analyze creditworthiness using scoring" and "Analyze creditworthiness using PE-ratio," etc.

This process representation has equal or greater formal expressiveness than other full-fledged process modeling languages (e.g. IDEF [24], PIF [25], PSL [26] or CIMOSA [27]) as well as greater expressiveness than the frame-based languages used in previous service retrieval efforts, by virtue of adding such important concepts as full typing, resource dependencies, ports, task decompositions, and exceptions.

The growing Handbook database currently includes over 5000 process descriptions ranging from specific (e.g. for a university purchasing department) to generic (e.g. for resource allocation and multi-criteria decision making). A subset of this database (containing a representative selection of process models but no exception-related information) is accessible over the Web at http://ccs.mit.edu/eph/

3.2 *Index Services*

Services are indexed into the process ontology so that they may be retrieved readily later on. Indexing a service in our approach comes down to placing the associated process model, and all of its components (attributes, ports, dependencies, subtasks and exceptions) in the appropriate place in the ontology. The fact that a substantial process ontology exists in the Process Handbook means that parties wishing to index a service can construct their service specification from already existing elements in the ontology, and then customizing them as necessary.

Imagine for example that we want to index a service that sells mortgages. The Handbook ontology already includes a general 'sell loan' process (Figure 7):

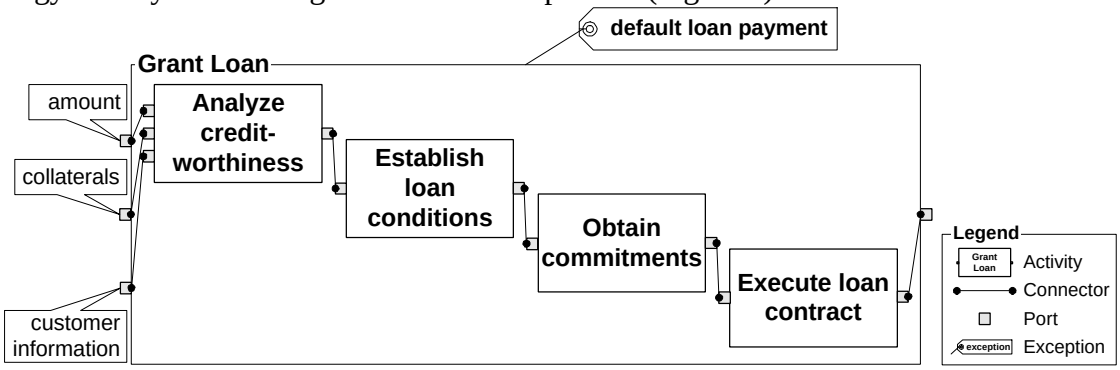


Figure 7: The loan selling service process model.

The 'sell loan' process model includes most of the requisite elements, including key sub-activities such as analyzing creditworthiness, establishing the loan's conditions, obtaining the commitments, and "executing" the loan contract. It also includes ports describing the process inputs (e.g. the size of the loan ("amount"), the collateral provided for the loan, information about the customer) and outputs (the loan itself).[1] The mortgage provider, therefore, need only create a specialization of the 'sell loan' process and make the few changes specific to mortgage loans, e.g. further specifying the types of customer information it wishes and type of creditworthiness analysis performed, elements that can also be defined by specializing existing elements in the process ontology. (Figure 8):

[1] To simplify this example we omitted dependencies and exceptions.

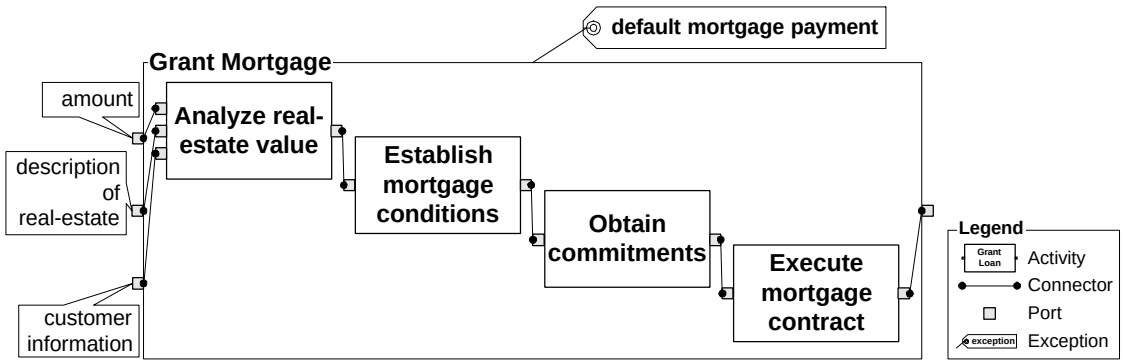


Figure 8: The mortgage selling service process model.

The Handbook project has developed sophisticated Windows-based tools for process indexing, based on this define-by-specialization principle, that have been refined over nearly eight years of daily use. Using these tools, most process models can indexed in a matter of minutes.

Note that when modeling a service, one needs in theory to account for all the possible uses the service may be put to. What to some users may be a side effect of a service may be a key attribute for others. To pick a homely example, a dog-walking service may be desired as a way to provide the dog exercise, or simply as a way to make sure the dog is cared for while the owner is out on an errand. This would suggest that a process model needs to be indexed under specializations representing all of its possible uses, which is difficult at best. As we shall see below, however, we can define query algorithms that, using query mutation operators, can potentially retrieve processes not explicitly indexed as serving a given purpose, thereby reducing the service-indexing burden.

We can, however, take the manual indexing approach described above one important step further. A key criterion for a successful service retrieval approach is minimizing the manual effort involved in listing new services with the search engine. Ideally services can be classified automatically. Automatic classification is predicated on having a similarity metric so one can place a process under the class it is most similar to. Previous efforts for automatic service classification have used similarity metrics based on either:

- Automatically calculated word frequency statistics for the natural language documentation of the service [28] [29] [30]
- Manually developed frame-based models of the service [9]

The first approach is more scalable because no manual effort is needed, but the latter approach results in higher retrieval precision because frame-based models, as we have seen, better capture service semantics. We can use a ontology-based approach to improve on these approaches by developing tools for [semi-] automated classification of process models. Process models, in the form of flowcharts for example, are typically created as part of standard programming practice, so in many if not most cases the services we want to index will already have process models defined for them. These models may even in some cases already be classified into some kind of process ontology. The greater expressiveness of process models as compared to standard frame-based service models (process models often include task decomposition information, for example, but frame-based models do not) means that we can

define similarity metrics that are more accurate than those that have been used previously. These metrics can use word frequency statistics augmented with semantic networks to compute similarity of natural language elements, combined with taxonomic reasoning and graph-theoretic similarity measures to assess structural and type similarities. At the least, we believe that this technique should greatly speed manual service indexing by providing the human user with a short list of possible locations for a service. Ideally, it will allow a human user to be taken out of the loop entirely. If fully automated indexing turns out to be feasible, then we can imagine on-line services being indexed using Internet "worms" analogous to those currently used by keyword-based search tools.

3.3 *Define Queries*

It is of course possible that we can do without query definition entirely once services have been indexed into a process ontology. In theory one can simply browse the ontology to find the services that one is interested in, as in [31]. Our own experience with the Process Handbook suggests however that browsing can be slow and difficult for all except the most experienced human users, because of the size of the process ontology. This problem is likely to be exacerbated when, as with online services, the space of services is large and dynamic.

To address this challenge we have begun to define a query language called PQL (the Process Query Language) designed specifically for retrieving full-fledged process models from a process ontology [32]. Process models can be straightforwardly viewed as entity-relationship diagrams made up of entities like tasks connected by relationships like 'has-subtask'. PQL queries are built up as combinations of two types of clauses that check for given entities and relationships:

Entity <entity> isa <entity type> :test <predicate>
Relationship <source entity> <relationship type> <target entity> :test <predicate>

The first clause type matches any entity of a given type. The second primitive matches any relationship of a given type between two entities. Any bracketed item <> can be replaced by a variable (with the format ?<string>) that is bound to the matching entity and passed to subsequent query clauses. The predicates do further pruning of what constitutes a match.

Let us consider some simple examples. Imagine we are searching for a loan service that accepts real estate as collateral (which includes mortgages), as follows:

```
Entity ?proc isa sell-loan                                  // finds a specialization of “Sell loan”

Relationship ?proc has-port ?port1                          // looks for a port that accepts real-
Entity ?port1 isa input-port                                // estate as a collateral
Relationship ?port1 propagates-resource ?res1
Entity ?res1 isa real-estate
Entity ?port1 has-attribute ?attr1
Entity ?attr1 isa name
Relationship ?attr1 has-value ?val :test (= ?val “collateral”)

Relationship ?proc has-port ?port2                          // looks for a port that “produces”
Entity ?port2 isa output-port                               // money
Relationship ?port2 propagates-resource ?res2
Entity ?res2 isa money
```

Query 1: A query for mortgage services.

This type of query is imaginably within the capabilities of frame-based query languages, especially if enhanced with a semantic network such as WordNet [5], since it references only the type and I/O for a service. Existing linguistically-oriented semantic networks do not, however, include a service-oriented process taxonomy like that incorporated in the Handbook, so we can expect that recall will be less than optimal using this approach.

We can go substantially further, however, using an ontology-based approach. Query 2 below, for example, retrieves a financing solution that has at least one internet-based step:

```
Entity ?proc isa sell-loan                                  // finds a specialization of “Sell loan”

Relationship ?proc has-subtask ?sub                         // has sub-process that is internet
Entity ?sub isa internet-process                            // based

Relationship ?proc has-port ?port1                          // looks for a port that accepts real-
Entity ?port1 isa input-port                                // estate as a collateral
Relationship ?port1 propagates-resource ?res1
Entity ?res1 isa real-estate
Entity ?port1 has-attribute ?attr1
Entity ?attr1 isa name
Relationship ?attr1 has-value ?val :test(= ?val “collateral”)

Relationship ?proc has-port ?port2                          // looks for a port that “produces”
Entity ?port2 isa output-port                               // money
Relationship ?port2 propagates-resource ?res2
Entity ?res2 isa money
```

Query 2: A query for internet-based financing services.

This useful capability – searching for services based on *how* they achieve their purpose – can *not* be achieved using frame-based languages since they do not capture process decompositions.

Another novel and powerful feature of the ontology-based approach is the ability to search based on how the service handles *exceptions*. Query 2 could be refined, for example, by adding a clause that searches for loan processes that provide insurance for handling payment defaults:

```
Entity ?proc isa sell-loan
…
Relationship ?proc has-exception ?exc
Entity ?exc isa payment-default
Relationship ?exc is-handled-by ?proc2
Entity ?proc2 isa insurance-process
```

Query 3: A query refinement that searches for payment default insurance.

Searching based on a services' exception handling processes is also not well-supported by frame-based techniques. We can see, in short, that an ontology-based approach offers substantively greater query expressiveness than keyword or frame-based approaches, by virtue of having a substantive process taxonomy, and by being able to represent process decompositions and exception handling.

PQL, like any query language, is fairly verbose and requires that users be familiar with its syntax. More intuitive interfaces suitable for human users are possible. One approach exploits the process ontology. It is straightforward to translate any process model into a PQL query that looks for a service with that function. Users can thus use the Handbook process modeling tools to express a query as a combination and refinement of existing elements in the process ontology, in much the same way new services are indexed. An advantage of specifying a query in this way is that the elements in the process ontology can give the user additional ideas concerning what to look for. Someone looking for a mortgage service, for example, might not initially think to check for whether or not that service provides mortgage insurance. If they define the query as a specialization of the generic 'sell loan' process, however, they may note that that process has a possible 'payment default' exception, and they mya as a result be inspired to search for mortgage services that handle that exception in a way the user prefers.

Other possibilities exist. One can reduce the query definition burden by allowing users to enter queries using a restricted subset of English (an approach that has been applied successfully to traditional database access). Substantial progress has also been made in defining graphical metaphors for query definition (e.g. see [33]). These approaches can of course be combined, so that for example a simple natural language query returns some candidates classes that are selected from and refined to define the final, more complete, query.

3.4 *Find Matches*

The algorithm for retrieving matches given a PQL query is straightforward. The clauses in the PQL query are tried in order, each clause executed in the variable binding environment accumulated from the previous clauses. The sets of bindings that survive to the end represent the matching services. There is one key problem, however, that has to be accounted for; what we can call *modeling differences*. It is likely that in at least some cases a service may be modeled in a way that is semantically equivalent to but nevertheless does not syntactically match a given PQL query. The service model may, for example, include a given subtask several levels down the process decomposition, while in the query that subtask may be just one level down. The service

model may express using several resource flows what is captured in the query as a single more abstract resource flow. The service model may simply be missing some type or resource flow information tested for in the query. This problem is exacerbated by the possibility of multiple service ontologies. As has been pointed out in [34], while we can expect the increasing prevalence of on-line ontologies to structure all kinds of knowledge including service descriptions, there will almost certainly be many partially-mapped ontologies as opposed to a single universally adopted one. This will likely increase the potential for modeling differences, e.g. if queries and services are defined using different ontologies. In order to avoid false negatives we must therefore provide a retrieval scheme that is tolerant of such modeling differences.

We can explore for this purpose the use of semantics-preserving query mutation operators. Imagine we have a library of operators that can syntactically mutate a given PQL query in a way that (largely) preserves its semantics. Some examples of such operators include:

(a) allow a type specification to be more general
(b) allow a subtask to be any number of levels down the task decomposition hierarchy
(c) allow 'siblings' or 'cousins' of a task to constitute a match
(d) relax the constraints on a parameter value
(e) remove a subtask

One can use mutation operators to broaden a search to allow for the discovery of novel alternatives. One could for example apply mutation operator (a) to Query 4 above so that it searches for "Sell" processes, not just "Sell loan" processes. This would result in additional hits like "Sell real-estate," a service which would sell the property to raise money, or "Sell real-estate lease," which would lease the property to generate funds. Assuming a homeowner would want to raise money for an extension, the first of those two additional options would not be desirable, as the property would be sold off. The second option, though, is often used. The key point to be made here is that the mutation operation broadened the search in a way that was *semantically motivated*, thereby avoiding the precipitous decline in precision typical of other imprecise matching techniques.

A second important potential use of mutation operators is to address the incomplete indexing issue described in section 3.2. It will often be the case that a user may be searching for all processes that serve a purpose not represented by any single element in the process ontology. For example, one may be searching for all the ways to raise money, but the different options may not all share a single generalization. One can address this issue by using mutation operators to define a query that searches for processes with *similar substeps*, regardless of what part of the process ontology they are indexed in. So we can, for example, search for all processes that offer money given some kind of collateral. This would return such options as services for getting a mortgage, raising money on the stock market, and so on.

4 Contributions of this Work

This paper has described a set of ideas that exploit the use of process model representations of service semantics, plus process ontologies, to improve service retrieval. These ideas offer the potential for the following important benefits:

Increased Precision: Our approach differs from previous efforts in that it models service semantics more fully than keyword or frame-based approaches, without imposing the unrealistic modeling burden implied by deductive retrieval approaches. This translates into greater retrieval precision.

Increased Recall: Modeling differences between queries and service descriptions can reduce recall, but this can be addressed in a novel way through the use of semantics-preserving query mutation operators.

While query definition and service indexing using process models is potentially more burdensome than simply entering a few keywords, we believe that existing "define-by-specialization" process modeling techniques developed in the Handbook project, coupled with our proposed advances in search algorithms and automated process model classification, should result in an acceptable increase in the query definition and service indexing burden when traded off against the increase in retrieval precision and recall.

To date we have developed an initial version of a PQL interpreter as well as a small set of semantics-preserving query mutation operators [31], which has demonstrated the viability of the query-by-process-model concept. While we have not yet evaluated PQL's performance in detail yet, it is clear that its primitive query elements have low computational complexity. Query performance can in addition be increased by using such well-known techniques as query clause re-ordering. Our future efforts will involve comparing the precision and recall of our approach with other search engines, refining the query definition and mutation schemes, and implementing and evaluating automated process classification techniques.

5 References

1. Bakos, J.Y., *Reducing Buyer Search Costs: Implications for Electronic Marketplaces.* Management Science, 1997. **43**.
2. Mili, H., F. Mili, and A. Mili, *Reusing software: issues and research directions.* IEEE Transactions on Software Engineering, 1995. **21**(6): p. 528-62.
3. Salton, G. and M.J. McGill, *Introduction to modern information retrieval*. McGraw-Hill computer science series. 1983, New York: McGraw-Hill. xv, 448.
4. Prieto-Diaz, R., *Implementing faceted classification for software reuse.* 12th International Conference on Software Engineering, 1990. **9**: p. 300-4.
5. Magnini, B., *Use of a lexical knowledge base for information access systems.* International Journal of Theoretical & Applied Issues in Specialized Communication, 1999. **5**(2): p. 203-228.
6. Brin, S. and L. Page. *The anatomy of a large-scale hypertextual Web search engine*. in *Computer Networks & ISDN System*. 1998. Netherlands: Elsevier.
7. Henninger, S., *Information access tools for software reuse.* Journal of Systems & Software, 1995. **30**(3): p. 231-47.
8. Fernandez-Chamizo, C., et al., *Case-based retrieval of software components.* Expert Systems with Applications, 1995. **9**(3): p. 397-421.
9. Fugini, M.G. and S. Faustle. *Retrieval of reusable components in a development information system*. in *Second International Workshop on Software Reusability*. 1993: IEEE Press.

10. Devanbu, P., et al., *LaSSIE: a knowledge-based software information system.* Communications of the ACM, 1991. **34**(5): p. 34-49.
11. ESPEAK, *Hewlett Packard's Service Framework Specification.* 2000, HP Inc.
12. Richard, G.G., *Service advertisement and discovery: enabling universal device cooperation.* IEEE Internet Computing, 2000. **4**(5): p. 18-26.
13. Sycara, K., et al. *Matchmaking Among Heterogeneous Agents on the Internet*. in *AAAI Symposium on Intelligent Agents in Cyberspace*. 1999: AAAI Press.
14. Meggendorfer, S. and P. Manhart. *A Knowledge And Deduction Based Software Retrieval Tool*. in *6th Annual Knowledge-Based Software Engineering Conference*. 1991: IEEE Press.
15. Chen, P., R. Hennicker, and M. Jarke. *On the retrieval of reusable software components.* in *Proceedings Advances in Software Reuse. Selected Papers from the Second International Workshop on Software Reusability*. 1993.
16. Kuokka, D.R. and L.T. Harada, *Issues and extensions for information matchmaking protocols.* International Journal of Cooperative Information Systems, 1996. **5**: p. 2-3.
17. Podgurski, A. and L. Pierce, *Retrieving reusable software by sampling behavior.* ACM Transactions on Software Engineering & Methodology, 1993. **2**(3): p. 286-303.
18. Hall, R.j. *Generalized behavior-based retrieval (from a software reuse library)*. in *15th International Conference on Software Engineering*. 1993.
19. Park, Y., *Software retrieval by samples using concept analysis.* Journal of Systems & Software, 2000. **54**(3): p. 179-83.
20. Malone, T.W. and K.G. Crowston, *The interdisciplinary study of Coordination. ACM Computing Surveys*, 1994. **26**(1): p. 87-119.
21. Malone, T.W., et al., *Tools for inventing organizations: Toward a handbook of organizational processes.* Management Science, 1999. **45**(3): p. 425-443.
22. Klein, M. and C. Dellarocas, *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems.* Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems., 2000. **9**(3/4).
23. NIST, *Integrated Definition for Function Modeling (IDEF0)*. 1993, National Institute of Standards and Technology.
24. Lee, J., et al. *The PIF Process Interchange Format and Framework Version 1.1*. in *European Conference on AI (ECAI) Workshop on Ontological Engineering*. 1996. Budapest, Hungary.
25. Schlenoff, C., et al., *The essence of the process specification language.* Transactions of the Society for Computer Simulation, 1999. **16**(4): p. 204-16.
26. Kosanke, K., *CIMOSA: Open System Architecture for CIM*. 1993: Springer Verlag.
27. Frakes, W.b. and B.a. Nejmeh, *Software reuse through information retrieval.* Proceedings of the Twentieth Hawaii International Conference on System Sciences, 1987. **2**: p. 6-9.
28. Maarek, Y.s., D.M. Berry, and G.e. Kaiser, *An information retrieval approach for automatically constructing software libraries.* IEEE Transactions on Software Engineering, 1991. **17**(8): p. 800-13.
29. Girardi, M.R. and B. Ibrahim, *Using English to retrieve software.* Journal of Systems & Software, 1995. **30**(3): p. 249-70.

30. Latour, L. and E. Johnson. *Seer: a graphical retrieval system for reusable Ada software modules*. in *Third International IEEE Conference on Ada Applications and Environments.* 1988: IEEE Comput. Soc. Press.
31. Klein, M. *An Exception Handling Approach to Enhancing Consistency, Completeness and Correctness in Collaborative Requirements Capture*. 1996.
32. Hendler, J., *Agents on the Semantic Web*. 2001.

A Semantic Web Approach to Service Description for Matchmaking of Services

David Trastour, Claudio Bartolini and Javier Gonzalez-Castillo
david_trastour@hp.com, claudio_bartolini@hp.com, javgon@hplb.hpl.hp.com
HP Labs, Filton Road, Bristol BS34 8QZ, UK

Abstract. Matchmaking is an important aspect of e-commerce interactions. Advanced matchmaking services require rich and flexible metadata that are not supported by currently available industry standard frameworks for e-commerce such as UDDI and ebXML. The semantic web initiative at W3C is gaining momentum and generating technologies and tools that might help bridge the gap between the current standard solutions and the requirement for advanced matchmaking services.
In this paper we examine the problem of matchmaking, highlighting the features that a matchmaking service should exhibit and deriving requirements on metadata for description of services from a matchmaking point of view. We then assess a couple of standard frameworks for e-commerce against these requirements. Finally, we report on our experience of developing a semantic web based matchmaking prototype. In particular, we present our views on usefulness, adequacy, maturity and tool support of semantic web related technologies such as RDF and DAML.

Keywords. Semantic Web; E-Commerce; Matchmaking Services; Automated Negotiation; Electronic Marketplaces; Ontology.

1. Introduction

E-commerce is done faster, on a global scale, and with fewer human interventions than traditional trade. Electronic interactions are increasing the efficiency of purchasing, and are allowing increased reach across a global market. With the proliferation of offers comes the problem of finding and selecting potential counterparts for service provision/consumption to engage in negotiation with them.

In the business-to-business (B2B) e-commerce arena, the last couple of years have seen a continuous flourishing of E-marketplaces. E-marketplaces aggregate buyers and sellers in a single virtual location to create dynamic trading exchanges. In doing so, they somehow simplify the problem of discovering potential counterparts for business. Still businesses come together based on the services they require or provide, and matchmaking - i.e. the process of matching service offers with service requests - might be a difficult task depending on the degree of flexibility and expressiveness of the service descriptions.

By analysing the features that we would like an advanced matchmaking service to have, we derived requirements for a language for service descriptions in the context of

matchmaking. These requirements are: high degree of flexibility and expressiveness; ability to express semi-structured data; support for type and subsumption; ability to express constraints over ranges of possible values as well as definite values of a specification.

We have studied industry standard frameworks for e-commerce such as UDDI and ebXML, to see whether the solutions they propose meet our requirements. We found that their main shortcoming is that they do not allow much flexibility and expressiveness in the service descriptions.

After looking at the industry standards, to develop our prototype of an advanced matchmaker, we have taken an approach to service matchmaking based on semantic web technologies. Our approach aims at providing a richer service description, while making use of existing ontologies and the ways of combining and extending them. In this paper, we report about our experience in applying semantic web related technologies to the service description problem. In particular, we investigate the use of RDF as a basis for a service description language[1], and we discuss how well it meets our requirements. In addition, we discuss our experience with some semantic web tools.

The remainder of the paper is structured as follows. In section 2 we describe the features of the matchmaking service; in section 3 we derive the requirements for a language to express descriptions of advertisements and queries to be used in matchmaking; in section 4 we assess current industry standards e-commerce frameworks, such as UDDI and ebXML against the requirements; in section 5 we describe our experience in applying semantic web technologies to the development of a matchmaking service prototype; in section 6 we present related work and in section 7 our future work intentions, to conclude in section 8.

2. Matchmaking

Matchmaking is the process by which parties that are interested in having exchange of economic value are put in contact with potential counterparts.

The matchmaking process is carried out by matching together features that are required by one party and provided by another. In the traditional way of doing business, this process is executed either through brokers, by actively seeking counterparts in directory services such as the yellow pages, or by looking at advertisements on media.

With the possibilities opened by e-commerce, the number of potential counterparts is multiplied. Therefore, who is seeking for a business counterpart is faced with the problem of filtering out relevant from irrelevant information.

2.1. Advertising, querying and browsing

The minimal functionalities that a matchmaking service provides are the features of *advertising* a service, and *browsing* or *querying* a repository of advertised services.

[1] The descriptions that we consider in this work do not involve behavioural aspects of a service, as these are not necessarily required for matchmaking. Therefore, “service description language” here and in the rest of the paper is shorthand for “language to express service parameters”.

2.1.1. Advertising

A party describes the features of the service or product that it is providing or requesting. Such description is published in an *advertisement* in the matchmaking service.

An advertisement defines a space of possible realizations of a service. The level of detail used to describe the service is completely up to the advertiser. It is even possible to advertise more specific and more general descriptions for analogous services at the same time.

The advertiser will add contact details to the advertisement to make it possible for a potential counterpart to follow up. Along with the service features and contact information, corollary information might be expressed on negotiable terms and condition as well as the rules of engagement for the negotiation process. Moreover, the advertiser can also specify visibility rules for the advertisement. The matchmaking services will take them into account when delivering information to interested parties that are browsing or querying the repository.

2.1.2. Querying

To find out a relevant advertisement among the currently available ones, a party can submit a *query*. The query expresses constraints over aspects of advertised services that the submitter is interested in. The query expression will be use to filter out the existing advertisements that are not important to the submitter.

2.1.3. Browsing

The matchmaker offers the possibility of *browsing* the currently available advertisements. The matchmaker maintains an advertisement repository, where posted advertisements are stored. In finding out about advertised services, browsing parties can make use of this information to tune the adverts that they will submit in turn, so as to maximize the likelihood of matching.

To facilitate browsing, the matchmaking services may provide a classification of adverts and of the terms used in them. Many current catalogue-based marketplaces organise products in predefined hierarchical categories, making this classification often too rigid.

2.2. Information in Advertisements and Queries

Functional aspects apart, descriptions of advertisements and queries have much in common. Both usually contain constraint expressions over the structure and the value of the attributes in the service descriptions.

We present an example to give a flavour of what information is contained through advertisements and queries. Let us consider a typical advertisement for the services of sale, shipping and insurance of a given good in a B2B marketplace. The advertisement has to contain parameters to describe aspects of all the services. For the sale itself, it is necessary to have a description of the good touching on its characteristic attributes. For instance, in a B2B marketplace for flowers, prospective sellers sort their offering by

variety, stem length, colour, region of provenience and price. Furthermore, product ratings provided by the growers themselves can be present, along with photos and descriptions for most products.

When services such as payment and shipping are provided together with the sale, descriptions are further complicated. The advertisement may then present aspects such as delivery date and location, or form of payment. Finally, for complex business interactions, a behavioural specification of the collaborative business process that includes definition of roles (e.g. payer, payee, insurer or shipper) and their interactions [14] could be included. An analysis of that is beyond the scope of this paper.

2.3. Use Cases for a Matchmaking Service

In this section, we sketch a short list of very simple use cases to make progress towards the definition of the features of a matchmaking service. In all use cases, the result is that the party requesting the matchmaking service obtain information on published advertisements. The party is then responsible for following up by getting in contact with the publisher of the advertisement.

2.3.1. Use Case 1: Browsing

Party browses the advertisement repository. Party manually finds what it wants by drilling down through the categories.

2.3.2. Use Case 2: Volatile query

Party submits a query to the matchmaker (advertisements repository). The matchmaker immediately returns matching advertisements that are currently present in the repository.

2.3.3. Use Case 3: Persistent query

Party submits a *persistent* query to the matchmaker (advertisements repository). The persistent query is a query that will remain valid for a length of time defined by the party itself. The matchmaker immediately returns matching advertisements that are currently present in the repository. Within the validity period of the query, whenever an advertisement is added or updated that matches the query, the matchmaker will notify the party. The party can decide to remove the persistent query from the matchmaker before the validity period is ended.

2.3.4. Use Case 4: Advertisement

Party posts an advertisement to the matchmaker. This advertisement describes what the party requires or provides and is publicly available to all parties. As with the persistent query, the advertisement is persistent and has a validity period. The matchmaker returns all matching advertisements that are currently present in the repository. Within the validity period of the advertisement, whenever an advertisement is

added or updated that matches the query, the matchmaker will notify the party. The party can decide to remove the advertisement from the matchmaker before the validity period is ended.

2.3.5. Use Case 5: Advertisement with visibility rules

Same as the previous case, except that the party adds visibility rules to the advertisement. These visibility rules define who can see the advertisement based on publicly available attributes of the requestor such as identity or business category.

2.4. How the Matchmaking Service Operates over Advertisements and Queries

As it is apparent from the use cases, the job of the matchmaker is to match together compatible advertisements and return advertisements that satisfy a query. To clarify this point it is worth specifying that:

- Two advertisements are compatible when there exists a realization of a service that has all the characteristics expressed in both service. The matchmaking service will match service requests with compatible service offers.
- An advertisement satisfies a query when there exists a realization of a service that satisfies all the constraints that are expressed in the query.

The job of the matchmaker is therefore to perform operations over the language constructs. In the following section, we start to investigate what are the properties that such a language should possess.

3. Requirements

From the analysis carried out in section 2, we derive a set of requirements for a language to express service descriptions in the context of a matchmaking service.

The first observation that we can make is about the potential complexity of the descriptions. While some aspects of the description can be expressed with simple attribute-value pairs, some others might require more structuring. Levels of specifications can be nested so to form grouping and tree/graph structures. This requires a flexible and expressive metadata model.

Requirement 1: High degree of flexibility and expressiveness

As it happens in traditional business, advertisements are very often under-specified to leave open some aspects of the service to a successive stage of negotiation. Moreover, advertisers should be allowed not to mention some details of the service they provide or require, because they might not have the information, they might not want to disclose it, or simply might not be interested in it.

Requirement 2: Ability to express semi-structured data

When publishing advertisements or submitting queries, it is essential to be able to work at different levels of generality. When querying the repository for services of a certain type, we need to make sure that all the instances of service types that are

subsumed by the requested type are retrieved. As an example, when we require flowers, we expect be matched with anyone providing roses.

Requirement 3: Support for types and subsumption

In querying and advertising, it is usually the case that what is expressed is not a single instance of a service, but rather a conceptual definition of the acceptable instances. A natural way of describing this is by expressing constraints over the parameters of the service.

Requirement 4: Ability to express constraints

As an aside, we note that the descriptions must be understandable by all the participants. This is difficult because the participants can potentially use their own formats to internally represent their products or services. In order for them to interoperate and to provide a powerful subsumption mechanism there is a need for using ontologies. An ontology goes beyond the simple specification of a set of terms; it also expresses relationships between them. There are many ontology efforts, either reference ontologies such as WordNet [8], or domain ontologies, i.e. developed for vertical industries (see TranXML [20] for the transportation as an example). To design an ontology is beyond the scope of our work. We only require descriptions to refer to an ontology in order to mediate between diverse information sources.

In the remainder of this paper, we take into consideration existing language for knowledge representation both from industry standard framework for e-commerce and from the W3C semantic web initiative.

4. Standard Frameworks for E-Commerce

In this section, we assess some industry standard frameworks for e-commerce with respect to the requirements that we identified in the previous section. The standards we considered were UDDI and ebXML.

4.1. Universal Description Discovery and Integration (UDDI)

UDDI is a cross-industry effort driven by a set of major platform and software providers, as well as marketplace operators and e-business leaders. The aim of UDDI is to create a global, platform-independent, open framework to enable businesses to discover each other, define how they interact over the Internet, and share information in a global registry that will more rapidly accelerate the global adoption of B2B e-commerce [4].

When trying to implement a matchmaking service based on UDDI, we incurred in the following problems:

- there is no classification or organisation of UDDI data structures, the tModels (cf. Req. 3);
- tModels only provide a tagging mechanism. UDDI is only intended to provide a first level filter. Further discrimination is done in direct communication with the service provider (cf. Req. 2);

- searching is only done by string equality matching on some fields such as name, location or URL (cf. Req. 3 and 4);
- the description schemata are not extensible (cf. Req. 1 and 2).

4.2. *E-Business eXtensible Markup Language (ebXML)*

ebXML is a set of specifications that together aim to enable a modular electronic business framework. ebXML specifications have XML messaging as a common basis. ebXML is a joint initiative of the United Nations (UN/CEFACT) and OASIS, developed with global participation for global usage.

We briefly considered ebXML as a platform for our matchmaking service. ebXML defines core components like name, address and suchlike information. However, ebXML is very focussed on defining business processes definition and business documents payload. The data model of the Core Component vocabulary does not look very rich and they do not provide support for semi-structured data (cf. Req. 1), inheritance (cf. Req. 3) and constraints (cf. Req. 4).

4.3. Other e-commerce frameworks

In terms of requirements for discovery and matchmaking, none of the other frameworks for e-commerce that we looked at (RosettaNet [17], eCo [6], BizTalk [12]) seemed to provide anything beyond a basic ontology definition.

5. Semantic Web Technologies

As we argued in the previous section, the metadata models used by the main industry standards do not meet the requirements that we stated. Therefore, for the development of a prototype of an advanced matchmaking service, we turned our attention to the semantic web initiative at the W3C consortium.

5.1. Semantic Web

> *The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications.*
> [From the Semantic Web activity statement]

The Semantic Web vision from the semantic web activity statement [18] fits well with the context that we set for our matchmaker. Moreover the efforts currently underway to develop metadata tools and languages promise to offer appropriate responses to the problem that arise in the development of a prototype matchmaker. This becomes evident when comparing the requirements that we collated in section 2 with some of the W3C specifications, namely the Resource Description Framework (RDF) and the Darpa Agent Markup Language (DAML). We have experimented with both RDF and DAML as well

as with some of the related tools currently available or even in course of development, and here we report on our experience with them.

5.2. RDF

RDF is a general-purpose knowledge representation language, and its flexible data model seems to fulfil our needs.

5.2.1. Expression of the service parameters in RDF

The basic RDF data model [10] consists of three object types: *resource*, *property* and *statements*. Resources are the central concept of RDF. They are used to describe anything, from web pages to people. Properties express specific aspects, characteristics, attributes, or relations used to describe a resource. Statements are composed of a specific resource together with a named property and the value of that property for that resource. The value can be a resource in turn. Alternatively, the value can be a *literal*, a primitive term that is not evaluated by an RDF processor. RDF models consist of a bag of statements and are represented as directed labelled graphs, as in the example in Figure 1.

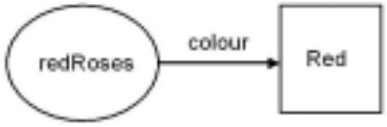


Figure 1: An RDF statement

Since RDF is about neutrally representing knowledge rather than associating specific semantic to a representation, we need to state what interpretation we are going to attach to the representation that we use. The interpretation will depend on the context. For example, let us consider the RDF statement in Figure 1. If it appears in an advertisement for the sale of roses, our matchmaker would interpret it as stating that the colour of the roses for sale *is* red. The same statement in an advertisement for the purchase of roses would be interpreted as stating that the colour of the roses is *required* to be red.

As one would expect, the matchmaker considers the two advertisements as compatible.

5.2.2. Ontology

As we hinted at in section 3, the terms that are expressed in the advertisements have to be defined in an ontology. The design of such an ontology is beyond the scope of this paper. However, we will just underline here that RDF models are typically enriched by an RDF Schema (RDFS) [5]. The RDFS Specification describes how to use RDF itself to define vocabularies of RDF terms. During the development of the matchmaking prototype we made the assumption that there exists some ontology description in RDFS. Our schema borrowed from various ontologies to express concepts such as delivery services or descriptions of flowers. In our example of the B2B flowers marketplace, the ontology describes concepts such as flower type, stem length, region, colour quality, as

well as service related concepts such as delivery date and location. It could also include rules of engagement for the negotiation of other terms and conditions of the service. The automatic – or semi-automatic - merging of ontologies is a difficult problem [13], which we do not take into consideration in this work.

As an aside, it is worth noting that the ontology might import or include information from UDDI yellow or white pages or ebXML registries. In this way, we get the best of the two worlds: extended reach thanks to the industry standard frameworks and expressiveness from the semantic web.

5.2.3. Advertisements in RDF

We describe an advertisement as an RDF graph that defines a space of possible realizations of one or more services, not by expressing a particular realization of the service(s). Therefore, some of the aspects of the advertisement need to be expressed through constraints (cf. Req 4 in the previous section).

Figure 2 represents an example of an advertisement. Together with the advertisement, terms are represented that belong to the ontologies and would be shared by the different parties involved in a matchmaking session. For the sake of our example, we have modelled the services of sale and delivery. The service of sale defines a set of items, the total price of the sale and whether the intention of the advertiser is to buy or to sell. The product can specify a quantity. In turn, the quantity is expressed by a measure that can be either volume or weight. The delivery service defines charge, delivery date, origin and destination locations.

Each advertisement is represented as an RDF resource of type `Advertisement` and as a result has its own URI. It designates the root node of the description. Properties from this resource will characterize the types of services that are required or provided. By using our example ontologies, it is possible to add further details and form a full RDF advertisement. In Figure 2[2], the RDF sub-graph representing the advertisement is highlighted in colour. The advertisement is for the wholesale purchase and delivery of 100 kg roses. The root of the advertisement is the `myAdvert` resource. Navigating the graph from `myAdvert`, all the relevant information is reachable. For instance, the desired quantity and colour of the item for sale can be read as properties of the `redRoses` node.

As we argued above, advertisements express constraints over the possible realizations of a service. RDF is useful in dealing with two kinds of constraints. Typing constraints are used to say that a node must be of a certain type, or any subtype or supertype of it. We express them by using the `rdf:type` relation. In our example, the node `redRoses` expresses that a potential matching advert needs to have a node of type `Rose`, or a subclass of it. Equality constraints on values are obtained by specifying a value for a literal. In our example, the buyer is only interested in buying exactly 100 kg of flowers.

[2] Obtained with the Protégé ontology editor [19].

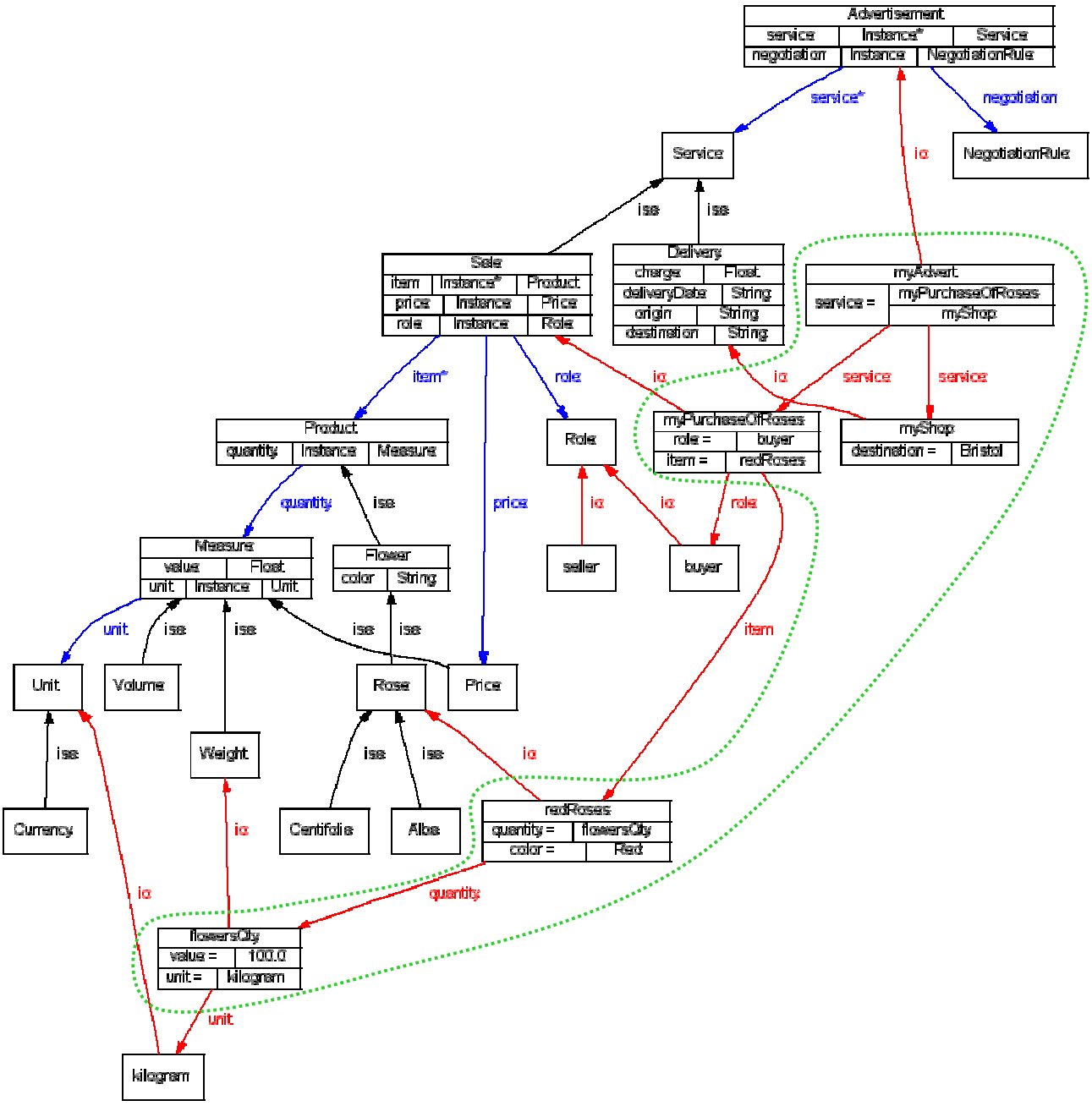


Figure 2: An advertisement enriched with ontologies

5.2.4. Matching

With the design decisions that we made, matching of advertisements is reduced to matching of RDF graphs. We have implemented a matching algorithm, based on the visitor pattern [7]. The algorithm is implemented in Java and based on the Jena RDF API [11]. Advertisements match when their root nodes (that must be instances of `Advertisement`) match, and all their respective sub-nodes do too.

Following the visitor pattern, we have defined a default matching rules and a mechanism for it to be overridden for nodes of particular types. The association of

overriding matching rule with a node type is done by annotating the RDFS schema. Precisely, a `matchingRule` property is added to the node representing a Class resource. The value of the property is a literal that expresses the fully qualified name of the java class that implements the matching rules for the resources of that type.

The default matching rule is presented in Figure 3. The main idea is to traverse simultaneously two advertisements by finding recursively the nodes that share a common type and making sure there are no incompatibilities between the advertisements. In the following algorithm description, we ignore the possibility of cycles, for simplicity of explanation.

```
Two Advertisements match when:
   Their root node match.

Two nodes match when:
   One of them is a subtype of the other.
   AND
      IF
        A matching rule is defined for the most specialised common type
        between the two nodes, the matching rule is evaluated positive.
      ELSE (default matching rule)
        FOR each property p1 that appears in one node such that there
        exists p2 in the other node where p1 = p2 or p2 is a sub-
        property of p1,
           The two object nodes from p1 and p2 match.
```

Figure 3: Matching algorithm for two RDF graphs

Let us go back to our flower advertisement (presented without the related ontologies in figure 4) and examine it in light of the matching algorithm. Our buyer is interested in having her advertisement matched against compatible advertisements from sellers. In the rest of this sub-section, we show some advertisement that would match our buyer's and some that would not. Our buyer advertised for the purchase of 100 kg of red roses.

Among the published advertisements in the matchmaking repository we consider the following:

Table 1: Matching of Advertisements

Sale and delivery of	*Result*	*Justification*
flowers	Hit	`Rose` is a subclass of `Flower`
roses	Hit	No problem
100 kg of blue roses	Miss	Mismatch on the colour property
daffodils	Miss	`Daffodil` is a subclass of `Flower` but is disjoint with `Rose`: mismatch on the types
100 kg of long stem centifolia roses	Hit	`Centifolia` is a subclass of `Rose`
100 kg of short stem alba roses	Hit	`Alba` is a subclass of `Rose`[3]
Up to 300 kg of roses	Hit	Compatible constraint. See discussion

[3] Actually it should be a miss because Alba is variety of white roses. It is a limitation of the design of our ontology. See the discussion subsection.

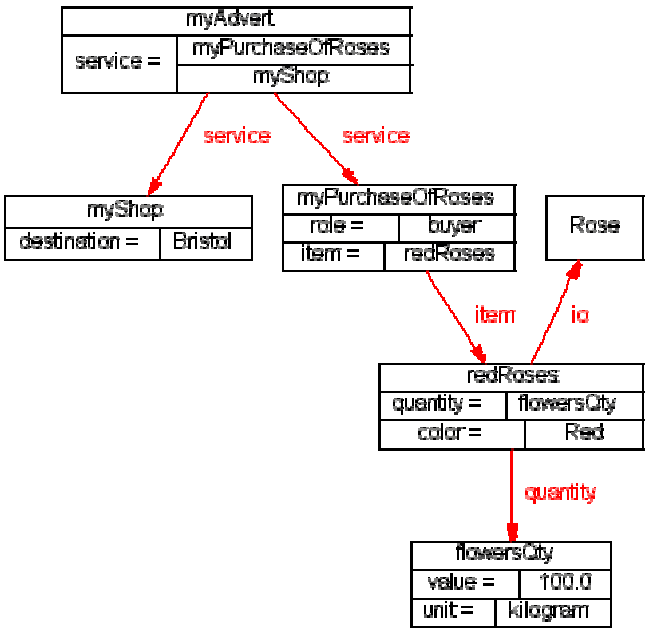


Figure 4: Buyer's advertisement

The first six examples are easily expressed by the syntax illustrated so far. However, for the last advertisement, we need to devise some mechanism to express and resolve the constraint expression that appears in the seller's advertisement.

In our first prototype of a matchmaker, we have designed a proprietary syntax to express the constraints directly in the value slot of the instance. In our example, to express an advertisement for the sale of a quantity of roses up to 300 kg, we set the literal corresponding to the `rdf:value` to be the literal `"LessOrEqualThan 300"`. The matching algorithm parses the value string and interprets the constraint. The drawback with this approach is that is based on sole syntax. To overcome this problem, we envisage to design a constraint ontology that will allow us to annotate nodes in the `Advertisement` graph as representing an instance of a particular kind of constraint (such as `LessOrEqualThan 300` in the example). We expect this to have a minimal impact on the design and implementation of the matching algorithm.

The discussion so far has been focussed on advertisements only. As we observed in section 2, advertisement and queries have much in common. Our bias is that they can and should use the same filtering mechanism and possibly be expressed through the same language constructs. The constraint system we have just introduced provides this filtering mechanism. The last example could be seen as both an advertisement and a query.

5.3. Discussion

At the end of section 3, we listed the following requirements for a description language to be used in matchmaking services:

- high degree of flexibility and expressiveness;
- ability to express semi-structured data;
- support for types and subsumption;
- support for constraints.

Based on our experience, RDF is offers valuable support to meeting the first three. On the other hand, we find that it falls short on our requirement 4: support for constraints. To overcome this problem we had to design a proprietary mechanism. One of the problems we had to express advertisements in RDF is the one raised in the Alba example in the previous sub-section. Alba is a variety of roses that comes in white colour only. If a seller advertises the sale of Alba roses, his advertisement should not be matched with the one proposing to purchase red roses. Our matching algorithm would mistakenly match the two advertisements because the ontology does not express the restriction of the colour property of Alba class to be white. We found this difficult to express in RDFS, as it does not seem possible to restrict the range of a property for a subset of its original domain

5.3.1. DAML

More recently, we started to look at DAML+OIL [9] to enrich our RDF descriptions. DAML+OIL looks promising to overcome both of the hurdles that we encountered with RDF. We have started to experiment and express the concepts of a service description as DAML+OIL classes. The service description is defined as the boolean combination – intersection, disjunction or complement – of a set of restrictions over datatypes and abstract properties. These restrictions are expressed either through DAML+OIL restrictions or through XML Schema restrictions.

On the constraint support side, DAML+OIL allows us to define concepts using restrictions, for instance existential qualifiers, universal qualifiers or cardinality over properties. Rich datatype definitions can also be used in these restrictions and are defined in XML Schema, leaving us in particular the possibility to express ranges.

Because DAML+OIL classes can be restricted on the target value/class of a property, it is possible to create richer ontologies. Our example ontology for flower could be more complete by adding the fact that Alba is a sort of Rose whose colour is always white.

5.3.2. RDF Tools

RDF tools that provide an interesting set of features start to become available. There are many APIs allowing to manipulate RDF models, either providing a low-level triple-store abstraction or a providing a higher graph abstraction. We chose the second type of implementation, and more particularly Jena, as we felt it would be more suited to our problem.

Protégé [19] is a very good tool to design ontologies in RDF, even though it does not support all the features of RDF (the absence of multi-class membership is an example). Protégé also provides some interesting features not present in RDF that look very promising for some future DAML support.

6. Related work

RDFSuite [1] and KRAFT [15] are highly relevant to our present work. For a discussion on how they relate, see the section on future work.

Reynolds [16] presented an RDF framework for resource discovery. In the context of his framework, the directed graph query language (DGQL) is a simple query language for RDF based on graph matching. However, now DGQL can only perform equality and indifference tests. What we propose to do is based on a wider array of possibilities for constraint matching.

7. Future work

As we highlighted in the discussion in section 5.3, an important part of follow up work for our matchmaker will be the design of a constraint ontology and the implementation of a mechanism to integrate constraint solving with the current matching algorithm. We envisage that RDFPath - or approaches of its kind - will be useful to apply the visitor pattern to the task of matching sub-graphs.

We also would like to extend this work and enlarge its scope from matchmaking to other phases of the e-commerce process. In particular, because constraints expressed in matchmaking advertisements may be similarly useful in expressing negotiation proposals [2], we intend to extend our matching algorithm and metadata model to cope with automated negotiation. In this framework, negotiation proposals are expressed as service descriptions the same way advertisements are.

Matchmaking does not necessarily require expressing behavioural aspects of a service. However, in the B2B environment, collaborative business processes are a fundamental concept. Therefore, we plan to extend our work to include aspects of service behaviour.

We follow with interest the work on RDFSuite and RQL in particular [1]. RDFSuite provides a suite of tools for RDF storage and querying. They manage to achieve good results in scalability because of their use of database technology. When we will tackle the problem of persistence of the advertisements repository, we plan to investigate the use of RDFSuite.

KRAFT [15] is an architecture for supporting virtual organization that uses constraints as a knowledge exchange medium. KRAFT is highly relevant to our work, especially as we move onto defining an ontology of constraints.

8. Conclusion

Our experience in prototyping an advanced matchmaking service made us to realize that there is a gap between what standard frameworks for e-commerce provide today and what could be achieved through the usage of semantic web technologies. We believe that in the near future automated matchmaking and negotiation will achieve results at a level of complexity far beyond what is possible today. And semantic web tools and technologies will play a primary role in making that happen.

9. References

[1] Alexaki, V; et al. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001), May 2001.

[2] Bartolini, C; Priest, C. A Framework for Automated Negotiation. HP Labs, Technical Report. 2001.

[3] Bechhofer, S; Gobbe, C. Delivering Terminological Services. University of Manchester.

[4] Boubez, T; et al. UDDI Data Structure Reference V1.0, UDDI Open Draft Specification, Sep 2000.

[5] Brickley, D; Guha, R.V. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, Mar 2000; available at http://www.w3.org/TR/rdf-schema/.

[6] CommerceNet, Inc. eCo Architecture for Electronic Commerce Interoperability, 1999.

[7] Gamma, E; Helm, R; Johnson, R; Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading MA: Addison-Wesley, 1995.

[8] Fellbaum, C. WordNet: An electronic lexical database. The MIT Press. 1998.

[9] Hendler, J; McGuinness, D.L. The DARPA Agent Markup Language, *IEEE Intelligent System*s, vol. 16, no. 6, Jan./Feb., 2000, pp. 67–73.

[10] Lassila, O; Swick, R. 1999. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, Feb 1999; available at http://www.w3.org/TR/REC-rdf-syntax/.

[11] Mc Bride, B. Jena: Implementating the RDF Model and Syntax Specification, Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001), May 2001.

[12] Microsoft. BizTalk Framework 2.0: Document and Message Specification, MSDN Online, Jun 2000.

[13] Omelayenko B., Syntactic-Level Ontology Integration Rules for E-commerce, Proceedings of The 14th International FLAIRS Conference (FLAIRS-2001), May 2001.

[14] Piccinelli, G., Mokrushin L. Dynamic Service Aggregation in Electronic Marketplaces. Special Issue on Electronic Business Systems of the Computer Networks journal. HP Labs, Technical Report. 2001.

[15] Preece, A; et al. KRAFT: Supporting Virtual Organisations thought Knowledge Fusion, *AAAI-99* Workshop on Artificial Intelligence for Electronic Commerce, 1999.

[16] Reynolds, F. An RDF Framework for Resource Discovery, Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001), May 2001.

[17] RosettaNet Organization. http://www.rosettanet.org, 2001

[18] Semantic Web Activity. http://www.w3.org/2001/sw/, 2001.

[19] Stanford Medical Informatics. The Protégé Project. http://protege.standford.edu, 2001

[20] Transentric. TranXML: the Common Vocabulary for Transportation Data Exchange. 2001.

The Briefing Associate: A Role for COTS applications in the Semantic Web

Marcelo TALLIS *mtallis@teknowledge.com*
Neil M. GOLDMAN *ngoldman@teknowledge.com*
Robert M. BALZER *bbalzer@teknowledge.com*

Teknowledge Corporation, 4640 Admiralty Way, Suite 231, Marina del Rey, CA, U.S.A.

Abstract. This paper identifies a set of semantic markup capabilities designed to benefit the author rather than the consumer of manually composed documents. By doing so it addresses one of the major challenges facing the semantic web vision – the generation of ontologically encoded descriptions of the content of manually produced documents. It also presents a novel approach to eliminating the currently large and tedious overhead required to produce such markup by augmenting the COTS tools that users already use to produce these documents so that the semantic markup is derived as a byproduct of composing the document. These ideas are currently being implemented in a tool called the Briefing Associate that augments Microsoft's PowerPoint to support the authoring of semantically grounded briefings.

1. Introduction

The semantic web promises to expand the services of the existing web by enabling software agents to automate procedures currently performed manually and by introducing new applications that are infeasible today. The enabling factor to materialize this vision is the availability of web documents containing ontologically encoded information that software agents and tools can accurately and reliably interpret. A major challenge facing the semantic web vision is the generation of this encoding, especially for encoding or summarizing the content of documents composed by people. The mark-up of such documents is currently a tedious and sometimes complex activity. Because the benefits of these markups accrue most immediately to the agent-assisted *consumers* of the web content, content *producers* are not highly motivated to undertake this extra effort.

Although there has been considerable technical progress in supporting other portions of the semantic web lifecycle, there has been little progress in the markup of manually composed documents. The prevalent approach is to create specialized tools that specifically support the association of semantic markups with the content of existing documents [1],[2]. These tools provide a GUI that permits an author to browse ontologies, find appropriate terms, generate syntactically correct markups,

and associate them with (portions of) the document's content. This activity remains an "extra" effort that does not directly reward to the person performing it.

We are experimenting with a different approach. Rather than add ontological encoding to completed documents, we propose to *augment* the COTS tools that users *already* use to produce these documents to produce the ontological encoding as a *byproduct* of document composition. The intent of such augmentation is to (nearly) eliminate any cost of producing ontologically encoded documents beyond the costs inherent in producing the equivalent semantics-free version. We are also exploring the incorporation of analysis and synthesis tools that utilize these semantic markups during document composition to improve the resulting document's accuracy, quality, and/or speed of production. Authors themselves will thus reap a direct benefit from creating documents with associated ontological encoding. Integrating this functionality into the COTS tools that authors already use, without restricting their use of the tools' existing functionality, obviously simplifies the transition to this paradigm.

This paper describes the Briefing Associate, an application of this approach tailored to the creation of briefings using Microsoft PowerPoint. The Briefing Associate augments PowerPoint's native GUI with graphics that represent concepts and relations imported from an ontology. The concepts and relations from the ontology also define a set of attributes authors can fill in through popup dialog boxes. The author builds a briefing in PowerPoint using a combination of these ontology-related shapes and connectors and native PowerPoint elements. As a byproduct of building the briefing the author is also describing the relationships among concept instances. Each ontology-related graphic represents an instance of a concept. The ontology-related connectors between graphics stand for relations between the associated instances. The ontological encoding created as a result of using the augmented GUI is stored persistently within the PowerPoint document.

As a briefing is being composed, this evolving semantic model of instances, relations, and attribute values is shared (through Microsoft COM interfaces) with external tools called *analyzers.* These tools can process the semantic model to determine whether it is consistent and complete, perform an analysis to determine derived properties, or augment it with additional information. The Briefing Associate provides a mechanism for these external *analyzers* to add visual annotations (e.g., highlights) to the graphics that stand for model elements to provide feedback to the author. It also provides a synthesis mechanism to these tools for augmenting the author's semantic markups.

These analyzers can be general or domain specific and can be individually activated or deactivated by the author through the Briefing Associate's enhanced GUI. One particular generic analyzer is a *publisher* that exports the semantic content (i.e. the markup) of the briefing in the DAML+OIL language [3]. Using a generic *briefing* ontology, it also exports the briefing meta-information (author, date, size, etc) and all *titles* and *text* appearing in the briefing.

We are also implementing a generic analyzer that imports the ontological encoding resulting from a semantic web query into a briefing and renders it graphically. It will persistently associate the query with the resulting model elements, allowing the query to be reused in the future to keep the briefing up to date.

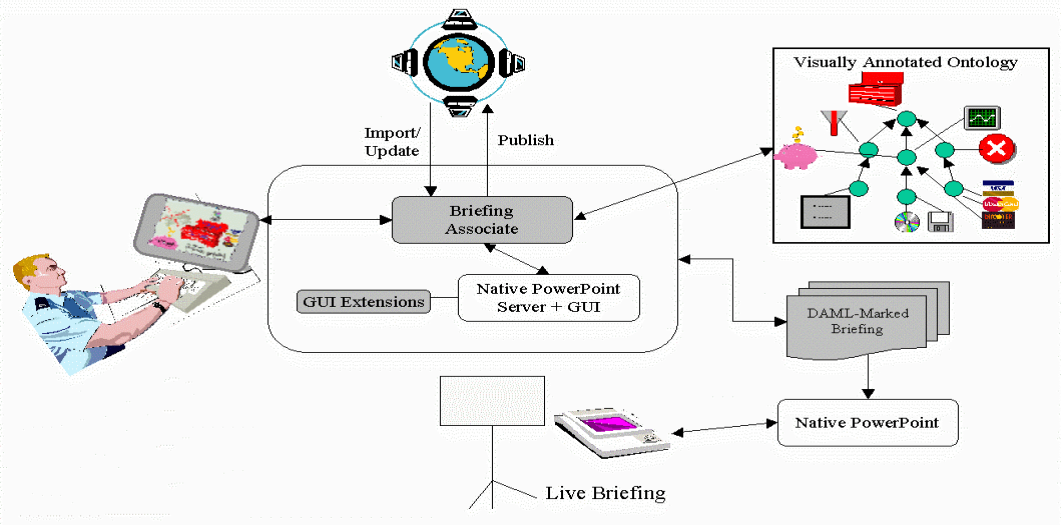


Figure 1 Briefing Associate Software Architecture

In the following sections we describe the Briefing Associate and how the ontologically encoded descriptions are introduced into a briefing, how the Briefing Associate interacts with external analyzers, and the current implementation of these ideas.

2. Briefing Associate

The *Briefing Associate* (BA) facilitates the composition and publication of semantically grounded briefings. The briefings contain markups that describe the domain-specific content matter of the briefing and are linked at a fine granularity to units of visual content in the briefing. A briefing may contain both *original* and *imported* semantic content. The BA generates DAML descriptions of a briefing's original content as a *byproduct* of creating that content's visual depiction. The creation of DAML markup for original content is mediated by *visually annotated* DAML ontologies (VAOs) from which authors select ontologically defined objects as predefined graphic shapes or icons to include in their briefing. These visually annotated ontologies are demand-loaded into the BA to specialize it to a particular subject-matter domain. They also permit the BA to generate graphical depictions of imported semantic content. Content imported from agents will be marked with the source agent and query used to obtain the content, permitting the BA to obtain, on request, an updated version of that content from the same agent.

The BA is implemented as an extension of Microsoft PowerPoint. Briefing authors familiar with that product can continue to rely on the native user interface tools, menus, and direct-manipulation actions to edit visual content. Extended interpretation of these tools and actions, and additional tools created from the ontology annotations, simplify the creation of new content, while simultaneously creating DAML markup. Figure 1 depicts the BA's architecture and major information flows.

The Briefing Associate augments PowerPoint with graphics-bearing ontological categories. These graphics represent instances of domain concepts, attributes

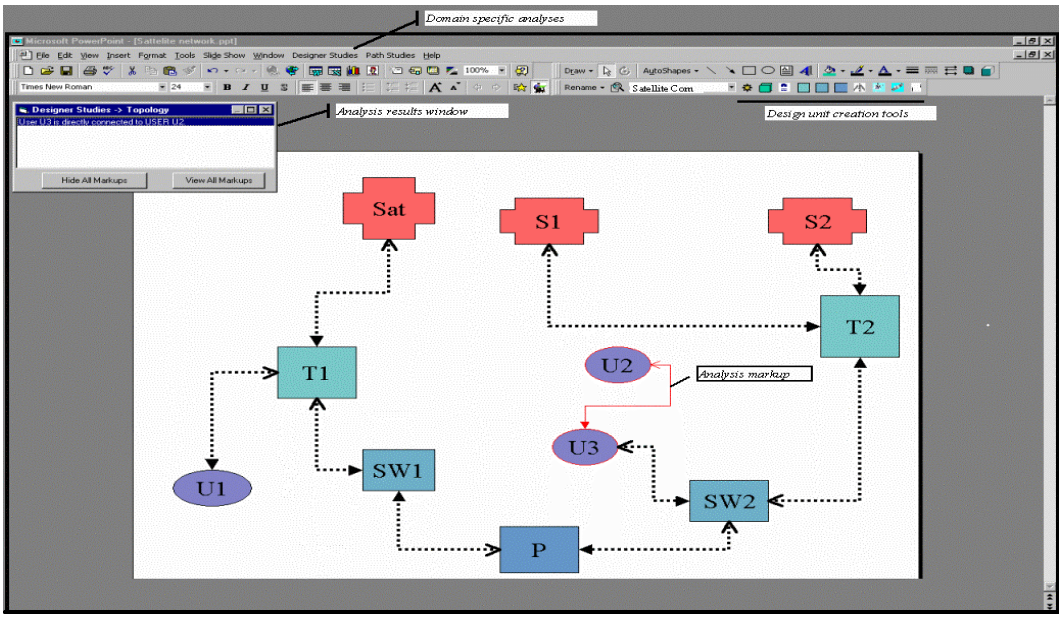


Figure 2 Ontology aware briefing editor GUI – satellite communications

(primitive data typed properties), and their relationships. The author, while composing a briefing using these graphics is indirectly constructing a semantic description of the briefing content. Besides supporting the construction of semantically grounded briefings, the Briefing Associate also exposes the briefing's emerging semantic descriptions to external modules called *analyzers* that perform specialized services or *analyzes for the author*. These analyses can provide feedback to the author, can extend or modify the briefing, or can produce external documents. One particular generic analyzer is a *publisher* that generates the semantic markups that describe the briefing content. The Briefing Associate extends the PowerPoint GUI with tools, menus, and gestures for instantiating the semantically annotated graphics, assigning attribute values to the instances and relations represented by these graphics, invoking analyses, importing and updating the graphic representations of imported semantic descriptions, and annotating domain ontology concepts and relations with their visual representations. The following subsections describe the components that achieve these added services.

3. Ontology-Aware Briefing Editor

The Ontology-Aware Briefing Editor allows a briefing author to create original content and to import and edit externally produced content. Visually annotated ontologies, discussed below, provide the means to relate DAML descriptions from a given ontology to a visual model.

Ontology-aware editing takes place through a combination of standard PowerPoint interface actions, additional GUI elements added by the BA, and extended interpretation of native controls and direct-manipulation actions. The entire native PowerPoint user interface continues to be functional. User-preference tailoring of that interface is preserved.

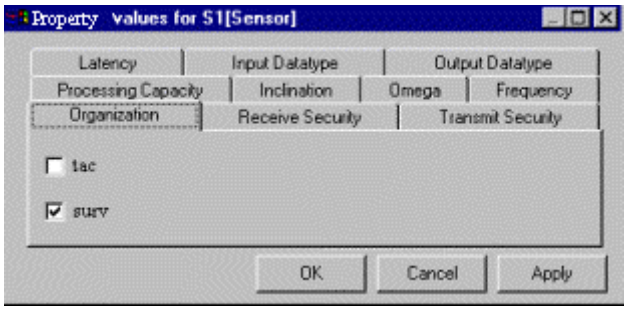


Figure 3 Property value dialog

A visually annotated ontology is the key to creating original content as well as to automatically depicting imported content. A new toolbar is added to the PowerPoint GUI for the ontology. For each concept and relation in the ontology, an *instantiation tool* is added to the toolbar. Our current implementation lays these tools out in a single list. We also plan to offer these tools in a cascading interface, mirroring the class inheritance of the ontology. Clicking on one of these tools, like PowerPoint's native autoshape tools, allows the author to insert a copy of the graphic template anywhere in his briefing. Domain relations defined in the Ontology are graphically depicted by arrows (more precisely PowerPoint connectors) whose ends are attached to the concept instances related by that relation. These instantiation tools simultaneously create the internal semantic representation for that concept or relation instance as defined by the ontology (including any default attribute values).

The Ontology-aware briefing editor also allows the author to edit these domain attribute values through a dialog box interface that is activated from the context menu of the graphic representing that instance in the briefing. A *tabbed dialog* is created for the selected instance with a tab for each attribute applicable to that instance. Each tab provides an interface, specific to the attribute type, for viewing and setting the value of that attribute.

We plan to augment this textual interface with a graphical one that enables some of an object's attributes to be modified by direct manipulation of the object's graphics (e.g. changing the size of an object might modify some aggregate value such as the length of a queue, and changing its color from a list of alternatives might modify some enumerated type such as its state). The correspondence of these direct manipulations to the attribute affected will be defined by additional visual annotations of the ontology.

Figure 2 is a screen shot of the ontology-aware briefing editor in a "satellite communications" domain. Everything in the figure is part of the GUI with the exception of the callouts highlighting specific elements.

In the central canvas is the depiction of a "satellite communications" configuration. The various labeled shapes represent instances of satellites, terminals, switches, processors, and users – the domain concepts defined in the ontology. They are connected by arrows representing communication links – the only (non data-typed) domain relation in this ontology.

The author created the preponderance of this briefing through the instantiation tools on the domain toolbar, on the right side of the second row of docked toolbars at the top of the figure. To the immediate left of these instantiation tools in the

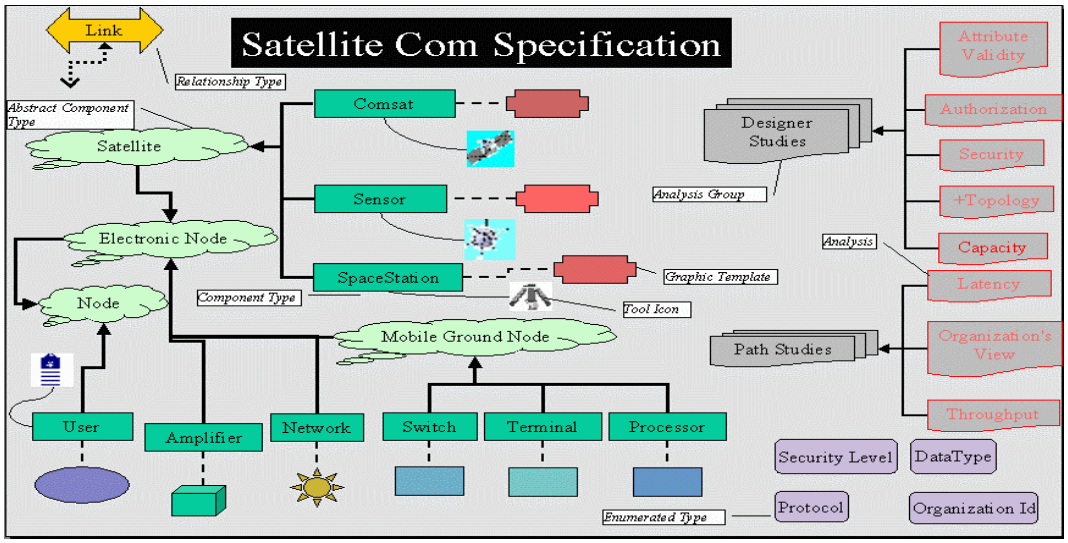


Figure 4 Visually annotated ontology – satellite communication

domain toolbar is a drop-down list box displaying the name of the current ontology ("Satellite Com"). When a briefing author starts a new briefing, this box allows him to choose an ontology. This triggers the creation and display of the appropriate domain toolbar for that ontology. Manipulation of the concept and property instances on the briefing – positioning, resizing, selecting, attaching/detaching links– is carried out through PowerPoint's native mouse gestures and/or keyboard shortcuts.

In Figure 2 the user has requested a "topology" analysis, one of the analyses in the "Designer Studies" group. The results of this analysis are displayed in a separate window, visible at the upper left of the canvas in Figure 2. The window displays a list of reports. In this example, there was just one report. Its explanation reads "User U3 is directly connected to user U2." When the user selects one of these reports, its associated *markups* are displayed as highlights. In this case, the only markup called for highlighting the communication link between U2 and U3. That is why that link has an appearance (a thin red arrow) different from the others. The effect of this highlighting is reversed when the report is deselected or the analysis window is closed.

Attribute values are viewed and assigned through dialogs, displayed on demand from the graphic instance's context menus. Figure 3 exhibits the dialog for a sensor satellite. The dialog contains a "tab" for each attribute associated with that concept in the ontology. The details of a tab depend on the value type of the attribute and on the concept specification.

Identical dialogs are used to gather the parameter values for parameterized analyses.

4. Visual-Annotation Ontology Editor

The Briefing Associate is not limited to any particular ontology--any DAML ontology can be annotated. Multiple annotated versions of a single ontology may be created, so that briefings can be tailored easily to different audiences with different

conventions for the visual representation of information. However, any single briefing will be based on a single visually annotated ontology.

The Visual-Annotation Ontology Editor provides an interactive means to establish a mapping between the concepts of an ontology and their visual representation. When an ontology O is imported into the VAO editor, the editor lays out O's concepts and properties depicting their hierarchical relationships (i.e., the subclass and subproperty properties). The user can assign graphic representations to these concepts and properties and assign icons to be used in the ontology tool bar used for briefings to be associated with O. The user also indicates the analyses that briefing authors will be able to invoke from the ontology-aware briefing editor through its GUI.

The VAO editor is actually a specialization of the ontology-aware briefing editor that uses the visual annotations defined for the *ontology* domain. These visual annotations allow the object and relation types in an ontology to be defined graphically. These ontology annotations are thus just briefings in this *ontology* domain and are saved as a PowerPoint presentation. They are loaded on demand by the Semantic Content Import and Update and Ontology-Aware Briefing Editor components. Importing an ontology is thus a case of content import, while adding visual annotations is a case of original content creation.

Figure 4 shows the Satellite Communications visually annotated ontology used in the example of Figure 2. The (green) rectangles labeled "Comsat", "Sensor", "User", etc. represent the leaf domain concepts. The cross shapes attached to them by dashed connections are their *graphic templates*. This determines the appearance of an instance of that concept. Any of PowerPoint's native autoshapes, formatted as desired, may be used as a graphic template. Alternatively, an image may be chosen as a graphic template.

A concept may be connected (via a curved solid connector) to an image that serves as the *tool icon* for that concept in the instantiation toolbar. Tool icons, like graphic templates, may be selected from a shape library or may be imported. If no tool icon is specified, a scaled version of the graphic template is used as the tool icon.

The (light green) clouds labeled "Satellite", etc. represent the non-leaf concepts. The (gold) arrow shape labeled "Link" defines the sole relationship type in this domain. The dashed, double-headed arrow attached to it is the graphic template for the "Link" relationship type. The user tailors the color, dashing and arrowhead styles of a relationship template in the graphic domain specification just as he tailors component type templates.

Any concept or relationship type may have initial attribute values specified through a property-editing dialog, identical to the ones used by the ontology-aware briefing editor. The default values are assigned when new instances of the type are created.

Figure 4 contains the specification of two analysis groups, "Designer Studies" and "Path Studies", and eight analyses in those analysis groups. The color and styling of the border of an analysis specify the means used to highlight components and relationships identified in *reports* in the feedback from the corresponding analyzers. For instance, the "U2-U3" connection in Figure 2 was highlighted as a thin red line because the border of the "Topology" analysis is a thin red line. Analogously, the text characteristics – font, face, size, color – of the label of an

analysis specify the textual characteristics of any markup text found in feedback from the analysis.

5. Semantic Content Import and Update

At the time this paper was written no generic semantic content import and update component suitable for any arbitrary ontology has been implemented. Instead a series of ontology-specific semantic content import and update components tailored to a particular domain have been created. The following is a description of the generic component we plan to implement.

The Semantic Content Import and Update component will allow the author of a PowerPoint presentation to contact DAML-aware agents, including search agents, and post queries to those agents. It will accept, as DAML descriptions, the results of those queries. The queries as well as the imported descriptions and meta-information will be incorporated as a persistent part of the presentation.

The component will determine how these objects should appear (i.e. be rendered) within the briefing, as specified in the respective ontology annotations to represent the imported DAML content. To do so, it will size, color, label, and place graphic renditions of these objects and interconnect them with one another. These graphic renditions will become a persistent part of the presentation, associated with the specific description units that they depict. The author will generally need to adjust the sizes and positions of these graphics to produce acceptable layouts.

The component will also provide information update capabilities, allowing information updates on demand through a menu item added to the PowerPoint user interface. Using the retained queries, the component will re-query the source agent(s) to retrieve updated content and generate an updated version. At the author's discretion, the component will visually correlate the two versions. The author may choose to incorporate the updated version as a whole, or to selectively incorporate changed information.

The author will also be able to request that any manual customization of the graphic rendition of imported content be reapplied to updates of that imported content so that it doesn't have to be reapplied manually.

6. Briefing Associate - analyzer interaction

Analyzers are external executable modules that process the internal semantic descriptions of the briefing content to provide an analysis, a synthesis, or some other service. An analyzer can be implemented so as to execute within the PowerPoint process, as a separate process on the same machine, or (via DCOM) on a different workstation. Analyses are associated with a particular domain and this association is indicated in the VAO and they are invoked through the BA editor menu for that domain. When an author requests an analysis, the BA creates a connection to the module implementing that analysis and passes it a reference to the briefing to be analyzed, together with any author-provided parameters for the analysis. That analyzer is subsequently expected to send the BA a set of reports

describing the analysis performed, the synthesis done, or the service rendered. The BA then presents the report(s) to the author.

For a *snapshot* analysis, the analyzer's responsibility ends with transmission of the reports detailing that analysis. An *incremental* analysis, however, is expected to send updates to its reports as the author continues to modify the briefing, until the author closes either the analysis or the briefing. To support incremental analyses, the briefing reference handed to the analyzer by the BA provides not only direct access to the *content* of the briefing, but to *events* representing changes to that content.

A *transaction* grouping is imposed on top of events. It is these transactions, not primitive events, that represent the unit of change to which an incremental analyzer commits to respond with updated analysis reports. Because the responses are permitted to be asynchronous, they are accompanied by the transaction id of the transaction that triggered them. This allows the BA to understand, and reflect in its GUI, whether a displayed set of analysis reports is *up-to-date*.

Although the briefing reference provided to an analyzer can be used to gain direct access to PowerPoint's detailed *graphic* model of a briefing, analyzers are typically interested in the *ontology-based* model information that is being automatically generated when content is imported from the semantic web or created through tools associated with the VAO. For each ontology, a COM *type library* is automatically generated. This type library reflects a straightforward mapping between classes and properties of the ontologies and the corresponding modeling concepts (classes, interfaces, and properties) of COM. Most, if not all, widely-used programming language IDEs for the Windows platform provide a declarative way to *import* such a type library, automatically building the client-side code needed to program directly in terms of the objects exposed by the library.

7. Implementation

The Briefing Associate is a descendent of the Design Editor [4], an application for producing visual domain-specific design environments. The Briefing Associate, like the Design Editor, is implemented as an extension of Microsoft PowerPoint. We regard this choice not as an implementation detail, but as central to this research. First, PowerPoint provides us as implementers with a far higher-level platform for building a briefing tool than generic middleware, such as COM/CORBA and GUI widget libraries. It provides an extensive ontology for representing the visual content of briefings, and support for making models that use that ontology persistent. Furthermore, it provides an extensive WYSIWYG user interface for viewing and editing the visual content of a briefing. This interface requires some extension, but no redesign or reimplementation, to accommodate DAML-aware briefings. Second, PowerPoint is the most widely used product for authoring briefings and hence it facilitates the adoption of the BA by briefing authors.

The BA is programmed primarily in Visual Basic. For PowerPoint 2000, this extension is a COM addin that receives "events" as the user creates, opens, closed, and modifies briefings. As a client of PowerPoint, this module can navigate through a briefing and paint analysis feedback directly onto it. For efficiency reasons, this

module runs entirely as an "in-process" component. This means it is incorporated into the PowerPoint process itself. Method calls are extremely efficient when both client and server are part of a single operating system process. Greater efficiency could be achieved by implementing the BA in C++, but the performance of the Visual Basic code has been acceptable to date.

PowerPoint's native extension mechanisms include a general, albeit low-level, ability to add arbitrary non-graphic information to a presentation and retain that information in the presentation's persistent file format. The BA implementation relies on this mechanism to retain all ontology-related information about a presentation across editing sessions – it does *not* attempt to infer ontological information on the basis of graphic attributes of existing graphic objects.

8. Rewarding the Briefing Associate Adopter

The Briefing Associate's authoring environment is an enhanced PowerPoint, the same environment most briefing authors already use. The enhancements do not *remove* capabilities, do not *necessitate* the use of new means for accomplishing old goals, do not *alter* the visual appearance of the ultimate product, and *do not* impose perceivable delays in processing speed. Thus, the Briefing Associate doesn't impose any extra impediments or costs on producing briefings with the standard tools in the standard way.

But we need to motivate the briefing author to use the Briefing Associate's markup tools. The biggest benefits of such markup will obviously accrue to the consumers of these briefings who will be able to quickly and accurately find specific content in those briefings because they have been semantically annotated. Realistically, we should never expect people to incur significant costs, whether in time, retraining, or reduced product quality, on the basis that some benefit *might* accrue in the future, especially when that potential benefit accrues to others.

We have therefore added several enhancements to the Briefing Associate that provide immediate benefit to the briefing author to motivate the use of the Briefing Associate's semantic markup capabilities:

- The Briefing Associate simplifies the construction of the briefings because authors will have readily available the graphic templates that they repeatedly use to represent objects of the domain.
- Ontology-based descriptions of a briefing's content are generated as a side-effect of briefing composition. The extent and value of such descriptions, however, depends on the extent to which the author makes use of the extensions offered by the BA.
- Generic and domain specific *analyzer* tools exploit the semantics of the briefing content to provide an analysis, synthesis, or other service for the author while the briefing is being created. Although such analyzers are not inherently tied to the semantic web, their implementations might well make use of web-based agents that *consume* the content of a briefing and provide feedback to the author.
- Ontology-based descriptions of briefing meta-data and textual content are produced at no cost and independent of the use of any extensions.

- The BA's extensions for importing and visualizing semantically marked-up content could be a significant time-saver in constructing certain classes of briefing. Since these facilities are designed to rely on queries *posted to* the semantic web, however, they server to *leverage*, rather than to *bootstrap*, the semantic web vision.
- The BA will automate the update of content that originated in the semantic web. Like import and visualization, this is a leveraging rather than a bootstrapping relationship of the BA to the semantic web.

These BA author-enhancements just embed the semantic web lifecycle into the briefing creation process so that briefing authors themselves can enjoy (some of) the benefits of semantic markup.

Ontology-based annotations will turn briefings into reusable resources. New content as well as novel aggregations of imported content will be published in a form accessible to DAML-enabled agents. Linking the graphic content to the semantic content in the published briefing will foster reuse of the visual as well as the semantic material.

The automated content update facilities of BA will transform briefings from information snapshots, whose value declines as the information in those snapshots becomes dated and obsolete, into renewable resources whose information can be automatically updated as needed.

The automatic generation of visual depictions for imported material, and ontology-specific interface editing extensions may actually *reduce* the effort needed to compose the visual content of a briefing, even though briefings will contain non-visual semantic content as well as traditional graphic content.

9. Related Work

Several initiatives aiming to establish a global semantic markup scheme for the web are currently being undertaken. The oldest and most widely adopted is the Dublin Core Metadata Initiative (DCMI) [5]. The DCMI has the goal of facilitating the discovery of electronic resources in the web. Its primary offering is the Dublin Core Metadata Element Set, a set of fifteen elements like Title, Creator, Subject, and Date that is used to describe web resources. The Dublin Core Metadata Element Set is the de facto worldwide standard for the description of information resources across disciplines and languages and has already been translated into 25 languages.

Newer undertakings like the European Community sponsored Ontobroker [6], its successor OntoWeb [7], and the DARPA sponsored DAML [8], go beyond the DCMI goals. Rather than annotating electronic resources to merely facilitate their discovery, these projects aim to describe electronic and real world entities using a machine understandable language that enables autonomous software agents to accurately understand and process their content [9]. Our BA is being developed under the DAML program.

There are two dimensions along which we can state requirements over semantic markup generator tools. The first dimension is the granularity of a description unit. This dimension ranges from coarse descriptions that relate a whole document with

a set of predefined conceptual categories to detailed descriptions of a document's content. The second dimension is the degree of regularity of the generated descriptions. This dimension ranges from highly regular data usually supported in relational databases to descriptions of highly unstructured and irregular information like the content of newspaper articles. The BA is aimed at detailed descriptions of irregular and unstructured documents.

The above dimensions are useful for comparing the BA with other tools for generating semantic markups. One kind of metadata generator tool is The Nordic DC metadata creator [10]. The Nordic DC metadata creator is a metadata editor for the Dublin Core Metadata Initiative. It consists in a Java applet that displays a form where users can fill in the values corresponding to the Dublin Core Metadata Element Set. The Nordic DC metadata creator generates a syntactically correct encoding of these values that a user can attach to the described document. This tool corresponds to the less elaborated form of semantic markups: Coarse descriptions based in a predefined set of conceptual categories. Another kind of tool is represented by Klarity [11], a metadata generator tool that supports the Dublin Core Metadata Element Set. Klarity is a tool that can automatically generate metadata for HTML pages based on the concepts found in the text. It uses statistic methods to allocate values based on the concepts it has identified from the 'seed' or exemplar documents that are significant to the concept in question. Klarity is another example of a tool that generates coarse metadata descriptions of documents.

A different approach for metadata generation is represented by ITTALKS [12]. ITTALKS is a portal for announcements about talks, seminars and colloquia related to Information Technology that is part of the DAML program. Although not its main focus, ITTALKS is able to generate DAML descriptions from the talks contained in its database. In this sense, ITTALKS is an example of a tool that generates descriptions from highly structured data.

Closer to the scope of the BA are the Annotation Tool of the KA^2 initiative (under the Ontobroker project) [1] and the Knowledge Annotator of the Shoe project [2]. These tools offer a GUI for authoring and attaching semantic annotations to web documents. They make available context sensitive instances and ontology browsers that facilitate the authoring of semantic descriptions. A second incarnation of the KA^2 annotation tool can also generate annotations semi-automatically from lexical analysis of text plus a vast word and domain lexicon.

These approaches contrast with the Briefing Associate in that the BA generates these markups as a byproduct of constructing the document and hence do not require the users to perform any extra activity. Additionally, because the semantic annotations are embedded in the original document (instead of being inserted in a second step using a different tool) modifying the original document does not lose the existing annotations. In the other hand, the BA approach might not be adequate for marking up existing documents that do not use the BA conventions for representing ontological relationships, and for documents whose type is not supported by the underlying COTS product.

Although the semiautomatic markup generation feature of the KA^2 annotation tool simplifies the production of semantic annotations, it stills constitutes an extra activity because the users need to check and revise the generated annotations. Furthermore, this approach is limited to textual documents that contain enough information as required to infer their semantic relationships. This limitation might

certainly exclude briefing documents because they usually contain diagrams that are not self-explained within the text.

10. Conclusions

The overload incurred in annotating documents with semantic markups should be kept to a minimum. With this goal in mind we have developed a new paradigm for facilitating the generation of semantic descriptions of the document content. Our approach consists in augmenting the same COTS products that users normally use to compose these documents with natural representations of domain ontology entities. By using these representations in composing a document, the user is simultaneously generating a semantic description of its content. We also suggest that in order to motivate document author's transition to this paradigm, these added semantic descriptions should be exploited by author-oriented tools that help improve the resulting document's accuracy, quality, and/or speed of production. It should be noted that this is just embedding the semantic web lifecycle into the document creation process so that document authors can enjoy (some of) the benefits of semantic markup.

We have implemented these ideas in the Briefing Associate, an extension to the Microsoft PowerPoint that reflects an internal semantic model of a briefing from its graphic representations of domain instances and their relationships. The Briefing Associate can produce DAML descriptions of a briefing content as a byproduct of creating the graphic content of that briefing. That semantic content can be analyzed for consistency and completeness to improve the briefings accuracy, quality, and speed of production. In addition, the Briefing Associate can automatically create graphic depictions from imported DAML descriptions, and these graphic depictions can be updated on demand to reflect changes in the imported DAML content.

References

[1] Erdmann, M., Maedche, A., Scnurr, H.-P., and Staab, S. From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools. *To appear in the Linköping Electronic Articles in Computer and Information Science.* http://www.ida.liu.se/ext/epa/cis/2001/002/tcover.html. 30.December, 2001

[2] Heffin, J, Hendler, J., Luke, S. SHOE: A prototype Language fro the Semantic Web. *To appear in the Linköping Electronic Articles in Computer and Information Science.* http://www.ida.liu.se/ext/epa/cis/2001/003/tcover.html.10.March.2001

[3] DAML+OIL (March 2001). http://www.daml.org/2001/03/daml+oil.daml

[4] Goldman, N. and Balzer, R. The ISI Visual Design Editor Generator, Proceedings of the 1999 IEEE Symposium on Visual Languages, Tokyo, Japan, September 13-16, 1999, IEEE Computer Society Press, pp. 20-27.

[5] Dublin Core Metadata Initiative. http://dublincore.org

[6] OntoBroker & Ontology Related Initiatives for the Semantic Web. http://ontobroker.semanticweb.org

[7] OntoWeb. http://www.ontoweb.org

[8] The DARPA Agent Markup Language Homepage. http://www.daml.org

[9] Tim Berners-Lee, James Hendler and Ora Lassila. The Semantic Web. *Scientific American.* May 2001. http://www.sciam.com/2001/0501issue/0501berners-lee.html

[10] Dublin Core Metadata Template. http://www.lub.lu.se/cgi-bin/nmdc.pl

[11] Klarity.http://www.klarity.com.au

[12] ITTALKS. http://ittalks.org

ITTALKS: A Case Study in the Semantic Web and DAML

R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Ian Soboroff,
Harry Chen, Lalana Kagal, Filip Perich, Youyong Zou, Sovrin Tolia
Laboratory for Advanced Information Technology
University of Maryland Baltimore County
{cost,finin,ajoshi,ypeng,nicholas,ian,hchen4,lkagal1,fperic1,yzou1,stolia1}@csee.umbc.edu

Abstract. Effective use of the vast quantity of information now available on the web will require the use of "Semantic Web" markup languages such as the DARPA Agent Markup Language (DAML). Such languages will enable the automated gathering and processing of much information that is currently available but insufficiently utilized. Effectively, such languages will facilitate the integration of multi-agent systems with the existing information infrastructure. As part of our exploration of Semantic Web technology, and DAML in particular, we have constructed ITTALKS, a web-based system for automatic and intelligent notification of information technology talks. In this paper, we describe the ITTALKS system, and discuss the numerous ways in which the use of Semantic Web concepts and DAML extend its ability to provide an intelligent online service to both the human community and the agents assisting them.

1 Introduction

With the vast quantity of information now available on the Internet, there is a need to manage this information by marking it up with a semantic language, such as DARPA Agent Markup Language (DAML) [14, 23], and using intelligent search engines, in conjunction with ontology-based matching, to provide more efficient and accurate information search results. The aim of the Semantic Web is to make the present web more machine-readable, in order to allow intelligent agents to retrieve and manipulate pertinent information. The key goal of the DAML program is to develop a Semantic Web markup language that provides sufficient rules for ontology development [20] and that is sufficiently rich to support intelligent agents and other applications [22, 36]. Today's agents are not tightly integrated into the web infrastructure. If our goal is to have agents acting upon and conversing about web objects, they will have to be seamlessly integrated with the web, and take advantage of existing infrastructure whenever possible (e.g., message sending, security, authentication, directory services, and application service frameworks). We believe that DAML will be central to the realization of this goal.

In support of this claim, we have constructed a real, fielded application, ITTALKS, which supports user and agent interaction in the domain of talk discovery. It also provides a simple web-driven infrastructure for agent interaction. In addition, ITTALKS serves as a platform for designing and prototyping the software components required to enable developers to create intelligent software agents capable of understanding and processing information and knowledge encoded in DAML and other semantically rich markup languages. To date, we have focused on developing the support and infrastructure required for intelligent agents to integrate into an

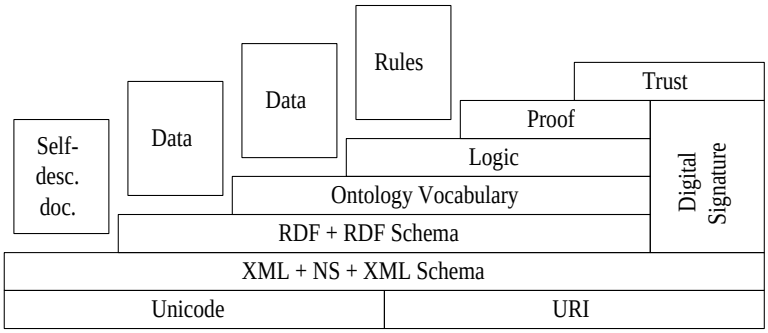


Figure 1: Tim Berners-Lee's vision of the Semantic Web is founded on a base that includes URIs, XML and RDF.

environment of web browsers, servers, application server platforms, and associated supporting languages (e.g., WEB/SQL, WEBL), protocols (e.g., SSL, S/MIME, WAP, eSpeak), services (e.g., LDAP) and underlying technologies (e.g., Java, Jini, PKI).

On the surface, ITTALKS is a web portal offering access to information about talks, seminars and colloquia related to information technology (IT). It is organized around domains, which typically represent event hosting organizations such as universities, research laboratories or professional groups, and which are represented by independent web sites. ITTALKS utilizes DAML for its knowledge base representation, reasoning, and agent communication. DAML is used to markup all the information in the knowledge base to provide additional reasoning capabilities otherwise unavailable. With information denoted in a semantically machine-understandable format, the computer can deduce additional information, a task which is difficult in a traditional database system. For example, if both ITTALKS and the user agree on a common semantics, the ITTALKS web portal can provide not only the talks that correspond to the user's profile in terms of interest, time, and location constraints, but can further filter the IT events based on information about the user's personal schedule, inferred location at the time of the talk, distance and current traffic patterns, etc. ITTALKS can also dynamically update the user's profile with incremental learning of the user's usage patterns.

ITTALKS demonstrates the power of markup languages such as DAML for the Semantic Web, drawing on its ability to represent ontologies, agent content languages and its ability to improve the functionality of agents on the web. We have developed DAML-encoded ontologies for describing event, temporal, spatial, personal, and conversational information, which enable us to represent all required knowledge in a DAML-encoded format. Moreover, these ontologies enable us to execute a computer understandable conversation. In addition, we have created several DAML-encoded classification ontologies, which provide us with additional reasoning capabilities in order to find the best matching IT talks for a particular user. Furthermore, in the ITTALKS application, any web page presented on the ITTALKS web sites contains the necessary information for an agent to retrieve the DAML-encoded description of this page as well as the contact information of a responsible agent in order to provide more effective conversation. ITTALKS thus provides each agent with the capability to retrieve and manipulate any ITTALKS-related information via a web site interface or through a direct agent-to-agent conversation. Hence, by combining the features of currently existing web applications with the DAML-based knowledge and reasoning capabilities, ITTALKS presents a true Semantic Web application.

```
<daml:class rdf:ID="Animal">
  <rdfs:label>Animal</rdfs:label>
  <rdfs:comment>An Example</rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>

<daml:class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal" />
  <daml:disjointWith rdf:resource="#Male" />
</daml:Class>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person" />
  <rdfs:subClassOf rdf:resource="#Male" />
</daml:Class>
```

Figure 2: An example of DAML-encoded knowledge.

2 Background

The Semantic Web [4, 3] is a vision in which web pages are augmented with information and data that is expressed in a way that facilitates its understanding my machines. The current human-centered web is still largely encoded in HTML, which focuses largely on how text and images would be rendered for human viewing. Over the past few years we have seen a rapid increase in he use of XML as an alternative encoding, one that is intended primarily for machine processing. The machine which process XML documents can be the end consumers of the information or they can be used to transform the information into a form appropriate for human understanding (e.g., as HTML, graphics, synthesized speech, etc.) As a representation language, XML provides essentially a mechanism to declare and use simple data structures and thus leave much to be desired as a language in which to express complex knowledge. Recent enhancements to basic XML, such as XML Scheme, address some of the shortcomings, but still do not result in an adequate language for representing and reasoning about the kind of knowledge essential to realizing the Semantic Web vision.

RDF (Resource Description Framework) [40] and RDFS (RDF Schema) [41] attempt to address these deficiencies by building on top of XML. They provide representation frameworks that are roughly the equivalent to semantic networks in the case of RDF and very simple frame languages in the case of RDFS. However, RDFS is still quite limited as a knowledge representation language, lacking support for variables, general quantification, rules, etc. DAML is one attempt to build on XML, RDF and RDFS and produce a language that is well suited for building the Semantic Web.

The goal of the DAML program (http://www.daml.org/), which officially began in August 2000, is to develop a universal Semantic Web markup language that is sufficiently rich to support intelligent agents and other applications. DAML can dramatically improve traditional ad hoc information retrieval because its semantics will improve the quality of retrieval results. Also, it will allow the intelligent agents of tomorrow to retrieve and manipulate the information on the semantic web.

3 ITTALKS

As part of UMBC's role in the DAML Program, we have developed ITTALKS; a web portal that offers access to information about talks, seminars, colloquia, and other information technology (IT) related events. ITTALKS provides users with numerous details describing the IT events, including location, speaker, hosting organization, and talk topic. More importantly, ITTALKS also provides agents with the ability to retrieve and manipulate information stored in the ITTALKS knowledge base. Below, we discuss various aspects of the system in more detail.

Unlike other web services, ITTALKS employs DAML for knowledge base representation, reasoning, and agent communication. The use of DAML to represent information in its knowledge base, in conjunction with its use for interchangeable type ontologies as described in Section 5.5, enables more sophisticated reasoning than would otherwise be available. For example, a simpler representation scheme might be able to provide the user with talks based on interest, time and location. When both ITTALKS and the user agree on a common semantics, the ITTALKS web portal will be able to perform further filtering, based on more sophisticated inference. In addition to enhancing knowledge representation and reasoning, DAML is used for all communication, including simple messages and queries, using the ITTALKS defined ontology. Moreover, ITTALKS offers the capability for each user to use his/her personal agent to communicate with ITTALKS on his/her behalf and provide a higher level of service.

3.1 Users

ITTALKS can be used anonymously, or, more effectively, with personalized user accounts. Users have the option to register with ITTALKS either by entering information manually via web forms, or by providing the location (URL) of a universally accessible DAMLized personal profile, which includes information such as the users location, his/her interests and contact details, as well as a schedule. This schedule might be as rudimentary as a list of available time periods for given days, or could even include a detailed schedule for each day. Subsequently, this information is used to provide each user with a personalized view of the site, displaying only talks that match the user's interests and/or schedule.

Since DAML is not yet in widespread use, ITTALKS provides a tool for creating a DAML personal profile. Currently, the tool constructs a profile containing only items used by the ITTALKS system. However, we believe that the profile, in one form or another, will ultimately provide a unique and universal point for obtaining personal information about the user, not just for ITTALKS, but for all information needs, and will include any sort of information the user would like to share. In the future, all services that require personal information about the user should access the same user profile, eliminating the need for the user to repeatedly enter the same information for a multitude of services. We believe that the new standard for XML Signature and Encryption under development may provide a mechanism by which users can have some control over access to parts of their profile.

3.2 Domains

To support our vision of a universal resource for the international IT research community, ITTALKS is organized around domains, which typically represent event hosting organization such as universities, research laboratories or professional groups. Each domain is represented by a separate web site and is independently maintained by a moderator who can, among other things, define the scope of the domain and delegate to other registered users the ability to edit talk entries. For example, the `stanford.ittalks.org` domain might be configured to include only talks hosted at Stanford University. On the other hand, another domain, `sri.ittalks.org`, might be configured to include not only talks about Semantic Web topics that are held at SRI, but also those at Stanford, as well as any talks within 15 mile range of the SRI facility in Palo Alto.

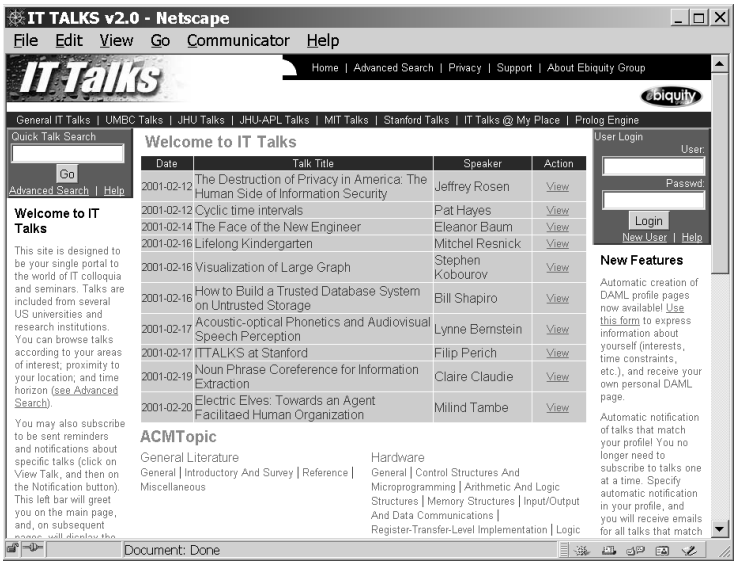


Figure 3: A screenshot depicting the main page of the ITTALKS system.

3.3 Access

The ITTALKS system is accessible either to users directly via the web, or to agents acting on their behalf. The web portal provides numerous features, including registration, search, entry and domain administration. An agent-based interface allows interaction with user agents or other services.

3.3.1 Human Interface

The web portal allows a user to browse desired information in a variety of formats, to provide the highest degree of interoperability. It permits a user to retrieve information in DAML, standard HTML format, which includes a short DAML annotation for DAML-enabled web crawlers, or WML [42] format, which supports WAP enabled phones. The ITTALKS web portal also has the ability to generate RDF Site Summary (RSS) [37] files for certain queries. These RSS files can then be used for various external purposes, such as displaying upcoming talks on a departmental web site for some particular university or domain.

3.3.2 Agent Interface

To provide access for agent based services, ITTALKS makes use of Jackal [12], a communication infrastructure for Java-based agents developed by our research group at UMBC. Jackal is a Java package, which provides a comprehensive communications infrastructure while maintaining maximum flexibility and ease of integration. The heart of Jackal is a simple conversation system, serving to maintain context for concurrent threads of conversation while providing a guide for judging behavioral correctness and modeling the actions of other agents. Jackal provides facilities for creating and manipulating user-defined conversation structures of arbitrary extent. Jackal has a very modular, loosely coupled architecture, designed to support maximal concurrency among components, accomplished with the use of multiple threads and buffered interfaces between subsystems. Its concise API allows for comprehensive specifica-

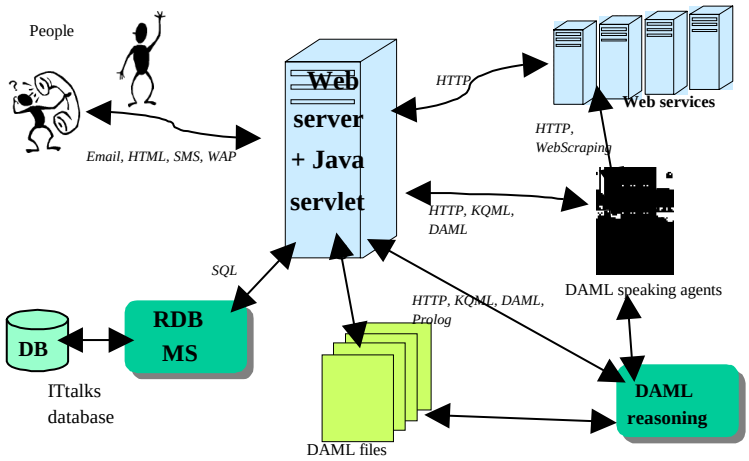


Figure 4: The architecture for ITTALKS is built around a web server backed by a relational database. Interfaces are provided to human users, software agents and web services.

tion of message requests, and for blocking or non-blocking message retrieval. Currently, it facilitates the use of KQML agent communication language [18] and employs a sophisticated protocol for agent naming, addressing and identity (KNS). Additionally, it is in the process of adapting to the FIPA standards [19, 2]. In addition, our research group, in cooperation with other universities, is developing a DAML ontology for the necessary conversation protocols.

3.4 Agents

In order to extend the capabilities of the system, we have defined a number of agents that support the operation of ITTALKS. Some can be seen as supporting services (such as external information services), while others we assume will exist in the general environment in the future.

3.4.1 ITTALKS Agent

The ITTALKS agent is a front-end for the ITTALKS system. It interacts with ITTALKS through the same web-based interface as human users, but communicates via an ACL with other agents on the web, extending the system's accessibility. At present, the agent does not support any advanced functionality, but acts primarily as a gateway for agent access.

3.4.2 User Agents

One longtime goal of agent research is that users will be represented online by agents that can service queries and filter information for them. While ITTALKS does not require that such agents exist, we recognize the added power that could be gained by the use of such agents. Therefore, ITTALKS supports interaction with User Agents as well as their human counterparts. The User Agent that we have developed understands DAML, supports sophisticated reasoning, and communicates via a standard agent communication language. Reasoning is accomplished with the use of the XSB, a logic programming and deductive database system for Unix and Windows developed at SUNY Stony Brook.

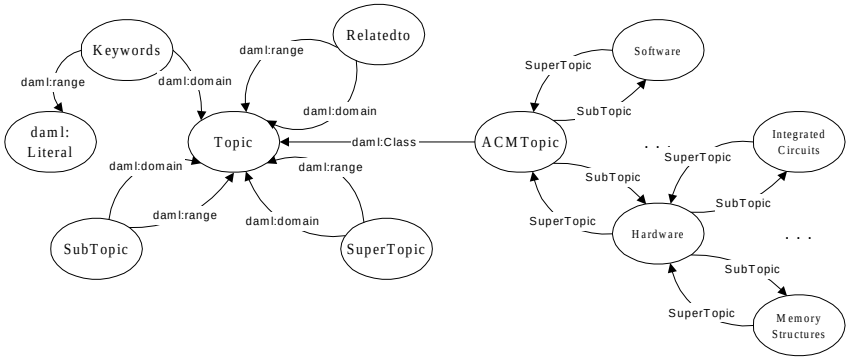


Figure 5: The Ontologies used by ITTALKS are relatively simple, such as the topics ontology used to describe talk topics and user interests.

3.4.3 Classifier Agent

ITTALKS uses a Classifier (or recommender) Agent that is invoked when a user is entering a new talk. Based on the talk's abstract, the Classifier returns ACM Classification Hierarchy Classification numbers along with a rank, in descending order. Using a local table of classification numbers and names, ITTALKS suggests to the user ten possible topics.

3.4.4 MapQuest Agent

The MapQuest Agent is a wrapper agent that allows ITTALKS to make use of external services. It interacts directly with agents (e.g. the ITTALKS agent, User Agents), and accepts requests for information such as the distance between two known locations. It then phrases an appropriate request to the MapQuest system [33], parses the results, and generates an appropriate response. Note that this agent could be generically named a Distance Agent, and make use of any external service (or combination of several, as needed).

3.5 Ontologies

The ITTALKS system is based on a set of Ontologies [1] that are used to describe talks and the things associated with them, e.g., people, places, topics and interests, schedules, etc. Figure 6 shows some of the dependencies that exist among these ontologies. The ontologies are used in the representation and processing of DAML descriptions and also as "conceptual schemata" against which the database and various software APIs are built.

We have developed a general ontology for describing the topics of arbitrary talks and papers. Using this, we have implemented an ontology to describe IT related talks based on the ACM's Computer Classification System. In addition, we currently are developing a DAML ontology for IT talks based on a portion of the Open Directory, and are considering additional classification ontologies. Figure 5 sketches some of the major classes and properties in these ontologies. These topic ontologies are used to describe talks as well as the users' interests throughout the system. This includes an automated talk classification, for which we have obtained a training collection for the ACM CCS and are also generating an Open Directory training collection to develop the necessary components. In addition, the DAML ontologies

[1] See http://daml.umbc.edu/ontologies.

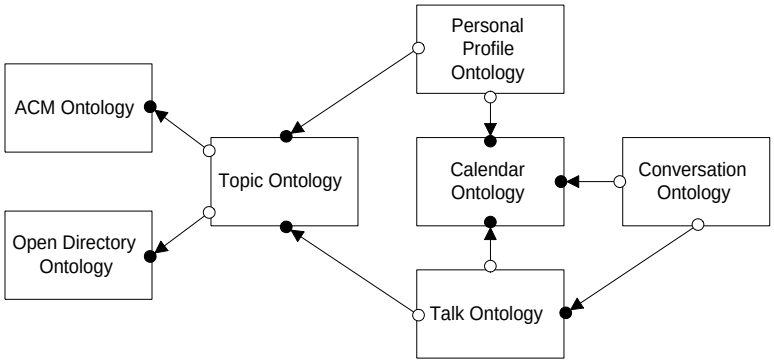


Figure 6: The relationships among the various ontologies used by the ITTALKS system.

will give a user the ability to add additional assertions in DAML to further characterize their interests. Lastly, we are also in the process of developing a component that can map topics in one ontology into topics in another, by taking advantage of the fact that nodes in each ontology have an associated collection of text as well as DAML information.

3.6 Data Entry

Currently ITTALKS requires that information about talks be manually entered via a web form interface, or be available in a DAML description available at a given URL. Although we have made this process as simple as possible (e.g., by supporting automatic form completion using information from the knowledge base and the user's DAML profile) it is still a time consuming process. Therefore, we are developing a focused web spider to collect talk announcements from open sources on the web. This spider will identify key information items using a text extraction system, and will automatically add information to the ITTALKS knowledge base. We are working with the Lockheed-Martin research group on the above task, and will use their AeroText information extraction system [1].

3.7 Architecture

The current implementation of ITTALKS uses a relational database, in combination with a web server, to provide user access to the system. To enable agents to access the system, the ITTALKS provides an interface for agent-based communication.

3.7.1 Database

The main software packages that are used in the ITTALKS system are the MySQL relational database software and a combination of Apache and Tomcat as the web portal servers. The contents of the ITTALKS knowledge base are stored in a database whose schema is closely mapped to our ontologies describing events, people, topics and locations. We have chosen MySQL because of its known reliability, and because we required software with a license that allows us to make the ITTALKS package available to additional academic and commercial institutions.

3.7.2 Web Server

As stated above, for our web, we have chosen a combination of Apache and Tomcat. This enables us to present the IT talk descriptions to the user using Java servlets and JSP files, which dynamically generate requested information in DAML, XML, HTML, RSS, and WML formats. The current ITTALKS implementation can provide information suitable for viewing on either a standard, computer-based or a WAP-enabled cellular phone.

3.7.3 Extensions

In addition, we are currently employing the Jackal agent communication infrastructure developed at UMBC and the Lockheed-Martin's AeroText information extraction system in order to facilitate ITTALKS-user agent interaction and the automated text extraction, respectively. We are in the process of modifying Jackal to provide support for FIPA ACL interoperability. Also, we are considering the possible replacement of MySQL with native XML database software such as dbXML.

4 Scenarios

We describe here a couple of typical interactions that illustrate some of the features of ITTALKS. The first involves direct use by a human user, and the second, advanced features provided through the use of agents.

4.1 Human Interaction

In this first scenario, a user, Jim, learns from his colleagues about the existence of the ITTALKS web portal as a source of IT related events in his area; Jim is affiliated with Stanford University.

Jim directs his browser to the `www.ittalks.org` main page. Seeing a link to a Stanford ITTALKS domain (`stanford.ittalks.org`), he selects it, and is presented with a new page listing upcoming talks that are scheduled at Stanford, SRI and other locations within a 15-mile radius (the default distance for the Stanford domain).

Jim browses the web site, viewing announcements for various talks matching his interests and preferred locations (as provided in his explicit search queries). He is impressed that he can see the talk information not only in HTML, but also in DAML, RSS and WML formats. Finding a talk of potential interest to a colleague, Jim takes advantage of the invitation feature, which allows him to send an invitational e-mail to any of his friends for any of the listed talks. Finally, using the personalize link on the bottom of the page, Jim creates his own ittalks.org main page, by providing the URL of his DAML-encoded profile. This customized page, listing talks based on his preferences, will be Jim's entrance to the ITTALKS site whenever her returns.

4.2 Agent Interaction

This scenario assumes that user Jim has already registered with ITTALKS, and has left instructions with the system to be notified of the occurrence of certain types of talks.

In the course of operation, ITTALKS discovers that there is an upcoming talk that may interest Jim, and of which Jim has not been notified. Based on information in Jim's preferences, which have been obtained from his online, DAML-encoded profile and from information

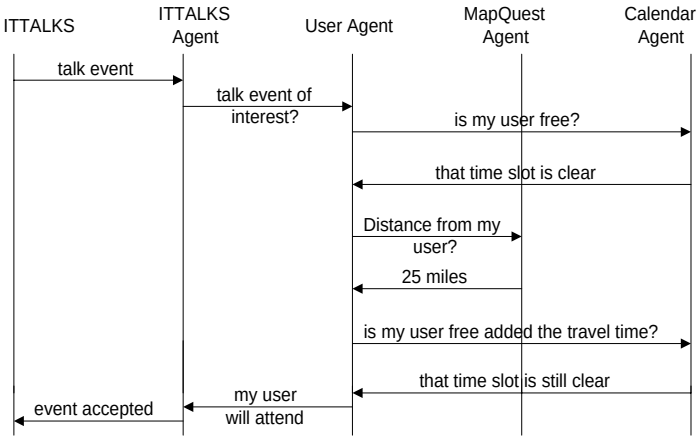


Figure 7: Interactions between the various agents described in the ITTALKS/Agent scenario.

entered directly, ITTALKS opts to notify Jim's User Agent directly. This is done via ITTALKS own agent, which forwards the message using an ACL.

Upon receiving this information, Jim's User Agent needs to know more; it consults with Jim's Calendar agent to determine his availability, and with the MapQuest agent to find the distance from Jim's predicted location at the time of the talk. Some more sophisticated interactions might take place at this time; for example, the Calendar and User agents may decide to alter Jim's schedule, and proceed to contact the User agent of some other individual. In addition, the User agent may request more information about the speaker and the event by contacting other agents or web sites, such as CiteSeer-based agent [8, 34, 9], to obtain more information necessary to make a decision. Finally, after making this decision, the User Agent will send a notification back to the ITTALKS agent indicating that Jim will/will not plan to attend. The ITTALKS agent will make the appropriate adjustments at the ITTALKS site.

5 Benefits of DAML

We believe that ITTALKS benefits significantly from its use of a semantic markup language such as DAML. DAML is used to specify ontologies that we use extensively in our system. It is also used for personal profiles, and as an agent content language. Without DAML, specifying schedules, interests and assertions about topics would be very difficult. In ITTALKS, a user can specify that from his/her perspective, two or more topics are equivalent, related, dissimilar, etc. This will allow ITTALKS to tailor the searching of talks to the users needs. As an agent content language, DAML provides more flexible semantics than KIF or other content languages that currently provide syntax only. The ultimate benefit of using DAML then lies in the ability of ITTALKS to independently interact with any DAML-capable agent without the need of a human supervision. Consequently, all these benefits, which are described in further details below, enable more efficient interaction between the system and its users, be they humans or software agents.

5.1 Interoperability Standard

As an interoperability layer, DAML allows the content of ITTALKS to be easily shared with other applications and agents. For example, a Centaurus room manager agent [26] could watch

ITTALKS for events happening in a room for which it is responsible in order to enable better scheduling. DAML also acts as an interoperability standard allowing other sites to make their talks available for inclusion in ITTALKS by publishing announcements marked up in our ontology.

5.2 Distributed Trust and Belief

Agents face a difficult problem of knowing what information sources (e.g. documents, web pages, agents) to believe and trust in an open, distributed and dynamic world, and how to integrate and fuse potentially contradictory information. DAML can be used to support *distributed trust and reputation management* [25, 31, 32]. This will form the basis of a logic for *distributed belief transfer* that will enable more sophisticated, semantically-driven rule-based techniques for information integration and fusion. We are making use of DAML's expressiveness and employing it to describe security policies, credentials and trust relationships, which form the basis of trust management. These policies contain more semantic meaning, allowing different policies to be integrated and conflicts to be resolved relatively easily. Also, it will be possible for other applications to interpret the agent's credentials, e.g. authorization certificates, correctly, making these credentials universal. Similarly, describing beliefs and associating levels of trust with these beliefs is more straightforward and the deduction of belief is uniform by different applications and services.

Authorization in a distributed system is quite different from that in a centralized system. Various schemes for decentralized security have been suggested, like Access Control Lists, Role based Access Control [38, 24], PolicyMaker [6, 5], etc. Although the above mentioned mechanisms are powerful, individually they are unable to meet all the requirements of trust management. Generally, security systems should not only authenticate users, but also allow them to delegate their rights and beliefs to other users securely, and have a flexible mechanism for this delegation. Most schemes either support only authentication, ignoring delegation altogether, or they support delegation to some extent without providing the required flexibility, or they provide insufficient restrictions on delegation of rights.

We have tried to solve this problem through the application of a chain of trust, using rights and delegations. In this system, we model permissions as the rights of an agent and associate rights with actions, so that possession of a right permits the corresponding agent to perform a certain action. These permissions can be extended by delegation from an authorized agent. We are also working with obligations, entitlements, and prohibitions and the delegation of these propositions.

Our work on distributed trust represented actions, privileges, delegations and security policy as horn clauses encoded in Prolog. In order to develop a approach that is better suited to sharing information in an open environment, we are recasting this work in DAML. We have defined an initial ontology [2] that covers the basic concepts including actions, agents, roles, privileges, prohibitions, obligations, security policies and other key classes and their properties. In applying our framework, one must extend the initial ontology by defining domain specific class of actions, permissions, etc. and creating appropriate individuals. A simple example of a delegation is given in Figure 8.

We hope to use this approach to distributed trust in ITTALKS to express the security policies which govern who can create, delete and edit talk announcements and who can further

[2] http://daml.umbc.edu/ontologies/trust-ont.daml

```
<rdf:RDF
    xmlns:rdf    = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs   = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:daml   = "http://www.daml.org/2001/03/daml+oil#"
    xmlns        = "http://daml.umbc.edu/ontologies/trust-ont#" >

  <delegation rdf:ID="Delegation1">
    <from>susan-agent</from>
    <to>marty-agent</to>
    <permission>
      <from>susan-agent</from>
      <to>marty-agent</to>
      <starttime>2001:8:1:10:00</starttime>
      <endtime>2001:8:5:24:00</endtime>
      <readfileaccess>
        <name>ReadFileAccess</name>
        <description >
          Accessing a file in read mode
        </description>
        <actor>umbc-agent</actor>
        <objects>susan-files</objects>
        <redelegatable>
          <permission>
            <readfileaccess>
              <actor>umbc-agent</actor>
              <objects>file123.txt</objects>
            </readfileaccess>
          </permission>
        </redelegatable>
        <precondition>
          <request>
            <readfileaccess>
              <objects>susan-files</objects>
            </readfileaccess>
          </request>
        </precondition>
      </readfileaccess>
    </permission>
  </delegation>
```

Figure 8: This DAML expression represents a delegation from susan to marty, allowing marty to access in read mode all susan's files marty is also given the ability to redelegate readfileaccess to file123.txt to all agents from UMBC.

redelegate these privileges to others. The host ITTALKS system might have a base policy that specified, for example, the initial privileges and obligations that the *root* user of each ITTALKS domain would have and any constraints on their delegation (e.g., that privileges could only be delegated to registered ITTALKS users). One of the privileges that a root user could have would be to extend the security policy for his domain through delegations. As root of the *umbc.ittalks.org* domain, I might delegate to all department faculty the right to create new talk announcements and the right to re-delegate this privileges to individual graduate students. I might further delegate the right to edit or delete a talk announcement to the agent who initially created it.

5.3 Data Entry Support

ITTALKS supports intelligent form filling, making it easier for users to enter and edit information in their profiles, and also to enter and edit talk announcements and other basic information. In addition, we provide automatic form filling when an editor tries to enter information about

an entity (e.g. a talk, person, room) that is already present in the knowledge base.

5.3.1 Entering Talks

In order to make ITTALKS successful, we need to make it as easy as possible for new talk descriptions to be entered into the system. We are addressing this problem using three complimentary approaches: an enhanced web interface, accepting marked up announcements, and automated text extraction. DAML plays a key role in the first two and is the target representation for the third.

5.3.2 Enhancing the Web Interface

We have used several techniques to enhance the web form interface for entering talk announcements. One of the simplest and most effective is to recognize then some of the information being entered about an object such as a person, a room or an organization has already been entered into the ITTALKS system and to "pre-fill" the remaining parts of the form from our stored information. For example, most talks at an organization are given in a small number of rooms. Once the complete information about a particular room (e.g., room number, building, address, seating capacity, longitude and latitude, A/V equipment, networking connection, etc.) has been entered for one talk, it need not be entered again.

Although the current implementation of this does not directly use DAML, its use can support a more generalized version of a web form-filling assistant. The approach depends on two ideas: (i) tagging web form widgets with DAML descriptions of what they represent and (ii) capturing dependencies among data items in DAML and (iii) compiling these dependencies into an appropriate execution form (e.g., JavaScript procedures) that can drive the web form interface.

5.3.3 Text Classification

In order for ITTALKS to filter talk announcements on topic matches, it needs to know the appropriate topics for each talk. Initially, we required that users manually select appropriate topic categories from a web interface to the ACM CCS hierarchy. This turns out to be a daunting task requiring the user to navigate in a hierarchy of nearly 300 topics, many of which about whose meaning he will not be sure. Some users will face a similar problem in trying to select topics to characterize their own interests. Ultimately we would like to use more that one topic hierarchy to classify both talk topics and user interests (e.g., ACM CCS and Open Directory nodes), which makes the problem even more difficult for our users.

To address this problem, we have built an automatic text classifier that can suggest terms in a hierarchy that are appropriate for classifying a talk based on its title and abstract. The classifier package used was from the Bag Of Words (BOW) toolkit [35] by Andrew McCallum at CMU. This library provides support for a wide variety of text classification and retrieval algorithms. We used the Naive Bayes algorithm, which is widely used in the classification literature, fairly effective, and quick to learn the 285 classes in our test collection. We plan to use the same classification agent to suggest interest terms for users based on the text found by searching their web pages.

5.3.4 Accepting Marked Up Announcements

One of the simplest ways to enter new talk announcements is to provide them as a document that is already marked up. The current ITTALKS interface allows one to enter a URL for a talk announcement that is assumed to be marked up in ontologies that ITTALKS understands. Currently, these are just the "native" ontologies that we have built for this application. In general, if some talk announcements were available with semantic markup using other ontologies, it might be possible to provide rules and transformation that could map or partially map the information into the ITTALKS ontologies. We expect that, as the Semantic Web develops, it will be more and more likely that talk announcements with some meaningful mark up will be found on the web.

5.3.5 Automated Information Extraction from Text

We would like to be able to process talk announcements in plain text or HTML and automatically identify and extract the key information required by ITTALKS. This would allow us to fill the ITTALKS database with information obtained from announcements delivered via email lists or found on the web. The problem of recognizing and extracting information from talk announcements has been studied before [16, 11] mostly in the context of using it as a machine learning application. We are developing an information extraction tool using the AeroText [1, 10] system that can identify and extract the information found in a typical talk announcement and use this to automatically produce a version marked up in DAML which can then be entered in the ITTALKS database.

5.4 User Profiles

We use personal profiles to help ITTALKS meet the requirements of individual users. A profile is a widely accessible source of information about the user, marked DAML, to which other services and individuals can refer. In the future, such a profile may be used by all web-based services that the user wants to access. The profile will ultimately provide a unique and universal point for obtaining personal information about the user for all services, preventing the need for duplication and potential inconsistencies. This profile can be easily shared, and with the use of DAML, will allow more expressive content for schedules, preferences and interests. The notion of a personal profile and a user agent are closely linked; a user might have one or the other, or both. The profile would likely express much of the information that might be encoded in a user agent's knowledge base. Conversely, an agent would likely be able to answer queries about information contained in a profile.

5.5 Modularity

With the use of DAML, we can define several ontologies for topics and switch between them with ease. Furthermore, to restrict the retrieval results, a user can perform the search with respect to a certain set of ontologies, such as the ACM or Open Directory Classification.

5.6 Application Scalability Support

As ITTALKS becomes the central repository of IT related information for various research institutes the ITTALKS knowledge base will be distributed among numerous, and possibly apriori-unknown, locations in order to provide a higher scalability and reliability support. Yet, it will be imperative that users and agents not be required to interact with all locations in order to find or manipulate the desired information. Instead, we envision that each user agent will interact with only one ITTALKS agent, which in turn will be able to efficiently locate and manage the distributed ITTALKS information. For this, we believe that a system of DAML-enabled agents can act as an intermediate between the distributed databases.

5.7 Agent Communication Language

DAML and ACLs can be successfully integrated. DAML documents will be the objects of discourse for agents that will create, access, modify, enrich and manage DAML documents as a way to disseminate and share knowledge. Agents will need to communicate with one another not only to exchange DAML documents but also to exchange *informational attitudes* about DAML documents. Using an Agent Communication Languages (ACL) agents can "talk" about DAML documents. Integrating ACL work and concepts with a universe of DAML content is our first goal. Using DAML as an agent content language will add more meaning to the message.

6 Current Status/Observations

We have currently implemented the web site and display normal HTML with embedded DAML. There is an option for viewing only DAML content for a certain page, talk, or user. All requests are made via HTTP. We also provide a form-based interface to add/modify the database, including talks and users. We have a two level moderation, with the root being the highest. The root can delegate rights to certain users, making them editors and allowing them to edit a particular domain. We also offer tools to generate personal profiles. We allow users to filter talks by interest and location. We also have a MapQuest agent that calculates the distance between a user's location and a talk.

7 Future Directions

Since most users do not currently have personal agents, we have been developing one that can be used with this system. It is our goal, however, that ITTALKS be able to interact with external agents of any type. The agent we are developing reasons about the user's interests, schedules, assertions and uses the MapQuest agent to figure out if the user will be able to attend an interesting talk on a certain date.

We are developing a framework to use DAML in distributed trust and belief. DAML expressions on a web page that encodes a statement or other speech act by an agent are signed to provide authentication and integrity. We are working on an ontology to describe permissions, obligations and policies in DAML and allow agents to make statements about and delegate them.

Currently we use only HTTP, but plan to move to Jackal for agent communication. Jackal currently supports KQML and we are in the process of adapting it to the FIPA standards. In addition, our research group, in cooperation with other universities, is developing a DAML ontology for the necessary conversation protocols.

In order to make the process of data entry more efficient, we are developing a focused web spider, which will collect talk announcements from open sources on the web and to identify the key information in these announcements using a text extraction system. The spider will add all found and relevant information to the ITTALKS knowledge base.

8 Conclusion

Effective use of the vast quantity of information now available on the web necessitates semantic markup such as DAML. With the use of such a tool, we can enable the automated or machine-facilitated gathering and processing of much information that is currently 'lost' to us. ITTALKS, our system for automatic and intelligent notification of Information Technology talks, demonstrates the value of DAML in a variety of ways. DAML is used throughout the ITTALKS system, from basic knowledge representation, to inter-agent communication.

DAML is intended to significantly enhance the usability of web content in a number of ways. It is expected that such an advance will have some cost; with DAML, this is largely in complexity. DAML-marked text is difficult for human users to read or construct. Unlike languages like HTML, which, with some small learning curve, can be used by hand, DAML requires the use of mechanized assistance. We encountered this in a number of places in the construction of ITTALKS; in the need for a classifier and other tools for using DAML-marked ontologies, in the automatic construction of a DAML user profile, in the hiding of DAML markup in general from the user in the presentation of data.

This first point leads to a second problem, which is the duplicate representation of data in multiple modes. Using a simpler markup-language, it is possible to use a single, common representation which may be used by both humans and machines. With a more complex markup such as DAML, there are good reasons to prefer the separation of data into different, paired documents. While this has some advantages, it leads to greater problems of synchronization. Note that presently, ITTALKS uses an internal relation database representation, and markup is applied to output data, so synchronization in this case is not a problem. Were multiple modes used, however, it would still be necessary for a user referencing one document to be able to identify and acquire the corresponding alternate documents.

These aspects of DAML are a necessary result of its increased expressive power. It is not necessarily the case that simpler is better - in fact, the reason DAML has been advanced is that existing markup frameworks are not sufficient to support the kind of sophisticated use of information on the web that we would like. But they are difficulties that must be addressed in order to encourage DAML's widespread acceptance.

Another point to bear in mind is that DAML's success is largely dependent on the success of efforts at constructing, representing, and merging/reconciling ontologies, something which is not always immediately obvious. Although many systems can easily make use of a language such as DAML internally in a number of ways (e.g. for representation, communication), their integration with systems in the community at large still depends on advances in our use of ontologies.

9 Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency under contract F30602-00-2-0 591 AO K528 as part of the DAML program (http://daml.org/).

References

[1] AeroText. AeroText web site. http://mds.external.lmco.com/Products_Services/aero/.

[2] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi agent systems with a fipa-compliant agent framework. *Software - Practice And Experience*, (3), 2001.

[3] Tim Berners-Lee and Mark Fischetti. Weaving the web: The original design and ultimate destiny of the world wide web by its inventor. *Harper, San Francisco*, 2001.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[5] M. Blaze, J. Feigenbaum, A. Keromytis, and J. Ioannidis. The keynote trust-management system, 1998.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. *IEEE Proceedings of the 17th Symposium*, 1996.

[7] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.

[8] Kurt D. Bollacker, Steve Lawrence, and C. Lee Giles. Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, Minneapolis, 1998. ACM Press.

[9] Sergey Brin and Lawrence Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Proceedings of the 7th International World Wide Web Conference. April 1998.

[10] Lois C. Childs. AeroText - a customizable information extraction system. Technical report, Lockheed Martin, 2001. Unpublished.

[11] Fabio Ciravegna. Learning to tag for information extraction from text. In Fabio Ciravegna and Roberto Basili, editors, *Proceedings of the ECAI Workshop on Machine Learning for Information Extraction, in conjuction with ECAI 2000*, Berlin, August 2000.

[12] R. Scott Cost, Tim Finin, Yannis Labrou, Xiaocheng Luan, Yun Peng, Ian Soboroff, James Mayfield, and Akram Boughannam. Jackal: A Java-based tool for agent development. In Jeremy Baxter and Chairs Brian Logan, editors, *Working Notes of the Workshop on Tools for Developing Agents, AAAI '98*, number WS-98-10 in AAAI Technical Reports, pages 73–82, Minneapolis, Minnesota, July 1998. AAAI, AAAI Press.

[13] Neal Coulter. Computing classification system 1998: Current status and future maintenance, report of the ccs update committee. *Computing Reviews*, January 1998.

[14] DARPA. DARPA agent markup language website. http://wwww.daml.org/.

[15] Stefan Decker, Frank van Harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michel Klein, and Sergey Melnik. The semantic web - on the roles of XML and RDF. *IEEE Internet Computing*, September/October 2000.

[16] T. Eliassi-Rad and J. Shavlik. Instructable and adaptive web-agents that learn to retrieve and extract information. Technical Report 2000-1, Machine Learning Research Group, Department of Computer Sciences, University of Wisconsin, 2000.

[17] D. Fensel et al. OIL in a nutshell. In R. Dieng et al., editor, *Knowledge Acquisition, Modeling and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, October 2000.

[18] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, 1997.

[19] FIPA. FIPA 97 specification part 2: Agent communication language. Technical report, FIPA - Foundation for Intelligent Physical Agents, October 1997.

[20] N. Guarino. *Formal Ontology in Information Systems*, chapter Formal ontology and information systems. IOS Press, 1998.

[21] Jeff Heflin, James Hendler, and Sean Luke. SHOE: A prototype language for the semantic webs. *Linköping Electronic Articles in Computer and Information Science*, 6, 2001. http://www.ep.liu.se/ea/cis/1997/013/.

[22] James Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, March/April 2001.

[23] James Hendler and Deborah McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, 15(6):72–73, November/December 2000.

[24] Herzberg, Mass, Mihaeli, Naor, and Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *RSP: 21th IEEE Computer Society Symposium on Research in Security and Privacy*, 2000.

[25] Lalana Kagal, Harry Chen, Scott Cost, Timothy Finin, and Yun Peng. An infrastructure for distributed trust management. In *Working Notes of the Second Workshop on Norms and Institutions in Multi-Agent Systems, Fifth Internation Conference on Autonomous Agents (Agents '01)*, Montreal, Quebec, Canada, May 29 2001.

[26] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus: A framework for intelligent services in a mobile environment. In *Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, April 2001.

[27] Robert Kass and Tim Finin. Architectures for intelligent interfaces: Elements and prototypes. 1996.

[28] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *Data Engineering Bulletin*.

[29] Ora Lassila. Web metadata: A matter of semantics. *IEEE Internet Computing*, 2(4):30–37, 1998.

[30] Ora Lassila and Deborah McGuiness. The role of frame-based representation on the semantic web. *Linköping Electronic Articles in Computer and Information Science*, 6, 2001. http://www.ep.liu.se/ea/cis/1997/013/.

[31] Ninghui Li, Joan Feigenbaum, and Benjamin Grosof. A logic-based knowledge representation for authorization with delegation (extended abstract). *Proc. 12th IEEE Computer Security Foundations Workshop, Mordano, Italy*, June 1999. IBM Research Report RC 21492.

[32] Ninghui Li and BBenjamin Grosof. A practically implementable and tractable delegation logic. *IEEE Symposium on Security and Privacy*, May 2000.

[33] MapQuest. MapQuest website. http://www.mapquest.com/.

[34] James Mayfield, Paul McNamee, and Christine Piatko. The jhu/apl haircut system at trec-8. *The Eighth Text Retrieval Conference (TREC-8)*, pages 445–452, November 1999.

[35] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996. http://www.cs.cmu.edu/ mccallum/bow/.

[36] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEEIntelligent Systems*, 16(2), March/April 2001.

[37] Netscape. RSS website. http://my.netscape.com/publish/formats/rss-spec-0.91.

[38] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and C harles E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[39] S. Staab, M. Erdmann, and A. Maedche. Ontologies in RDF(S). *Linköping Electronic Articles in Computer and Information Science*, 6, 2001. http://www.ep.liu.se/ea/cis/1997/013/.

[40] W3C. RDF website. http://www.w3c.org/RDF.

[41] W3C. RDFS website. http://www.w3c.org/TR/rdf-schema.

[42] WAP Forum. WML spec. http://www1.wapforum.org/tech/documents/SPEC-WML-19991104.pdf.

Open Learning Repositories and Metadata Modeling

Hadhami Dhraief, Wolfgang Nejdl, Boris Wolf, Martin Wolpers
Knowledge Based Systems
Institute of Computer Engineering
University of Hannover
Appelstr. 4, D-30167, Hannover, Germany
Tel: +49 511-762-19714
Fax: +49 511-762-19714
E-mail: {dhraief, nejdl, wolf, wolpers}@kbs.uni-hannover.de

Abstract. Building repositories for e-learning is an iterative process and course content and course structure are always changing. We realized the necessity to separate content from structure of a given course during the conception of our first e-learning repository, which we called KBS-Hyperbook, several years ago at our institute. This system has been built around a conceptual model for structure and contents of the domain, which is expressed in the O-Telos conceptual modelling language. To ease exchange of metadata between such repositories, the Open Learning Repository (OLR), an e-learning repository we built during the last year to experiment with various features useful for such repositories, has been developed using RDF/RDFS as modelling language.
In the first part of this paper, we describe the OLR system in more detail, and show how it uses RDF/RDFS as its underlying modelling language to express information about the learning objects contained in the repository, as well as information about the relationships between these learning objects. Based on our experience in meta-modelling using different modelling languages, we will in the second part of this paper discuss RDF/RDFS and O-Telos modelling in more depth and will analyse similarities and differences of these two modelling languages.

Keywords
Meta-modelling, RDF/RDFS, conceptual modelling, hypermedia, learning repositories.

1 The Open Learning Repository

1.1 Motivation

Our Open Learning Repositories aim at metadata-based course portals, which structure and connect modularised course materials over the Web. The modular content can be distributed anywhere on the internet, and is integrated by explicit metadata information in order to build courses and connected sets of learning materials. Modules can be reused for other courses and in other contexts, leading to a course portal which integrates modules from different sources and authors. Semantic annotation is necessary for authors to help them choose modules and to connect them into course structures.

We use a relational database to store all metadata, but store no content in the database itself. The stored metadata represent information about the structure and the

access paths within a particular course, the URLs as identifiers for single elements (modules, courslets, course units, subunits, etc.) and other useful metadata about the content itself (i.e. Dublin Core or IEEE LOM metadata). We are currently using the OLR system in the context of two courses, one in artificial intelligence and one in software engineering.

1.2 OLR functionality

The OLR repository can store RDF (Resource Description Framework) [1] metadata from arbitrary RDF schemas. However, we have chosen not to implement a one-size-fits-all approach, and follow a customisable approach, implementing different interfaces together with their schemas and metadata for different courses using a common infrastructure. Initial loading for a specific course is done by importing an RDF metadata file (using XML syntax) based on this course's RDFS [2] schema. Our Artificial Intelligence course prototype uses a simple schema describing course structure (units, subunits, elements and arbitrary links between these elements) and simple cataloguing of its elements using the Dublin Core metadata [3] set. We are currently moving these metadata to the LOM standard, using the recently developed LOM-RDF-binding.

The web interface for navigating the course follows a multi-view approach. A user visiting the course currently has a choice between three different navigation schemes. The first one is a hierarchical tree-like navigation

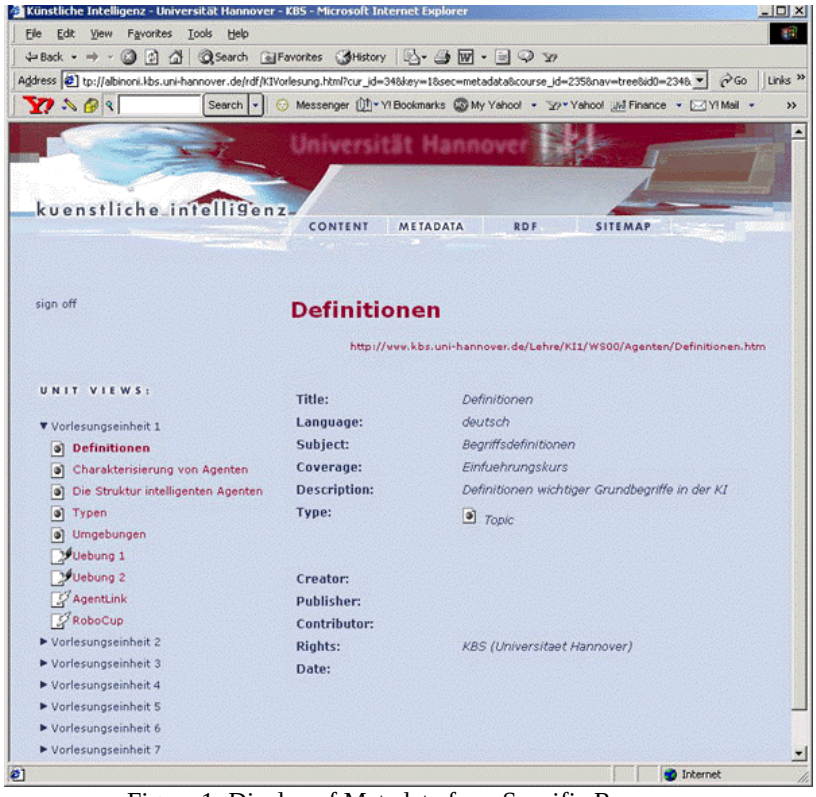


Figure 1: Display of Metadata for a Specific Resource

directly reflecting the course structure stored in the database. A visitor may open and close units and subunits to display the elements/pages of the logical document (figure 5). The second view provides a trail navigation where the user has the possibility to move forward and backward on a trail. Third we are experimenting with a semantic net or context net navigation. In this approach the user can view units in different contexts, navigation is

implemented as a kind of fish-eye view with the current unit located in the centre surrounded by related units and contexts. All navigation elements are created dynamically on demand.

In addition to displaying course content we are providing different ways of reviewing the metadata stored about course elements. Either the system displays metadata in a nicely formatted way suitable for a human reader or it generates the corresponding RDF source in XML notation (figure 1).

For content developers we implemented an enhanced web interface which allows the developer to manipulate metadata through HTML forms (figure 1). The OLR system translates all user input into suitable SQL update and insert statements hence avoiding to confront the user with having to understand XML/RDF notation. To evaluate OLR usage, the system tracks all user behaviour in the database, including which course elements are accessed and when, which updates are made and by whom. We are using this information to evaluate different navigation schemes and different types of course units.

1.3 OLR-Architecture

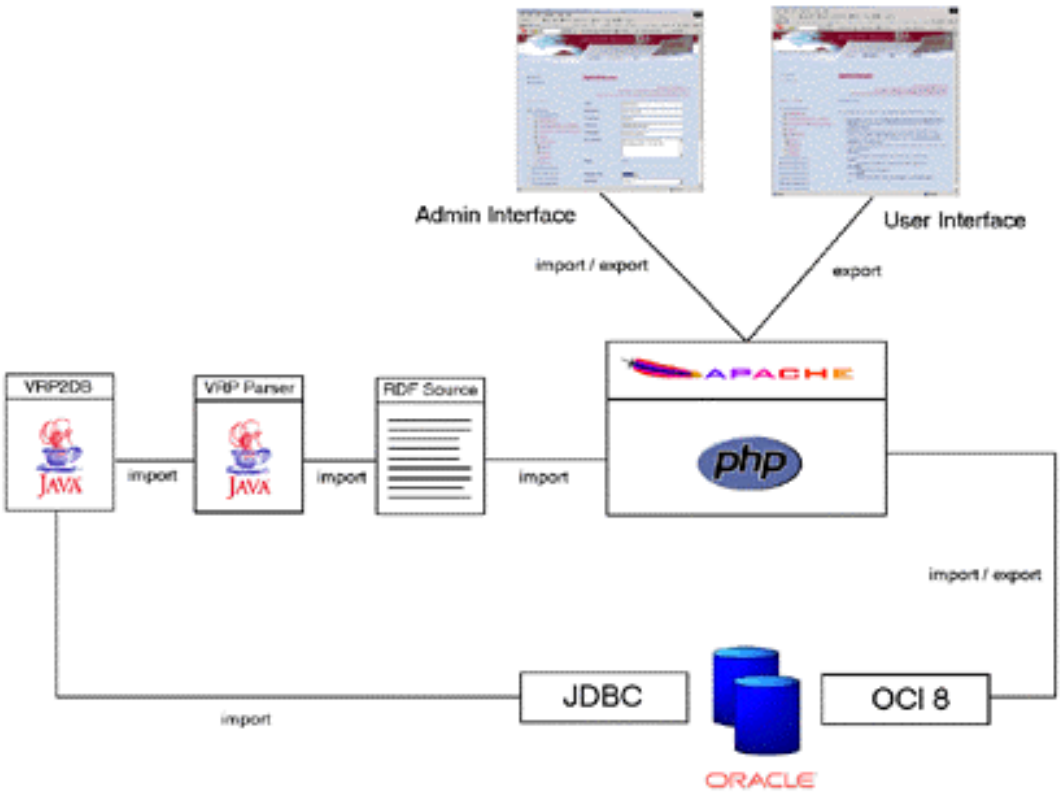


Figure 2: OLR Architecture

The OLR architecture is shown in figure 2. The system is based on a 3-tier architecture. As front end any state-of-the-art web browser may be used (IE5, NS4). The mid-tier is a combination of Apache Web server and PHP4 module. The backend holds an Oracle 8i database and can physically be the same machine as the one running the Web server.

Whenever the user selects a link or button Apache delegates the client request to the PHP module executing the appropriate PHP script. In most cases this script will need to interact with the database since it stores all RDF metadata. For communication with Oracle PHP uses its built-in OCI8 interface. The PHP script evaluates the data returned by Oracle

and dynamically creates a HTML page which in turn is sent back to the client browser initially requesting the page.

In addition the web interface allows to upload raw RDF source code in XML syntax to be stored in temporary files within the server's file system. A shell script then runs the VRP parser [4] against these RDF metadata. The generated triples are input to a Java application using the JDBC interface which imports all statements into the database.

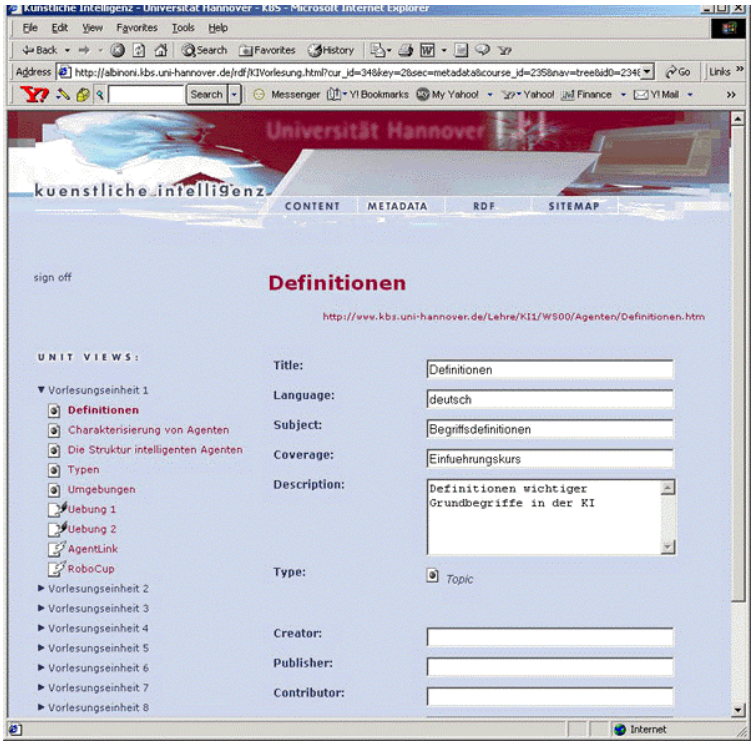


Figure 3: Adding New Metadata

1.4 Technology

1.4.1 RDF Annotation

The OLR system stores virtually anything it knows about courses as RDF metadata. In web based learning and teaching, the trend is to encode learning materials with meaningful and machine understandable metadata in order to facilitate modular and reusable content repositories.

One of the practical uses of RDF, as it has been described by W3C, is in Web sitemaps. "*The RDF schema specification provides a mechanism for defining the vocabulary needed for this kind of application*" [5].

Thus, with RDF, we can describe for our application, how modules, course units, courselets are related to each other or which examples or exercises belong to a course unit, RDF metadata used in this way are called structural or relational metadata. Another practical use of RDF is the description of web pages/units, which is mandatory to build a course based on modular content, distributed over different sites. To standardize these kinds of descriptions, initiatives like IMS and IEEE LOM specify schemas suitable for learning objects, and we have been involved in the German LOM version as well as in a LOM-RDF-binding suitable for these learning objects.

RDF (Resource Description Framework) is supported by a growing Web community. The primary target of RDF is to provide a standardized way of creating and using such specialized metadata schemas to describe resources on the Web.

Some of the goals the W3C aims to reach using RDF are:

- Resource Discovery to improve the results of Search Engines.
- Cataloguing to describe content and its relationships at a particular Web.
- Interoperability and Knowledge Sharing for information exchange between different applications, Software Agents etc.
- Logical Document: Several pieces of content physically distributed over the Internet build one single Logical Document, where RDF is the glue holding these resources together.

Everything in RDF is expressed through statements, which are triples consisting of subject, predicate and object (corresponding to instantiated binary predicates). Expressing the sentence "Smith is the author of the HTML document that can be found at the URL "http://www.xyz.com/somedoc.html"" for example is done by a statement, where "http://www.xyz.com/somedoc.html" is the subject of our statement, its predicate is "author" (which is a property in RDF terminology) and its object is the literal "Smith". Another possibility would be to use a resource (with an URL) as the object of such a statement, like "http://www.xyz.com/smith.html", assuming we want to use this URL as identifier for the person Smith.

This simple example reveals the basic building blocks of any RDF statement: resources and literals. Anything that can be reached by a URL is a resource whereas a literal is a simple character string. Subjects and predicates always need to be resources while an object may be either resource or literal. In addition predicates normally are properties described by an RDF schema.

The RDF specification does not insist on any implementation of the statement concept in particular. It introduces a graph representation suitable for the human reader and an XML-encoding of that graph suitable for XML based parsers. The XML encoding is probably the most popular RDF representation.

To create self-defined predicates like "author" in our example, one needs to create an RDF schema. Like RDF metadata these RDF schemas consist of statements and hence can be expressed utilizing the same XML syntax or any other representation.

With RDF Schema resources can be modelled as classes and predicates as properties. Thus it is possible to constrain the type of a predicate's range and domain. For example we can say that the predicate *author* may only point to resources that are instances of a class *Person* and may only be applied to resources being instances of a class *Book*.

Since we decided to utilize RDF in OLR for both the annotation of content as well as the description of course structures we developed an RDF schema for this purpose. Our implementation focuses on the cataloguing/annotation and on the logical document features of RDF. An OLR course is a Logical Document and cataloguing is used to store element information (e.g. title, author).

Each course consists of a number of units that contain elements and further subunits. Each element represents any kind of Internet resource accessible through a known URL. For the first version of our introductory course on Artificial Intelligence we defined five types of basic elements: Topics, examples, slides, exercises and further references. This choice reflects the typical building blocks of a lecture at a university on an abstract level. If necessary, further element types can be incorporated easily to satisfy other people's needs (we are using additional elements in our Software Engineering course). The basic building blocks (units and elements) are linked together in a tree-like structure that represents a

course. Each element is described by metadata. The vocabulary describing each element is basically the Dublin Core Metadata set.

We currently use RDF sequences to link elements to units and units to courses. This is necessary because the order of the course elements is essential. The disadvantage of this is, that in the current version of RDF Schema it is not possible to constrain the type of container elements. In the second part of the paper we include several examples, which use stronger typing constraints instead of RDF sequences.

Database Schema

In essence, everything in RDF is expressed through statements: simple triples composed of resources, namespaces and literals - no matter how complex the RDF schema behind might be. XML syntax is the standard approach for hiding RDF in HTML pages it describes. This approach always requires a parser to analyse the meta-information and it conflicts with one of RDF's key concepts where a group of RDF statements makes propositions about several distributed resources linking them together to one Logical Document.

In contrast, using triples directly makes it easy to store RDF metadata in a relational database. Doing so enables us to create a repository for metadata managed at one central location using relational database technology. This approach separates the metadata from the content it describes. SQL queries are used to extract the relevant RDF statements.

An obvious advantage of storing RDF in a relational database is performance: A SQL query selecting a couple of statements can be much faster than parsing an RDF document in XML representation to retrieve the same results. Especially when a lot of similar queries are executed the database's query optimiser and cashing mechanisms can speed things up considerably. When looking at large numbers of statements compact storage is another plus for the database approach: Within a set of RDF metadata a lot of literals tend to occur more than once. Namespaces are a good example for this characteristic: Every resource name is preceded by a namespace and often these namespaces are similar or identical. Being kept in a separate table, each namespace needs to be stored in the database only once. For multiple usage any namespace only needs to be referenced by its ID.

For our OLR server, we modified the McBride schema, which is one of several suggestion presented on the RDF/DB Page from Sergey Melnik [6] , also discussed within the RDF community. The OLR system is based on the Oracle 8i database, but any standard relational database would be suitable.

The main table in our database is RDF_STATEMENT. This table represents the relationship between the three parts of a statement consisting of RESOURCE (stored in RDF_RESOURCE), PREDICATE (also stored in RDF_RESOURCE) and OBJECT (stored in either RDF_RESOURCE or RDF_LITERAL). Therefore RDF_STATEMENT contains three main attributes: SUBJECT, PREDICATE and OBJECT. These attributes are references to the resource and the literal table. Since the object can either be a resource or a literal, we use two attributes for OBJECT: OBJ_RESOURCE and OBJ_LITERAL.

The Open Learning Repository is a repository to integrate, manipulate and annotate more than one course. Thus, we need to store large amounts of statements for every course. For this purpose, we utilize the table RDF MODEL. Each model currently corresponds to one course.

Distinctions to the McBride schema

Because OLR is used in a learning context, we establish different user groups with different roles and rights. Every group may have a specific view on courses and metadata.

Hence we define a table RDF_USER for user administration which is connected to the other tables via the attribute USR. We also add the attribute MODIFIED representing the last modification date.

In OLR all dynamic content is created based on SQL queries stored in the table SQL_QUERY together with a short description to facilitate the reuse of such queries and to support the PHP interface. From a developers perspective this greatly enhances reusability and maintainability of the underlying PHP source code.

In order to evaluate the different visualizations and navigation possibilities in OLR, we define a table RDF_TRACK to record the user behaviour while accessing course elements (which resources have been visited, in which order, how often, in which view). Our current database schema is shown in figure 4.

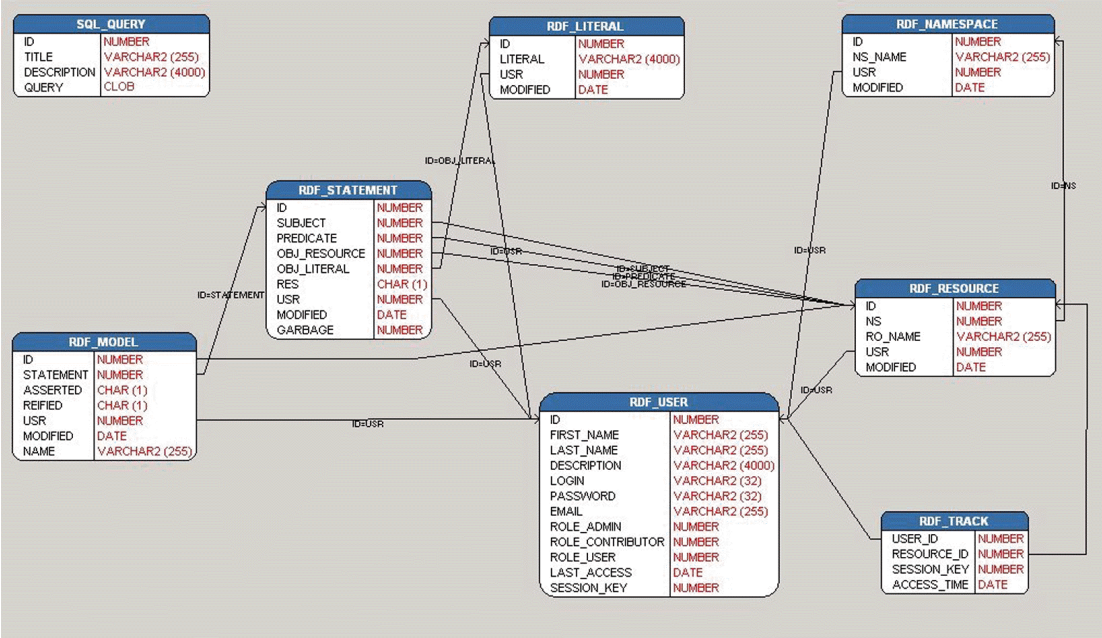


Figure 4: Database Schema

1.4.4 Architecture and Features of the OLR Web Interface

The Web interface for browsing and manipulating OLR courses needs to be highly dynamic since it needs to take into account the current state of the database. For this reason all HTML code is generated on demand by PHP scripts. To control the complexity of the system the PHP scripts are organized in several layers. Structure and purpose of the different layers are briefly outlined below.

Database access with PHP is straightforward: It already comes with a built-in API for communicating with an Oracle database through the standard OCI8 interface. We designed a number of SQL queries to suit the special needs of the OLR system. These queries are stored in a database table SQL_QUERY itself along with a unique ID, a query name and a short text describing the query's purpose. This approach greatly enhances maintainability and transparency of the system. Queries may contain parameters like resource IDs specified in brackets.

Core of the code for running SQL queries is the PHP class *RDFStatement*. Its constructor requires the query name and eventually a number of parameters. *RDFStatement* then executes the query and transfers all results into a PHP array. All database specific code is hidden behind the public interface of this class.

On top of the *RDFStatement* class we develop the OLR API - a growing number of PHP functions like *getResourceTitle(resource_id)* that take some resource ID as in-parameter and retrieve all statements about the specified resource for a specific property. These *getResourceXXX()* - functions utilize the *RDFStatement* class. Note that database primary keys (usually integer values) serve as in-parameters to identify resources rather than a combination of namespace and literal which tends to be long strings. This is extremely useful for our web interface since it keeps track of all state information (e.g. current course, unit or element) by URL parameters. The OLR API is accompanied by a number of other APIs such as an API for user and session management and an API for import and export of RDF source in XML syntax.

The next layer consists of a number of basic building blocks – PHP script fragments calling API functions and performing the HTML markup of the returned results. For instance there are PHP blocks for creating the different navigation elements or for displaying content or metadata of a course element. The final abstraction layer is represented by templates. In essence templates are HTML files composed by dynamically putting together the basic building blocks. Most templates follow the same structure with a navigation element on the left, a content area on the right and above that a header section displaying title and essential metadata. The templates also verify user access rights.

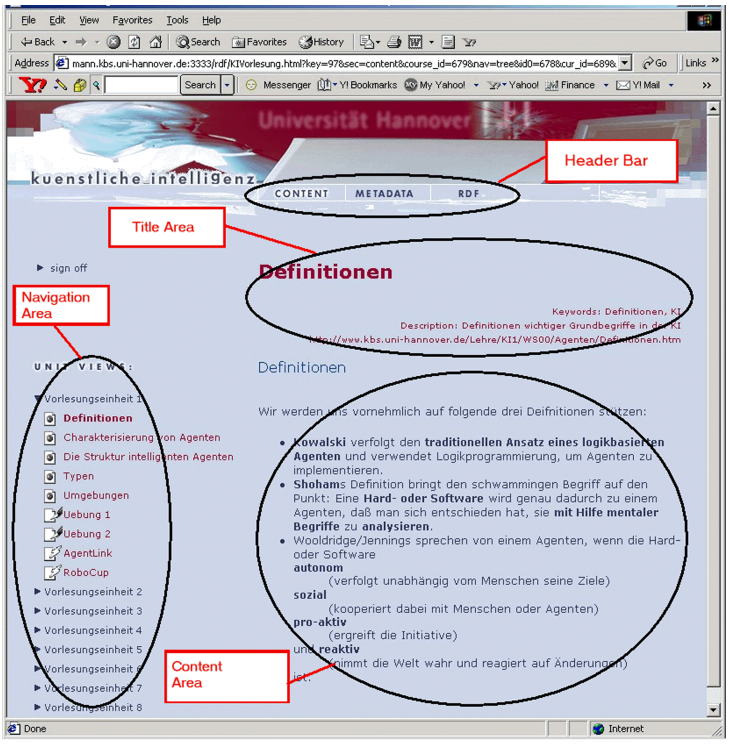


Figure 5: OLR sample template

The structure of the templates directly supports our multi-view vision. If for example you want to use a trail instead of a hierarchical navigation only one line of code needs to be changed in the appropriate template to replace inclusion of the hierarchy-block by the trail-block.

Though all RDF data are stored in the database tables the content contributor's web interface allows direct import and export of RDF source in XML syntax. After inserting XML code describing an OLR course by copy&paste into an HTML form its content is uploaded to the server, analysed by the VRP parser and then imported into the database by a java application through the Oracle JDBC interface. A newly imported course then

appears in the list of available courses. Without any knowledge of RDF or the specifics of the underlying OLR schema another content contributor then has the opportunity to modify the course through clearly arranged HTML forms. The system allows to create, modify or delete course elements and units. This is one conceptual advantage of the combination database plus web interface over the standard approach of hiding some static XML RDF within an HTML file: A authorized subgroup has the opportunity to dynamically change and extend the content of the repository through an intuitive interface and the database always keeps track of who did what and at which time modifications where issued. In addition it is possible to export XML RDF metadata on any level of granularity: One can export the XML RDF for a single course element, a unit including all its elements and subunits or a complete course. This feature is beneficial for reuse when creating new courses and supports metadata processing by other RDF XML compatible applications.

2 Comparing RDF/RDFS to the O-Telos modelling language

2.1 Motivation

As noted above, RDF is a simple but quite powerful modelling language to annotate WWW resources with semantical information. RDFS enables the simple construction of conceptual models of sets of WWW resources, and on the other had has been designed as a quite flexible representation language for these conceptual models. Unfortunately, the RDF Schema Specification [2] fails to give simple, yet formal explanations of RDFS concepts, which causes a lot of confusion when one really tries to use all RDFS possibilities. RDFS tries to be as self-expressible as possible, which leads to several properties playing dual roles both as primitive constructs and as specific instances of RDF/RDFS properties (*rdfs:domain*, *rfds:range*, *rdfs:subClassOf*, and *rdf:type*, see also the detailed discussion in [7]), where these properties are both defined in the RDF or RDFS-Schema and are used to define those schemas at the same time. On the other hand, the self-expressibility of RDFS falls short of fulfilling its promise for meta-modelling, because of the constraints of the underlying triple model, only a three level modelling hierarchy is possible (*rdfs:class*, specific classes as instances of *rdfs:class*, and instances of classes).

Another drawback of RDFS is its poor support of the reification of statements. An object identifier must be assigned explicitly to each statement that is to be reified. This has to be done by adding explicit statements about the subject, predicate and object of the specific statement.

Building on our previous work on open learning repositories [8], [9], we will in this second part of the paper compare RDF/RDFS modelling and annotation with the conceptual modelling language O-Telos, which has been strictly axiomatized in [10], based on the formalization of Telos (see e.g. [15]). As a conceptual modelling language, O-Telos is used in various contexts to describe and formalize conceptual models [9], [11], [12], [13]. As for reification, O-Telos, being based on 4-tuples instead of triples, assigns a unique object identifier to each statement, which can be used to directly reference that statement.

In this paper, we will compare RDF/RDFS with O-Telos, and discuss possible mappings from RDF to O-Telos and back, which is useful in our context (making it possible to exchange metadata between our O-Telos- and RDF-Hyperbook Systems), and also sheds light on some advantages and disadvantages of the design decisions of RDFS. In [16] we formalize an RDF variant we call O-Telos-RDF based on the O-Telos model, which allows annotation in a way very similar to RDF, but extends RDFS with enhanced reification and meta-modelling capabilities.

2.2 An introduction to O-Telos

O-Telos is a deductive object-oriented conceptual modelling language very suitable for modelling and meta-modelling tasks. It has been implemented in the ConceptBase database system [12]. Its object-oriented constructs like object, class, meta-class, etc. are expressed using a frame syntax. Each frame declares an object by stating its name, the classes it subclasses, the classes it instantiates and the attributes it declares or instantiates.

Frames are declared using predefined classes: Individual containing all individuals as instances, Attribute containing all attributes as instances, Class containing all classes as instances, String, Integer, etc. The use of the predefined classes is defined by a set of axioms to insure referential integrity, correct instantiation and inheritance.

The following example is taken from a simplified version of the OLR schema. It is used to illustrate the O-Telos language:

The lecture material of a course consists of course units, which group the specific elements. All units/elements can be annotated according to Dublin Core, i.e. they have a name and a description etc..

The model of the above example declares the following O-Telos frames, which define the two classes course and course unit as well as a Dublin Core Class, and the corresponding attributes:

```
Class DC_Unit with
   attribute
      about: URL;
      title: String;
      description: String
end
Class Course isA DC_Unit with
   attribute
      units: CourseUnit
end
Class CourseUnit isA DC_Unit with
   attribute
      parent_course : Course;
      theory_unit: TheoryUnit;
      example_unit: Example
end
```

The frame *Course* declares a class named *Course* consisting of arbitrarily many units. A unit is declared by the frame *CourseUnit*, and groups *TheoryUnits*, *Examples*, etc. Both are subclasses of *DC_Unit*, stating that they can have a *title* and a *description*, both of type *String*.

The next frames declare the individuals, e.g. a course unit with the title “Lecture Unit 1”, the description “Introduction to Intelligent Agents”. This resource belongs to the course “Introduction to AI 1” which is an introductory course in Artificial Intelligence. Additionally, this resource belongs to another course “AI 2” which is an advanced course in Artificial Intelligence.

```
Individual IntroAILecture in Course with
  title
    t1 : "Introduction to AI 1"
```

```
  description
    d1 : "Introductory course in AI"
end
Individual AdvancedAILecture in Course with
  title
    t1 : "AI 2"
  description
    d1 : "Advanced course in AI"
end
Individual IntroAILectureUnit1 in CourseUnit with
  title
    t1 : "Lecture Unit 1"
  description
    d1 : "Introduction to Intelligent Agents"
  theory_unit
    tu1: "http://www.kbs.uni-hannover.de/.../Definitions.htm";
    tu2: "http://www.kbs.uni-hannover.de/.../Characterisation.htm"
  parent_course
    c1 : IntroAILecture;
    c2 : AdvancedAILecture
end
```

The frame *IntroAILectureUnit1* shows how the declared attributes *title*, *description*, *theory_unit* and *parent_course* are instantiated. The *theory_unit* and *parent_course* attributes show that O-Telos attributes usually are multi-valued.

The frames are translated to sets of propositions which can be stored e.g. in the ConceptBase database. The definition of O-Telos propositions is a relation P(oid,x,l,y) with oid being the identifier, x being the source, l being the label and y being the destination. Consequently P(oid,x,l,y) states a relationship called l with ID oid from object x to object y. O-Telos defines specific interpretations for four predefined types of propositions. The first of these types is the object declaration P(oid,oid,l,oid) declaring an object named l. As second predefined type an instance relationship is expressed using the proposition P(oid,x,*instanceof,y) stating that x is an instance of y. The third type declares the inheritance relationship by stating propositions of the kind P(oid,x,*isa,y) saying that x is a specialisation of y. The fourth predefined type of proposition P(oid,x,l,y) represents ordinary attributes: x has an attribute named l with value y.

2.3 Simple mapping of RDF to O-Telos

Let us now construct a simple mapping from RDF to O-Telos and vice versa. We will recognize, that both languages are based on very similar ideas for their basic representation.

We start with a simple RDF declaration:

```
<rdf:Description ID="LectureUnit1">
   <rdf:type resource="http://.../olr_schema_6#Unit"/>
   <dc:title>Lecture Unit 1</dc:title>
   <dc:description>Introduction to intelligent agents</dc:description>
   <olr:parentCourse rdf:resource="#AILecture"/>
   <olr:theoryUnit rdf:resource="http://.../Agents/Definitions.htm"/>
   <olr:theoryUnit rdf:resource="http://.../Agents/Characterisation.htm"/>
   <olr:theoryUnit rdf:resource="http://.../Agents/Structure.htm"/>
```

```
   <olr:theoryUnit rdf:resource="http://.../Agents/Types.htm"/>
</rdf:Description>
```

This RDF declaration can be mapped to the following O-Telos frame which contains basically the same information:

```
Individual LectureUnit1 in CourseUnit with
    dc_title
       t1: "Lecture Unit 1"
    dc_description
       d1: "Introduction to intelligent agents"
    parent_course
       pc1: AILecture
    theory_unit
       tu1: "http://www.kbs.uni-hannover.de/.../Definitions.htm";
       tu2: "http://www.kbs.uni-hannover.de/.../Characterisation.htm";
       tu3: "http://www.kbs.uni-hannover.de/.../Structure.htm";
       tu4: "http://www.kbs.uni-hannover.de/.../Types.htm"
end
```

The example shows that the *rdf:type* property is mapped to the O-Telos relationship in (instanceof). Also the property declarations *dc:title*, *dc:description*, etc. are mapped to the respective O-Telos attributes. Both representations require the declarations of the objects/classes *Unit/olr_unit* and the course *AILecture*.

2.4 Enhancing the simple mapping (descriptions and aggregations)

A more in-depth examination of the RDF Model and Syntax Specification and our OLR Schema shows that we can distinguish two types of general classes in RDF. The first type are classes whose instances group/aggregate other instances. We will call these classes *aggregation classes*. In RDF an *aggregation class* is defined using the following statement:

```
<rdf:Description ID="...">
</rdf:Description>
```

These aggregation classes sometimes include additional attributes for their aggregates. As shown in the above example these types of classes can directly mapped to O-Telos constructs.
The second type of the general classes in RDF are classes whose instances are assigned to web pages directly. We will call these classes *annotation classes* (see also the discussion in [7]). In RDF an *annotation cla*ss is defined using the following statement:

```
<rdf:Description about="http://...">
</rdf:Description>
```

These annotation classes define attributes to describe the assigned web pages. Annotation classes can be used in various RDF schemas to declare attributes on the same resource (referenced by its URI). Thus annotation objects can be mapped to O-Telos constructs only if there is no other annotation object stating some attribute about the same resource. Because the O-Telos object takes the URI as its unique ID and all other attributes are referenced as above. In general this cannot be assured, as RDF, in contrast to (the frame syntax of) O-Telos, is a property centric language, where properties about a given resource can be declared in different locations. To reflect this modularity, we need a different approach for mapping RDF annotation classes to O-Telos.

As mentioned, resources which are described by the RDF declaration <rdf:Description about="http://..."> have no ID property. They are just groupings of attributes, as the following example shows:

```
<rdf:Description about="http://.../Agents/Definitions.htm">
   <rdf:type resource="http://.../rdf/olr#TheoryUnit"/>
      <dc:title>Definitions</dc:title>
      <dc:description>Definitions of the basics of AI</dc:description>
      <dc:subject>Definitions</dc:subject>
      <dc:language>german</dc:language>
      <dc:coverage>Introductory course</dc:coverage>
      <dc:rights>KBS (Universität Hannover)</dc:rights>
      <olr:parentUnit rdf:resource="#LectureUnit1"/>
</rdf:Description>
```

Seven attributes are assigned to the web page, which is defined by the URL "http://.../Agents/Definitions.htm".

Table 1. RDF-Triples of the single declaration about "http://.../Agents/-Definitions.htm"

Nr.	Subject	Predicate	Object
1	http://.../Agents/Definitions.htm	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://albinoni.kbs.uni-hannover.de/rdf/olr#TheoryUnit
2	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#title	Definitions
3	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#description	Definitions of the basics of AI
4	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#subject	Definitions
5	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#language	German
6	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#coverage	Introductory course
7	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#rights	KBS (Universität Hannover)
8	http://.../Agents/Definitions.htm	http://…/rdf/olr_schema_5#parentUnit	online:#LectureUnit1

Table 1 shows the RDF-triples representing the RDF declaration of properties to the resource "http://.../Agents/Definitions.htm". The triples are generated by the SIRPAC [14] parser.

The above example can also be expressed in two separate RDF declarations about the resource "http://.../Agents/Definitions.htm". Both declarations assign values to attributes but represent two different grouping objects.

```
<rdf:Description about="http://.../Agents/Definitions.htm">
   <rdf:type resource="http://.../rdf/olr#TheoryUnit"/>
      <dc:title>Definitions</dc:title>
      <dc:description>Definitions of the basics of AI</dc:description>
      <dc:subject>Definitions</dc:subject>
</rdf:Description>

<rdf:Description about="http://.../Agents/Definitions.htm">
   <rdf:type resource="http://.../rdf/olr#TheoryUnit"/>
      <dc:language>german</dc:language>
      <dc:coverage>Introductory course</dc:coverage>
      <dc:rights>KBS (Universität Hannover)</dc:rights>
      <olr:parentUnit rdf:resource="#LectureUnit1"/>
</rdf:Description>
```

Table 2. RDF-Triples of one the multiple declarations about "http://.../Agents/-Definitions.htm"

Nr.	Subject	Predicate	Object
1	http://.../Agents/Definitions.htm	http://www.w3.org/1999/02/22-rdf-syntax-s#type	http://.../rdf/olr#TheoryUnit
2	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#title	Definitions
3	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#description	Definitions of the basics of A
4	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#subject	Definitions

The number of triples = 4

Table 3. RDF-Triples of another of the multiple declarations about "http://.../Agents/-Definitions.htm"

Nr.	Subject	Predicate	Object
1	http://.../Agents/Definitions.htm	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://.../rdf/olr#TheoryUnit
2	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#language	German
3	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#coverage	Introductory course
4	http://.../Agents/Definitions.htm	http://purl.org/dc/elements/1.0#rights	KBS (Universität Hannover)
5	http://.../Agents/Definitions.htm	http://.../rdf/olr_schema_5#parentUnit	online:#LectureUnit1

The number of triples = 5.

The above two tables Table 2 and Table 3 contain the RDF triples for the two separate RDF declarations of attributes to "http://.../Agents/Definitions.htm". By comparing the different triple sets that describe the example above we recognize that both declarations (compare Table 1 with the Tables 2 and 3) are identical which they have to be according to the RDF's specification. Looking at the example, we again realize the RDF property-centric approach, i.e. properties are the basic RDF constructs while classes etc. are just an add on to define *rdfs:domain* and *rdfs:range* constraints of these properties.

The advantage of the property-centric approach is that properties can be assigned to websites in a modular way. Furthermore it is semantically unimportant whether all properties are instantiated at once. As a result properties are always multi-valued , i.e. the expression <rdf:description about="…"> for a specific web page can be used repeatedly in an RDF file (possibly in several RDF files!)

A disadvantage of this modularity is of course that we cannot define single-valued attributes in RDF. For instance, it is not possible to define a property with a single value to represent the size of a resource. This, by the way, makes it difficult, if not impossible, to watch for violations of the single value property of *rdfs:range*. Several people can define different (in this case inconsistent) RDF-Statements for the size of the resource which leads to inconsistent information about the resource. In contrast, although attributes are basically multi-valued in O-Telos, too, they can be constrained to be single valued by O-Telos constraints.

Using the frame syntax of O-Telos, modularity like in RDF is not possible, as definitions and instances in O-Telos are class-centric and not property-centric. So, in O-Telos it is not possible to use e.g. "http://…/Agents/Definitions.htm" as ID for two instances. In order to declare several O-Telos objects about the same resource it is necessary to introduce an additional attribute "about" holding the URI of the resource, which however introduces an additional identifier which is not necessary in the tuple representation. Using this workaround, different O-Telos objects describing a resource have

their own IDs as required by the O-Telos axioms but can describe the same resource. A similar approach has to be used in XML Schema, by the way.

The previous RDF example of the resource "http://…/Agents/Definitions.htm" is declared in O-Telos by the following single frame:

```
Individual AgentDefinition1 in TheoryUnit with
  about
    a : "http://…/Agents/Definitions.htm"
  language
    l : "german"
  coverage
    c : "Introductory course"
  rights
    r : "KBS"
  parent_unit
    pu : LectureUnit1
end
```

In order to represent the above object *AgentDefinition1* by two frames an explicit *about*-attribute is used in the frames below. The instances *AgentDefinition1* and *AgentDefinition2* have different identifiers while they hold the same reference in their about-attribute to "http://…/Agents/Definitions.htm".

```
Individual AgentDefinition1 in TheoryUnit with
  about
    a : "http://…/Agents/Definitions.htm"
  language
    l : "german"
  coverage
    c : "Introductory course"
rights
    r : "KBS"
end
Individual AgentDefinition2 in TheoryUnit with
  about
    a : "http://…/Agents/Definitions.htm"
  parentUnit
    pu : LectureUnit1
end
```

Using this approach it is possible to declare various objects about the same resource in the same model. Because O-Telos does not have a feature like the namespace declaration of RDF it is not possible to declare objects about the same resource in different models.

2.5 Sequences and Reification in RDF and O-Telos

Let us look briefly at sequencing and reification in RDF and O-Telos. As an example we use the following RDF declaration of the resource *LectureUnit1* which we will translate to O-Telos. *LectureUnit1* defines a sequence for values of the *olr:theoryUnit* property. Its RDF declaration is given below:

```
<rdf:RDF xml:lang="en"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:olr="http://.../rdf/olr_schema_5#"
   xmlns:dc="http://purl.org/dc/elements/1.0#">
   <rdf:Description ID="LectureUnit1">
```

```
    <rdf:type resource="http://…/rdf/olr_schema_5#Unit"/>
    <dc:title>Lecture Unit 1</dc:title>
    <dc:description>Introduction to intelligent agents</dc:description>
    <olr:parentCourse rdf:resource="#AILecture"/>
    <olr:theoryUnit>
      <rdf:Seq>
        <rdf:li rdf:resource="http://.../Agents/Definitions.htm"/>
        <rdf:li rdf:resource="http://.../Agents/Characterisation.htm"/>
        <rdf:li rdf:resource="http://.../Agenten/Structur.htm"/>
        <rdf:li rdf:resource="http://.../Agenten/Types.htm"/>
      </rdf:Seq>
    </olr:theoryUnit>
  </rdf:Description>
</rdf:RDF>
```

In this example the order of resources of the property *olr:theoryUnit* is defined by the container object RDF sequence (rdf:Seq). This order is used for the visualisation of the course hierarchy. While it is a convenient way to represent sequences, it is conceptually questionable, as *rdf:seq* is used as range of *olr:theoryUnit*, instead of the more explicit ranges describing the specific type of the child resource (like *theoryUnit*, or, for other properties, *example*, *slide*, etc. which we use in OLR).

O-Telos does not define such a construct for stating sequences, but represents sequences implicitly by the order of attribute statements in the O-Telos frames. Of course it is not insured that each implementation of O-Telos interprets the frames in the same way so that the attribute order (the sequence) might vary from one implementation to another.

If we want to state our RDF example without using RDF sequence but still represent sequences, we could use an attribute *ordinal* for the RDF-statements representing the sequence of the property values. These statements then look like:

<oid ,ordinal,i>, with i:integer and oid:ID is the ID of a statement <s,p,o> with s:subject, p:predicate and o:object.

In other words we need the possibility to make statements about statements, e.g. by referring to the IDs of statements in statements. Unfortunately, RDF statements do not have IDs. Instead we have to introduce higher-order statements which are a special kind of statements about statements:

<s,p,o,t> with s:subject, p:predicate, o:object and t:type

Applied to our example this could be written as follows:

```
<olr:Unit rdf:ID="LectureUnit1"/>
<rdf:Description>
  <rdf:subject resource="#LectureUnit1" />
  <rdf:predicate resource="http://.../#theoryUnit" />
  <rdf:object rdf:resource="http://.../Agents/Definitions.htm"/>
  <rdf:type resource="http://.../22-rdf-syntax-ns#Statement"/>
  <olr:ordinalNo>1</olr:ordinalNo>
</rdf:Description>
<rdf:Description>
  <rdf:subject resource="#LectureUnit1" />
  <rdf:predicate resource="http://.../#theoryUnit " />
  <rdf:object rdf:resource="http://.../Agents/Characterisation.htm"/>
  <rdf:type resource="http://.../22-rdf-syntax-ns#Statement"/>
  <olr:ordinalNo>2</olr:ordinalNo>
</rdf:Description>
```

Of course the disadvantage is the lost simplicity of the model and a rather complex und unreadable declaration. In O-Telos, specifying properties for other properties can be handled more directly, as all property statements have their own unique identifier, and thus can be directly annotated with additional attributes like in

```
Attribute LectureUnit1!tu1 in CourseUnit!theoryUnit with
  ordinalNo
    o : 1
end
Attribute LectureUnit1!tu2 in CourseUnit!theoryUnit with
  ordinalNo
    o : 2
end
```

In [16] we show how to use this idea in an extended variant of RDF (O-Telos-RDF), which easily allows reifications of arbitrary statements by referencing statement IDs. Of course, introducing unique ids for property statements in RDF is not possible globally. Still, locally at one site, this is possible, and the site prefix can make these ids unique worldwide (which is the approach we propose in [16]).

2.6 Comparing RDF and O-Telos on the Tuple Level

As mentioned before RDF declarations can be represented as triples. A RDF triple has the definition:

> <s,p,o> with s:subject, p:predicate and o:object, reading: there is a property p from subject s to object o.

O-Telos declarations can be represented by quadruples which are called propositions. In general propositions represent relationships:

> P(oid,x,l,y) with oid:objectID, x:source, l:label, y:destination, reading: there exists an object with oid stating a relationship called l from object x to object y.

So, O-Telos propositions include an explicit ID, while RDF triples do not. Using this ID, instantiation of properties is handled differently (explicitly in O-Telos, and implicitly in RDF), which results in a marked difference in the meta-modelling capabilities of RDF (rather restricted) and O-Telos (unrestricted meta-modelling hierarchies possible). Usually, each RDF triple is expressed by two O-Telos propositions, where the instantiation of a property is an own statement in O-Telos, but is handled implicitly (by directly using the predicate name) in RDF. In general, O-Telos propositions, which have a unique id, are much better suited for reification than RDF triples.

The following RDF declarations define three properties for the resource “http://.../Agents/Characterisation.htm”. The *rdf:type* property defines the resource as of type *olr#TheoryUnit* while *dc:title* states the name of the resource and *olr:parentUnit* defines the resource *LectureUnit1* as *parentUnit*. The triple representation shows three triples corresponding to this declaration.

```
<rdf:Description about="http://.../Agents/Characterisation.htm">
  <rdf:type resource="http://.../rdf/olr#TheoryUnit"/>
  <dc:title>Characterisation of agents</dc:title>
  <olr:parentUnit rdf:resource="#LectureUnit1"/>
</rdf:Description>
```

Table 4. RDF-Triples of the declaration about "http://.../Agents/Characterisation.htm"

Nr.	Subject	Predicate	Object
1	http://.../Agents/Characterisation.htm	http://…/22-rdf-syntax-ns#type	http://.../rdf/olr#TheoryUnit
2	http://.../Agents/Characterisation.htm	http://purl.org/dc/elements/1.0#title	Characterisation of agents
3	http://.../Agents/Characterisation.htm	http://…/rdf/olr_schema_7#parentUnit	online:#LectureUnit1

Table 4 states these three RDF triples. They show explicitly that all three properties belong to the resource, and the predicates (second argument) directly state name and the accompanying namespace of the properties.

The O-Telos frame declares the object "http://.../Agents/Characterisation.htm" as instance of (the keyword "in" in the frame) class *TheoryUnit* similarly to the *rdf:type* property of the RDF declaration. The other two attributes *dc_title* and *parentUnit* correspond to the respective properties.

```
Individual "http://.../Agents/Characterisation.htm" in TheoryUnit with
  dc_title
    t1 : "Characterisation of agents"
  parentUnit
    pu1 : LectureUnit1
end
```

However, the O-Telos propositions show more detail than the corresponding RDF triples:

Table 5. O-Telos propositions of the declaration of "http://.../Agenten/Characterisation.htm"

oid	source	Label	destination
#1	#1	"http://.../Agents/Characterisation.htm"	#1
#2	#1	*instanceof	#TheoryUnit
#3	#1	T1	"Characterisation of agents"
#4	#3	*instanceof	#dc_title
#5	#1	pu1	#LectureUnit1
#6	#5	*instanceof	#parentUnit

Table 5 shows the O-Telos propositions of our example. Proposition *#1* explicitly represents the object "http://…//Agents/Characterisation.htm" while proposition *#2* states that this object is instance of class *TheoryUnit*. Proposition *#3* declares that the object from *#1* has an attribute *t1* with value "Characterisation of agents" while proposition *#4* declares the attribute *t1* from *#3* as instance of *#dc_title*. Proposition *#5* declares that the object from *#1* has an attribute *pu1* as a reference to *#LectureUnit1* while proposition *#6* declares the attribute from *#5* as instance of *#parentUnit*. O-Telos also requires that the declaration of the attributes *t1* and *pu1* is included in the class *TheoryUnit* from which this object is an instance.

We have no direct possibility to represent RDF namespace information in our O-Telos propositions, as O-Telos relies on the declaration of schema and metadata in one file. In [16] however we specify statement IDs for O-Telos-RDF (which are invisible in O-Telos), that include namespace information in a way similar to RDF/RDFS.

3 Conclusion and future work

This paper discussed the use of RDF metadata in our open learning repository system OLR, as well as its underlying architecture. We are currently extending this system by different navigation schemes and are working on making it still easier to modify/extend metadata and metadata schemas in/with OLR. To support LOM metadata annotation of a large amount of (often hierarchically related) document pages, we will have to add some inferencing capabilities which for example allow (default) inheritance of LOM attributes along the LOM *isPartOf* relation. An further extension will be P2P exchange functionality between distributed OLR systems.

In the second part of this paper we have compared RDF/RDFS with the conceptual modelling language O-Telos and discussed some mappings, which hopefully shed some light on the advantages and disadvantages of RDFS design decisions. We have continued this work in another report, which defines an RDF-variant called O-Telos-RDF with extended reification and meta-modelling capabilities. Further interesting work includes a comparison of the O-Telos query language (as implemented in Conceptbase) for RDF and O-Telos-RDF.

This work has profited much from several discussions with our colleagues, and we want to thank especially Changtao Qu for his comments on several of the issues discussed in this paper.

References

[1] RDF Model and Syntax Specification, World Wide Web Consortium (W3C), February 1999, http://www.w3.org/TR/REC-rdf-syntax/

[2] RDF Schema Specification, World Wide Web Consortium (W3C), March 2000, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/

[3] Dublin Core Initiative, 2001, http://dublincore.org/

[4] Karsten Tolle, Analyzing and Parsing RDF, Master's Thesis, Institute of Computer Engineering - University of Hannover in cooperation with the Institute of Computer Science - Foundation of Research Technology Hellas - Greece (ICS-FORTH), 2001, http://www.kbs.uni-hannover.de/Arbeiten/Diplomarbeiten-/00/tolle/AuPRDF.pdf

[5] Semantic Web Activity Statement, World Wide We Consortium (W3C), 2001, http://www.w3.org/2001/sw/Activity

[6] Sergey Melnik, Storing RDF in a relational database, 2000, http://www-db.stanford.edu/~melnik/rdf/db.html

[7] Nejdl, M. Wolpers, C.Capelle, The RDF Schema Specification Revisited, Modellierung 2000, 5. - 7.4.2000, St Goar, Germany, http://www.kbs.uni hannover.de/Arbeiten/Publikationen/2000/modeling2000/-wolpers.pdf

[8] Wolfgang Nejdl and Martin Wolpers: KBS Hyperbook - A Data-Driven Information System on the Web. *WWW8 Conference*, Toronto, May 1999, http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1999/-www8/index.html

[9] Wolfgang Nejdl and Nicola Henze: Adaptivity in the KBS Hyperbook System. 2nd Workshop on User Modeling and Adaptive Systems on the WWW, May 1999, Toronto, Canada, http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1999/Henze.html

[10] M. Jeusfeld, Änderungskontrolle in deduktiven Objektbanken, Infix-Verlag 1992, St. Augustin, Germany

[11] Johan Gamper, Wolfgang Nejdl and Martin Wolpers: Combining Ontologies and Terminologies in Information Systems. 5th International Congress on Terminology and Knowledge Engineering, Innsbruck, Austria, August 1999, http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1999/tke99/index.html

[12] M. A. Jeusfeld, M. Jarke, H. W. Nissen and M. Staudt, ConceptBase - Managing Conceptual Models about Information Systems, in Handbook on Architectures of Informations Systems, P. Bernus, K. Mertins and G. Schmidt (eds.), Springer Verlag 1998

[13] M. Ashrafuzzaman: Deductive Object-Oriented Database for Geographic Data Handling. Course project CMPT826, University of Saskatchewan, Canada, March 1996

[14] Janne Saarela, SiRPAC - Simple RDF Parser & Compiler World Wide Web Consortium (W3C), 2001, http://www.w3.org/RDF/Implementations/SiRPAC/

[15] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, Telos: A langugage for representing knowledge about information systems, ACM Transaction on Information Systems, 8.4, 1990

[16] W. Nejdl, H. Dhraief and M. Wolpers, O-Telos-RDF: A Resource Description Format with Enhanced Meta-Modelling Functionalities based on O-Telos, Technical Report, Inst. f. Technische Informatik, Uni. Hannover, Germany

CREAM — Creating relational metadata with a component-based, ontology-driven annotation framework

[1]Siegfried Handschuh, [1,2]Steffen Staab, [1,3]Alexander Maedche

[1]Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany
http://www.aifb.uni-karlsruhe.de/WBS
{sha,sst,ama}@aifb.uni-karlsruhe.de

[2]Ontoprise GmbH, Haid-und-Neu Straße 7, 76131 Karlsruhe, Germany
http://www.ontoprise.de

[3]FZI Research Center for Information Technologies,
Haid-und-Neu Straße 10-14, 76131 Karlsruhe, Germany
http://www.fzi.de/wim

"The Web is about links;
the Semantic Web is about the relationships implicit in those links."
Dan Brickley

Abstract. Richly interlinked, machine-understandable data constitutes the basis for the Semantic Web. Annotating web documents is one of the major techniques for creating metadata on the Web. However, annotation tools so far are restricted in their capabilities of providing richly interlinked and truely machine-understandable data. They basically allow the user to annotate with plain text according to a template structure, such as Dublin Core. We here present CREAM (Creating RElational, Annotation-based Metadata), a framework for an annotation environment that allows to construct *relational metadata*, i.e. metadata that comprises class instances and relationship instances. These instances are not based on a fix structure, but on a domain ontology. We discuss some of the requirements one has to meet when developing such a framework, e.g. the integration of a metadata crawler, inference services, document management and information extraction, and describe its implementation, *viz.* Ont-O-Mat a component-based, ontology-driven annotation tool.

1 Introduction

Research about the WWW currently strives to augment syntactic information already present in the Web by semantic metadata in order to achieve a Semantic Web that human and software agents alike can understand. RDF(S) or DAML+OIL are languages that have recently advanced the basis for extending purely syntactic information, *e.g.* HTML documents, with

semantics. Based on these recent advancements one of the the most urgent challenges now is a knowledge capturing problem, *viz.* how one may turn existing syntactic resources into interlinked knowledge structures that represent relevant underlying information. This paper is about a framework for facing this challenge, called CREAM[1], and about its implementation, Ont-O-Mat.

The origin of our work facing this challenge dates back to the start of the seminal KA2 intiative [1], *i.e.* the initiative for providing semantic markup on HTML pages for the knowledge acquisition community. The basic idea then was that manual knowledge markup on web pages was too error-prone and should therefore be replaced by a *simple* tool that should help to avoid syntactic mistakes.

Developing our CREAM framework, however, we had to recognize that this knowledge capturing task exhibited some intrinsic difficulties that could not be solved by a *simple* tool. We here mention only some challenges that immediately came up in the KA2 setting:

- **Consistency**: Semantic structures should adhere to a given ontology in order to allow for better sharing of knowledge. For example, it should be avoided that people confuse complex instances with attribute types.

- **Proper Reference**: Identifiers of instances, *e.g.* of persons, institutes or companies, should be unique. For instance, in KA2 metadata there existed three different identifiers of our colleague Dieter Fensel. Thus, knowledge about him could not be grasped with a straightforward query.[2]

- **Avoid Redundancy**: Decentralized knowledge provisioning should be possible. However, when annotators collaborate, it should be possible for them to identify (parts of) sources that have already been annotated and to reuse previously captured knowledge in order to avoid laborious redundant annotations.

- **Relational Metadata**: Like HTML information, which is spread on the Web, but related by HTML links, knowledge markup may be distributed, but it should be semantically related. Current annotation tools tend to generate template-like metadata, which is hardly connected, if at all. For example, annotation environments often support Dublin Core [12], providing means to state, e.g., the name of authors, but not their IDs[3].

- **Maintenance**: Knowledge markup needs to be maintained. An annotation tool should support the maintenance task.

- **Ease of use**: It is obvious for an annotation environments to be useful. However, it is not trivial, because it involves intricate navigation of semantic structures.

- **Efficiency**: The effort for the production of metadata is a large restraining threshold. The more efficiently a tool support the annotation, the more metadata will produce a user. These requirement stand in relationship with the ease of use. It depends also on the automation of the annotation process, e.g. on the pre-processing of the document.

[1] CREAM: Creating RElational, Annotation-based Metadata.

[2] The reader may see similar effects in bibliography databases. E.g., query for James (Jim) Hendler at the — otherwise excellent — DBLP: `http://www.informatik.uni-trier.de/~ley/db/`.

[3] In the web context one typically uses the term 'URI' (uniform resource identifier) to speak of 'unique identifier'.

CREAM faces these principal problems by combining advanced mechanisms for inferencing, fact crawling, document management and — in the future — information extraction. Ont-O-Mat, the implementation of CREAM, is a component-based plug-in architecture that tackles this broad set of requirements.[4]

In the following we first sketch two usage scenarios (Section 2). Then, we explain our terminology in more detail, derive requirements from our principal considerations above and explain the architecture of CREAM (Section 3). We describe our actual tool, Ont-O-Mat, in Section 4. Before we conclude, we contrast CREAM with related work, namely knowledge acquisition tools and annotation frameworks.

2 Scenarios for CREAM

We here only summarize two scenarios, two knowledge portals, for annotation that have been elaborated in [21]:

The first scenario extends the objectives of the seminal KA2 initiative. The KA2 portal provides a view onto knowledge of the knowledge acquisition community. Besides of semantic retrieval as provided by the original KA2 initiative, it allows comprehensive means for navigating and querying the knowledge base and also includes guidelines for building such a knowledge portal. The potential users provide knowledge, e.g. by annotating their web pages in a decentralized manner. The knowledge is collected at the portal by crawling and presented in a variety of ways.

The second scenario is a knowledge portal for business analysts that is currently constructed at Ontoprise GmbH. The principal idea is that business analyst review news tickers, business plans and business reports. A considerable part of their work requires the comparison and aggregation of similar or related data, which may be done by semantic queries like“Which companies provide B2B solutions?”, when the knowledge is semantically available. At the Time2Research portal they will handle different types of documents, annotate them and, thus, feed back into the portal to which they may ask questions.

3 Design of CREAM

In this section we explain basic design decisions of CREAM, which are founded on the general problems sketched in the introduction above. In order to provide a clear design rationale, we first provide definitions of important terms we use subsequently:

- **Ontology**: An ontology is a formal, explicit specification of a shared conceptualization of a domain of interest [8]. In our case it is constituted by statements expressing definitions of DAML+OIL classes and properties [7].

- **Annotations**: An annotation in our context is a set of instantiations attached to an HTML document. We distinguish *(i)* instantiations of DAML+OIL classes, *(ii)* instantiated properties from one class instance to a datatype instance — henceforth called attribute instance (of the class instance), and *(iii)* instantiated properties from one class instance to another class instance — henceforth called relationship instance.

[4]The core Ont-O-Mat can be downloaded from:
`http://ontobroker.semanticweb.org/annotation`.

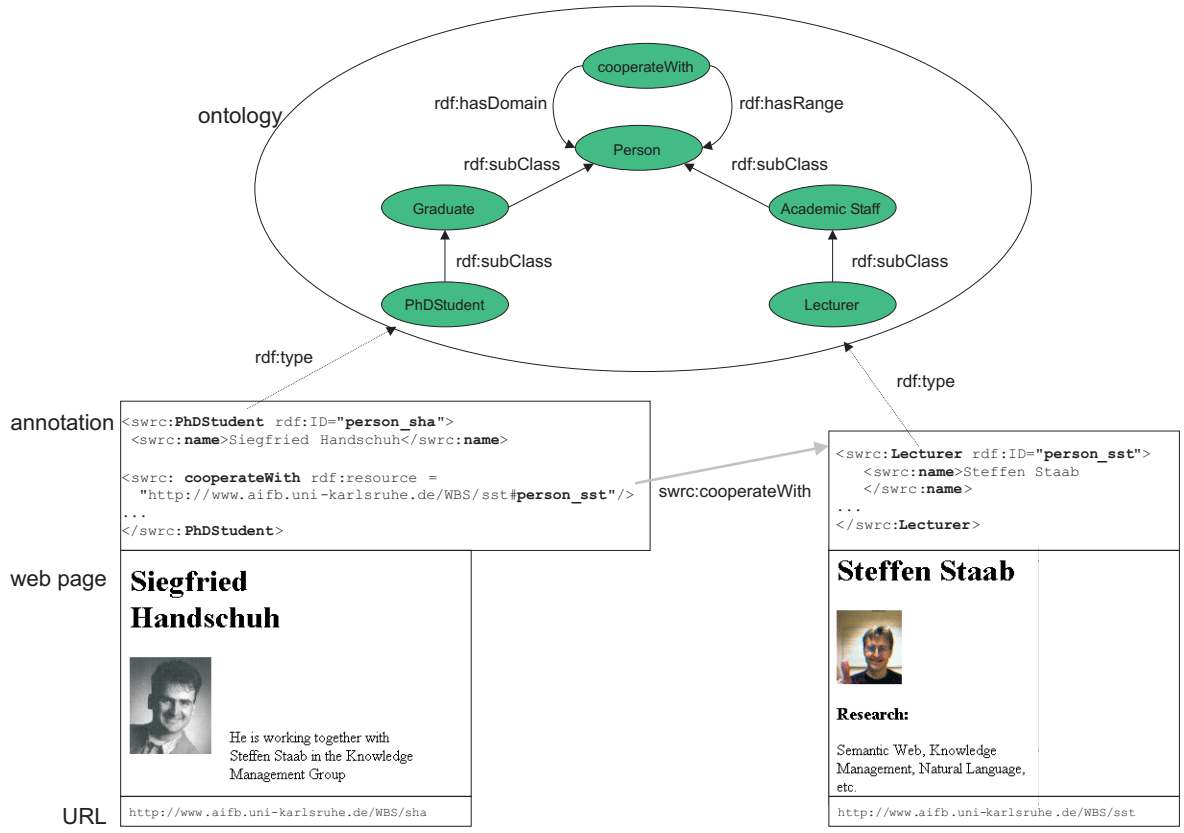


Figure 1: Annotation example

Class instances have unique URIs. Instantiations may be attached to particular markups in the HTML documents, *viz.* URIs and attribute values may appear as strings in the HTML text.

- **Metadata**: Metadata are data about data. In our context the annotations are metadata about the HTML documents.

- **Relational Metadata**: We use the term relational metadata to denote the annotations that contain relationship instances.

  Often, the term "annotation" is used to mean something like "private or shared note", "comment" or "Dublin Core metadata". This alternative meaning of annotation may be emulated in our approach by modeling these notes with attribute instances. For instance, a comment note "I like this paper" would be related to the URL of the paper via an attribute instance 'hasComment'.

  In contrast, relational metadata contain statements like 'student Siegfried cooperates with lecturer Steffen', *i.e.* relational metadata contain relationships between class instances rather than only textual notes.

Figure 1 illustrates our use of the terms "ontology", "annotation" and "relational metadata". It depicts some part of the SWRC[5] (semantic web research community) ontology. Fur-

[5] `http://ontobroker.semanticweb.org/ontos/swrc.html`

Table 1: Design Rationale — Linking Challenges with Required Modules

Requirement / General Problem	Document Viewer	Ontology Guidance	Replication: Crawler	Storage: Annotation Inference Server	Storage: Document Management	Information Extraction
Consistency		X		X		
Proper Reference			X	X		
Avoid Redundancy			X	X	X	
Relational Metadata		X	X	X		
Maintenance					X	X
Ease of use	X	X				X
Efficiency	X	X	X	X	X	X

thermore it shows two homepages, viz. pages about Siegfried and Steffen (`http://www.aifb.uni-karlsruhe.de/WBS/sha` and `http://www.aifb.uni-karlsruhe.de/WBS/sst`, respectively) with annotations given in an XML serialization of RDF facts. For the two persons there are instances denoted by corresponding URIs (`person_sha` and `person_sst`). The `swrc:name` of `person_sha` is "Siegfried Handschuh". In Addition, there is a relationship instance between the two persons: they cooperate. This cooperation information 'spans' the two pages.

3.1 Requirements for CREAM

The difficulties sketched in the introduction directly feed into the design rationale of CREAM. The design rationale links the challenges with the requirements. This results in a N:M mapping (neither functional nor injective). An overview of the matrix is given in Table 1. It shows which modules (requirements) are mainly used to answer challenges set forth in the introduction, *viz.*:

- **Document Viewer**: The document viewer visualizes the web page contents. The annotator may easily provide annotations by highlighting text that serves as input for attribute instances or the definition of URIs. The document viewer must support various formats (HTML, PDF, XML, etc.).
- **Ontology Guidance**: The annotation framework needs guidance from the ontology. In order to allow for sharing of knowledge, newly created annotations must be consistent with a community's ontology. If annotators instantiate arbitrary classes and properties the semantics of these properties remains void. Of course the framework must be able to adapt to varying ontologies in order to reflect different foci of the annotators.

  Furthermore, the ontology is important in order to guide annotators towards creating relational metadata. We have done some preliminary experiments and found that subjects have more problems with creating relationship instances than with creating attribute instances (cf. [22]). Without the ontology they would miss even more cues for assigning relationships between class instances.

Both ontology guidance and document viewer should be easy to use: Drag'n'drop helps to avoid syntax errors and typos and a good visualization of the ontology can help to correctly choose the most appropriate class for instances.

- **Crawler:** The creation of relational metadata must take place *within* the Semantic Web. During annotation annotaters must be aware of which entities exist in the part of the Semantic Web they annotate. This is only possible if a crawler makes relevant entities immediately available. So, annotators may look for proper reference, i.e. decide whether an entity already has a URI (e.g. whether the entity named "Dieter Fensel" or "D. Fensel" has already been identified by some other annotators) and thus only annotators may recognize whether properties have already been instantiated (e.g. whether "Dieter Fensel" has already be linked to his publications). As a consequence of annotators' awareness relational metadata may be created, because class instances become related rather than only flat templates are filled.

- **Annotation Inference Server**: Relational metadata, proper reference and avoidance of redundant annotation require querying for instances, i.e. querying whether and which instances exist. For this purpose as well as for checking of consistency, we provide an annotation inference server in our framework. The annotation inference server reasons on crawled and newly annotated instances and on the ontology. It also serves the ontological guidance, because it allows to query for existing classes and properties.

- **Document Management**: In order to avoid redundancy of annotation efforts, it is not sufficient to ask whether instances exist at the annotation inference server. When an annotator decides to capture knowledge from a web page, he does not want to query for all single instances that he considers relevant on this page, but he wants information, whether and how this web page has been annotated before. Considering the dynamics of HTML pages on the web, it is desirable to store annotated web pages together with their annotations. When the web page changes, the old annotations may still be valid or they may become invalid. The annotator must decide based on the old annotations and based on the changes of the web page.

  A future goal of the document management in our framework will be the semi-automatic maintenance of annotations. When only few parts of a document change, pattern matching may propose revision of old annotations.

- **Information Extraction**: Even with sophisticated tools it is laborious to provide semantic annotations. A major goal thus is semi-automatic annotation taking advantage of information extraction techniques to propose annotations to annotators and, thus, to facilitate the annotation task. Concerning our environment we envisage two major techniques: First, "wrappers" may be learned from given markup in order to automatically annotate similarly structured pages (cf., e.g., [16]). Second, message extraction like systems may be used to recognize named entities, propose co-reference, and extract some relationship from texts (cf., e.g., [20]).

Besides of the requirements that constitute single modules, one may identify functions that cross module boundaries:

- Storage: CREAM supports two different ways of storage. The annotations will be stored inside the document that is in the document management component, but it is also stored in the annotation inference server.
- Replication: We provide a simple replication mechanism by crawling annotations into our annotation inference server.

3.2 Architecture of CREAM

The architecture of CREAM is depicted in Figure 2. The complete design of CREAM comprises a plug-in structure, which is flexible with regard to adding or replacing modules. Document viewer and ontology guidance module together constitute the major part of the graphical user interface. Via plug-ins the core annotation tool, Ont-O-Mat, is extended to include the capabilities outlined above. For instance, a plug-in for a connection to a document management system provides document management and retrieval capabilities that show the user annotations of a document he loads into his browser. This feature even becomes active when the user does not actively search for already existing annotations. Similarly, Ont-O-Mat provides extremely simple means for navigating the taxonomy, which means that the user can work without an inference server. However, he only gets the full-fledged semantics when the corresponding plug-in connection to the annotation inference server is installed.

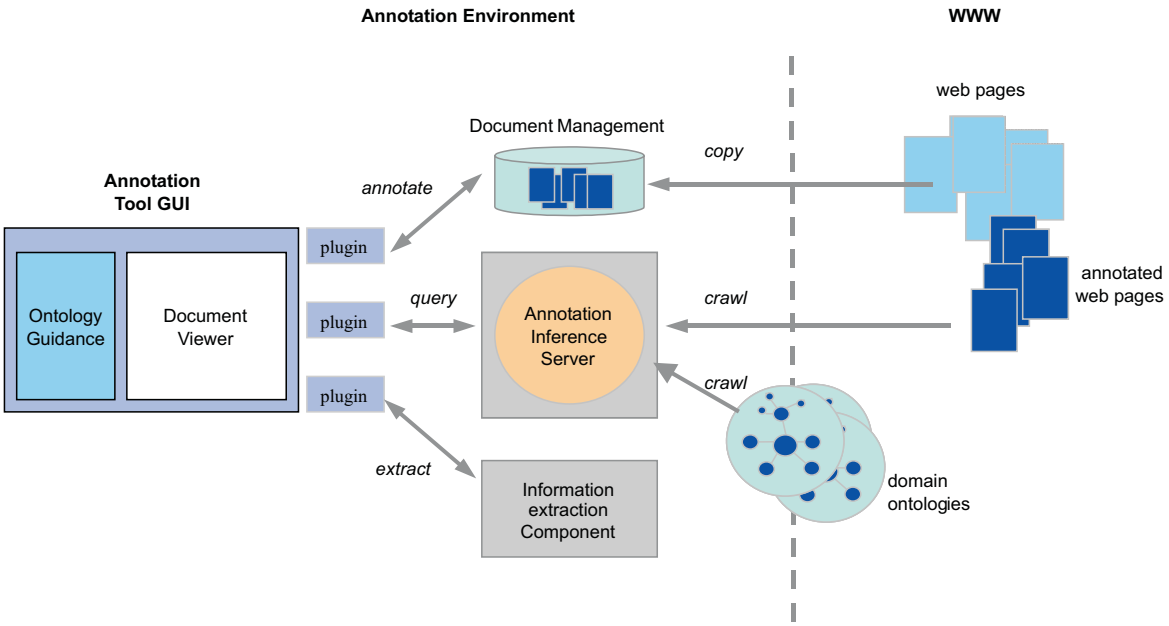


Figure 2: Architecture of CREAM.

4 Implementation: Ont-O-Mat

This section describes Ont-O-Mat, the implementation of our CREAM framework. Ont-O-Mat is a component-based, ontology-driven markup tool. The architectural idea behind CREAM is a component-based framework, thus, being open, flexible and easily extensible.

In the following subsection we refer to the concrete realization and the particular technical requirements of the components. In subsection 4.2 we describe the functionality of Ont-O-Mat based on an example ontology for annotation that is freely available on the web.

4.1 Ont-O-Mat services and components

The architecture of Ont-O-Mat provides a plug-in and service mechanism. The components are dynamically plug-able to the core Ont-O-Mat. The plug-in mechanism notifies each installed component, when a new component is registered. Through the service mechanism each component can discover and utilize the services offered by another component [9]. A service represented by a component is typically a reference to an interface. This provides among other things a de-coupling of the service from the implementation and allows therefore alternative implementations.

The Ont-O-Mat services have been realized by components according to the requirements listed in subsection 3.1. So far we have realized the following components: a comprehensive user-interface, component for document-management, an annotation inference-server and a crawler:

- **Document Viewer and Ontology Guidance**: There are various ways how the gained knowledge database can be visualized and thus experienced. On the one hand, the system can be used as a browser. In the annotated web pages, the extracted text fragments are then highlighted and an icon after each fragment is visible. By clicking on the icon, the name of the assigned class or attribute will be shown. On the other hand, the user can browse the ontology and retrieve for one class all instances or for one instance all attributes.

  The underlying data model used for Ont-O-Mat has been taken from the comprehensive ontology engineering and learning system OntoEdit / Text-To-Onto (see [18]).

  Ont-O-Mat works currently in "read-only–mode" with respect to the ontology and only operates on the relational metadata defined on top of the given ontology.

- **Document Management**: A component for document management is required in order to avoid duplicate annotations and existing semantic annotations of documents should be recognized. In our current implementation we use a straight forward file-system based document management approach.

  Ont-O-Mat uses the URI to detect the re-encounter of previously annotated documents and highlights annotations in the old document for the user. Then the user may decide to ignore or even delete the old annotations and create new metadata, he may augment existing data, or he may just be satisfied with what has been previously annotated. In order to recognize that a document has been annotated before, but now appears under a different URI, Ont-O-Mat computes similarity with existing documents by simple information retrieval methods, e.g. comparison of the word vector of a page. If thereby a similarity is discovered, this is indicated to the user, so that he can check for congruency.

- **Annotation Inference Server**: The annotation inference server reasons on crawled and newly annotated instances and on the ontology. It also serves the ontological guidance, because it allows to query for existing classes and properties. We use Ontobroker's [3] underlying F-Logic [14] based inference engine SilRI [2] as annotation inference server.

The F-Logic inference engine combines ordering-independent reasoning in a high-level logical language with a well-founded semantics.

- **RDF Crawler**: As already mentioned above, the annotation must take place right within the Semantic Web and not isolated. Therefore, we have built a RDF Crawler[6], a basic tool that gathers interconnected fragments of RDF from the Web and builds a local knowledge base from this data.

  In general, RDF data may appear in Web documents in several ways. We distinguish between *(i)* pure RDF (files that have an extension like "*.rdf"), *(ii)* RDF embedded in HTML and *(iii)* RDF embedded in XML. Our RDF Crawler relys on Melnik's RDF-API[7] that can deal with the different embeddings of RDF described above. One general problem of crawling is the applied filtering mechanism: Baseline document crawlers are typically restricted by a predefined depth value. Assuming that there is an unlimited amount of interrelated information on the Web (hopefully this will soon hold about RDF data as well), at some point RDF fact gathering by the RDF Crawler should stop. We have implemented a baseline approach for filtering: At the very start of the crawling process and at every subsequent step we maintain a queue of all the URIs we want to analyze. We process them in the breadth-first-search fashion, keeping track of those we have already visited. When the search goes too deep, or we have received sufficient quantity of data (measured as number of links visited or the total web traffic or the amount of RDF data obtained) we may quit.

- **Information Extraction**: This component has not yet been integrated in our Ont-O-Mat tool. Actually, we are near finishing an integration of a simple wrapper approach [15], but we have not yet the message extraction approach for Ont-O-Mat that suggests relevant part of the texts for annotation.

4.2 Using Ont-O-Mat — An Example

Our example is based on the freely available SWRC (**S**emantic **W**eb **R**esearch **C**ommunity)[8] ontology , the successor of the KA2 ontology. The SWRC ontology models the semantic web research community, its researchers, topics, publications, tools, etc. and properties between them. It is available in the form of DAML+OIL classes and properties, in pure RDF-Schema and in F-Logic. The general idea behind SWRC is that the SW research community creates relational metadata according to the SWRC ontology to enable semantic access to their web pages. In the following we shortly explain how Ont-O-Mat may be used for creating relational metadata based on the SWRC ontology.

The annotation process is started either with an annotation inference server or the server process is fed with metadata crawled from the web and the document server. Figure 3 shows the screen for navigating the ontology and creating annotations in Ont-O-Mat. The right pane displays the document and the left panes show the ontological structures contained in the ontology, namely classes, attributes and relations. In addition, the left pane shows the cur-

[6] RDF Crawler is freely available for download at:
`http://ontobroker.semanticweb.org/rdfcrawler`.

[7] http://www-db.stanford.edu/∼melnik/rdf/api.html

[8] http://www.semanticweb.org/ontologies/

rent semantic annotation knowledge base, i.e. existing class instances, attribute instances and relationship instances created during the semantic annotation.

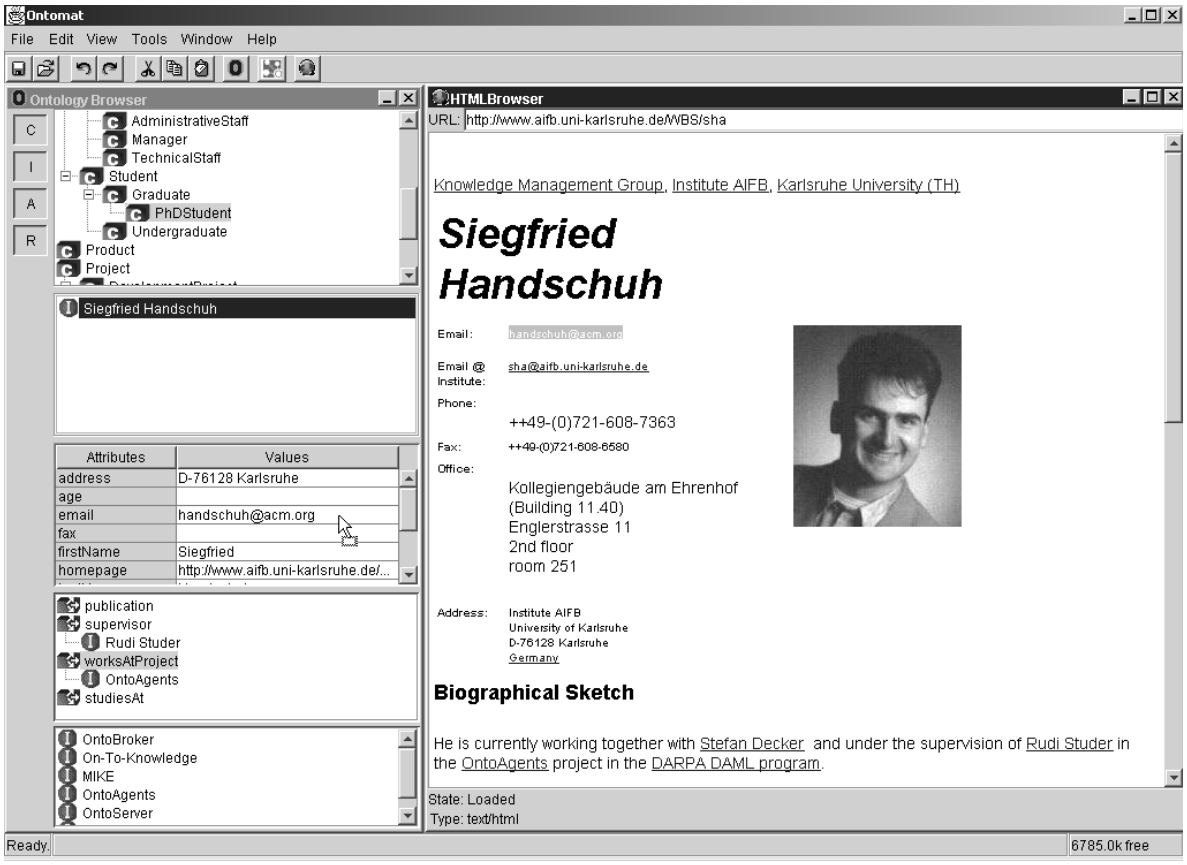


Figure 3: Ont-O-Mat Screenshot.

1. First of all, the user browses a document by entering the URL of the web document that he would like to annotate. This step is quite familiar from existing browsers.

2. Then the user selects a text fragment by highlighting it and takes a look on the ontology which fits in the topic and is therefore loaded and visible in ontology browser.

3. There are two possibilities for the text fragment to be annotated: as an instance or as a property. In the case of an instance, the user selects in the ontology the class where the text fragment fits in, e.g. if he has the text fragment ”Siegfried Handschuh”, he would select the class ”PhD Student”. By clicking on the class, the annotation gets created and thus the text fragment will be shown as an instance of the selected class in the ontology at the ontology browser.

4. To each created instance, literal attributes can be assigned. The choice of the predefined attributes depends on the class the instance belongs to, e.g. the class ”PhD Student” has the attributes name, address, email, and telephone number. The attributes can be assigned to the instance by highlighting the appropriate text fragment of the web document and dragging it to the related property field.

5. Furthermore, the relationships between the created instances can be set, e.g. the PhD Student Siegfried Handschuh ”works at” the OntoAgent project and ”is supervised” by Rudi Studer. Ont-O-Mat preselects class instances according to the range restrictions of the chosen relation, e.g. the ”works at” of a PhD Student must be an Project. Therefore only Projects are offered as potential fillers to the ”works at” relation of Siegfried.

5 Comparison with Related Work

CREAM can be compared along three dimensions: First, it is a framework for mark-up in the Semantic Web. Second, it can be considered as a particular knowledge acquisition framework vaguely similar to Protégé-2000[6]. Third, it is certainly an annotation framework, though with a different focus than ones like Annotea [13].

5.1 Knowledge Markup in the Semantic Web

We know of three major systems that intensively use knowledge markup in the Semantic Web, viz. SHOE [10], Ontobroker [3] and WebKB [19]. All three of them rely on knowledge in HTML pages.

They all started with providing manual mark-up by editors. However, our experiences (cf. [5]) have shown that text-editing knowledge mark-up yields extremely poor results, viz. syntactic mistakes, improper references, and all the problems sketched in the introduction.

The approaches from this line of research that are closest to *CREAM* is the *SHOE Knowledge Annotator*[9].

The SHOE Knowledge Annotator is a Java program that allows users to mark-up webpages with the SHOE ontology. The SHOE system [17] defines additional tags that can be embedded in the body of HTML pages. The Knowledge Annotater is less user friendly compared with our implementation Ont-O-Mat. It shows the ontology in some textual lists, whereas Ont-O-Mat gives a graphical visualization of the ontologies. Furthermore, in SHOE there is no direct relationship between the new tags and the original text of the page, i.e. SHOE tags are not annotations in a strict sense.

5.2 Comparison with Knowledge Acquisition Frameworks

The CREAM framework is specialized for creating class and property instances and for populating HTML pages with them. Thus, it does not function as an ontology editor, but rather like the instance acquisition phase in the Protégé-2000 framework [6]. The obvious difference of CREAM to the latter is that Protege does not (and does not intend to) support the particular web setting, *viz.* managing and displaying web pages.

5.3 Comparison with Annotation Frameworks

There are a lot of — even commercial — annotation tools like ThirdVoice[10], Yawas [4], CritLink [23] and Annotea (Amaya) [13].

[9] http://www.cs.umd.edu/projects/plus/SHOE/KnowledgeAnnotator.html

[10] http://www.thirdvoice.com

These tools all share the idea of creating a kind of user comment on the web pages. The term “annotation” in these frameworks is understood as a remark to an existing document. As mentioned before, we would model such remarks as attribute instances only in our framework. For instance, a user of these tools might attach a note like ”A really nice professor!” to the name “Studer” on a web page.

Annotea actually goes one step further. It allows to rely on an RDF schema as a kind of template that is filled by the annotator. For instance, Annotea users may use a schema for Dublin Core and fill the author-slot of a particular document with a name. This annotation, however, is again restricted to attribute instances. The user may also decide to use complex RDF descriptions instead of simple strings for filling such a template. However, he then has no further support from Amaya that helps him providing syntactically correct statements with proper references.

To summarize, CREAM is used to generate really machine-understandable data and addresses all the problems that come from this objective: relational metadata, proper reference and consistency.

6 Conclusion and Future Plans

CREAM is a comprehensive framework for creating annotations, relational metadata in particular — the foundation of the future Semantic Web. The framework comprises inference services, crawler, document management system, ontology guidance, and document viewers.

Ont-O-Mat is the reference implementation of CREAM framework. The implementation supports so far the user with the task of creating and maintaining ontology-based DAML+OIL markups, i.e. creating of class, attribute and relationship instances. Ont-O-Mat include an ontology browser for the exploration of the ontology and instances and a HTML browser that will display the annotated parts of the text. Ont-O-Mat is Java-based and provides a plugin interface for extensions for further advancement.

Our goal is a constant advancement of Ont-O-Mat and the CREAM framework in order to answer basic problems that come with semantic annotation.

We are already dealing with many different issues and through our practical experiences we could identify problems that are most relevant in our scenario/settings, KA2 and Time2Research. Nevertheless our analysis of the general problem is far from being complete. Some further important issues we want to mention here are:

- **Information Extraction**: We have done some first steps to incorporate information extraction. However, our future experiences will have to show how and how well information extraction integrates with semantic annotation.
- **Multimedia Annotation**: This requires considerations about time, space and synchronization.
- **Changing Ontologies**: Ontologies on the web have characteristics that influence the annotation process. Heflin & Hendler [11] have elaborated on changes that affect annotation. Future annotation tools will have to incorporate solutions for the difficulties they consider.
- **Active Ontology Evolvement**: Annotation should feed back into the actual ontologies, because annotators may find that they should consider new knowledge, but need revised

ontologies for this purpose. Thus, annotation affects ontology engineering and ontology learning.

Our general conclusion is that providing semantic annotation, relational metadata in particular, is an important complex task that needs comprehensive support. Our framework CREAM and our tool Ont-O-Mat have already proved very successful in leveraging the annotation process. They still need further refinement, but they are unique in their design and implementation.

7 Acknowledgements.

The research presented in this paper would not have been possible without our colleagues and students at the Institute AIFB, University of Karlsruhe, and Ontoprise GmbH. We thank Kalvis Apsitis (now: RITI Riga Information Technology Institute), Stefan Decker (now: Stanford University), Michael Erdmann, Mika Maier-Collin, Leo Meyer and Tanja Sollazzo. Research for this paper was partially financed by US Air Force in the DARPA DAML project "OntoAgents" (01IN901C0).

References

[1] R. Benjamins, D. Fensel, and S. Decker. KA2: Building Ontologies for the Internet: A Midterm Report. *International Journal of Human Computer Studies*, 51(3):687, 1999.

[2] S. Decker, D. Brickley, J. Saarela, and J. Angele. A Query and Inference Service for RDF. In *Proceedings of the W3C Query Language Workshop (QL-98), http://www.w3.org/TandS/QL/QL98/, Boston, MA, December 3-4*, 1998.

[3] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer Academic Publisher, 1999.

[4] L. Denoue and L. Vignollet. An annotation tool for web browsers and its applications to information retrieval. In *In Proceedings of RIAO2000*, Paris, April 2000. http://www.univ-savoie.fr/labos/syscom/Laurent.Denoue/riao2000.doc.

[5] M. Erdmann, A. Maedche, H.-P. Schnurr, and Steffen Staab. From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *P. Buitelaar & K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg, August 2000.

[6] H. Eriksson, R. Fergerson, Y. Shahar, and M. Musen. Automatic generation of ontology editors. In *Proceedings of the 12th Banff Knowledge Acquisition Workshop, Banff, Alberta, Canada*, 1999.

[7] Reference description of the daml+oil (march 2001) ontology markup language. http://www.daml.org/2001/03/reference.html, March 2001.

[8] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 6(2):199–221, 1993.

[9] Siegfried Handschuh. Ontoplugins – a flexible component framework. Technical report, University of Karlsruhe, May 2001.

[10] J. Heflin and J. Hendler. Searching the web with shoe. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01*, pages 35–40. AAAI Press, 2000.

[11] J. Heflin, J. Hendler, and S. Luke. Applying Ontology to the Web: A Case Study. In *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks, IWANN'99*, 1999.

[12] Dublin Core Metadata Initiative. http://purl.oclc.org/dc/, April 2001.

[13] J. Kahan, M. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In *Proc. of the WWW10 International Conference. Hong Kong*, 2001.

[14] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[15] J. Klotzbuecher. Ontowrapper. Master's thesis, University of Karlsruhe, to appear 2001.

[16] N. Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1), 2000.

[17] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based Web Agents. In *Proceedings of First International Conference on Autonomous Agents*, 1997.

[18] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

[19] P. Martin and P. Eklund. Embedding Knowledge in Web Documents. In *Proceedings of the 8th Int. World Wide Web Conf. (WWW'8), Toronto, May 1999*, pages 1403–1419. Elsevier Science B.V., 1999.

[20] *MUC-7 — Proceedings of the 7th Message Understanding Conference*. http://www.muc.saic.com/, 1998.

[21] S. Staab and A. Maedche. Knowledge portals — ontologies at work. *AI Magazine*, 21(2), Summer 2001.

[22] S. Staab, A. Maedche, and S. Handschuh. Creating metadata for the semantic web: An annotation framework and the human factor. Technical Report 412, Institute AIFB, University of Karlsruhe, 2001.

[23] Ka-Ping Yee. CritLink: Better Hyperlinks for the WWW, 1998. http://crit.org/ ping/ht98.html.

OntoWebber: Model-Driven Ontology-Based Web Site Management

Yuhui Jin, Stefan Decker, Gio Wiederhold
Stanford University
{yhjin, stefan, gio}@db.stanford.edu

Abstract. Building data-intensive Web sites and especially Web portals is a costly task, which requires considerable effort for data integration and maintenance and usually does not result in many reusable components. This is mainly because most of the design is hard-coded in static or dynamic Web pages. In this paper we integrate three different approaches to create a comprehensive solution to Web site and Web portal creation dubbed OntoWebber. OntoWebber integrates (1) the explicit modeling of different aspects of Web sites, (2) the use of ontologies as the foundation for Web portal design and (3) semi-structured data technology for data integration and Web site modeling. The resulting system and methodology supports the creation of reusable specifications of Web sites. OntoWebber is the basis for creating the Semantic Web Community Portal as part of the OntoAgents project, which will help the Semantic Web research community (distributed on the Web) to exchange and share knowledge conveniently and effectively.

1 Motivation

Building *data-intensive Web sites* is a high-effort task, which usually does not result in many reusable components, mainly because nowadays most of the design is hard-coded in HTML and executable code like CGI scripts, Active Server Pages (ASP) or Java Server Pages (JSP).

Web portals is a special kind of data-intensive Web sites, which presents a large collection of information related to specific topics and are often organized by hierarchical directories. Examples of Web Portals are Yahoo!, and company portals, which present available resources inside and outside the company to their employees to facilitate cooperation. Knowledge management and dynamic personalization are key features of these Web portals, which make the management of these portals even more demanding than that of an ordinary Web site.

Building and maintaining a portal requires considerable effort for data integration and maintenance, since quite often available information (e.g. inside a large corporation) is heterogeneous, distributed and constantly changing. Therefore it is highly desirable to automate the data integration and maintenance tasks as much as possible.

In software engineering area, design patterns [6], declarative specification approaches and modeling of software artifacts (using e.g. UML) help to generate reusable components and models – this is already partially used for modeling Web sites by approaches like WebML [3].

In the database area, semi-structured data has proven to be very successful as a means to integrate heterogeneous data sources [7]. The usefulness of semi-structured data approaches for modeling Web sites was demonstrated by Strudel Web site management system [5].

Finally, AI-centric approaches have suggested ontologies as a means to organize and present Web Portals [9] [15].

OntoWebber brings efforts from all these different areas together into a coherent system and methodology. It adopts a model-driven, ontology-based approach for declarative Web site management and data integration, and offers support throughout the life cycle of a Web site, including design, generation, personalization and maintenance. The fundamental idea behind OntoWebber is the use of ontologies as the basis for constructing different models necessary for creating a Web site.

As a demonstration of our idea, we are building the Semantic Web Community Portal (SWCP) as part of the OntoAgents project*, which will help the Semantic Web research community (distributed on the Web) to exchange and share knowledge conveniently and effectively. The reference ontology used to structure the design of the Web site is the Semantic Web Research Community Ontology†.

The rest of the paper is organized as follows: the next section presents the architecture and different layers of the OntoWebber system. Section 3 defines the different ontologies necessary to specify a Web site, and describes the modeling of the SWCP as a running example. Section 4 discusses the Web site generation process. Finally Section 5 and 6 present related work, conclusion and future work.

2 The OntoWebber Web Site Management System

In this section, we first describe the system architecture, with all the important software components and how they offer support throughout a Web site's life cycle. Then we introduce the Web site design methodology, which is based on an ontology-based declarative modeling approach.

2.1 OntoWebber Architecture

The architecture of OntoWebber system is shown in Figure 1, which can be decomposed into four layers:

Integration layer. The integration layer resolves *syntactic differences* between different distributed heterogeneous data sources. We have adapted approaches for integration of heterogeneous information sources (e.g. the TSIMMIS approach [7]) by establishing a joint data format over all information sources. As for the semi-structured data format, we have chosen RDF (Resource Description Framework) since it is essentially identical to the OEM (Object Exchange Model) format used in the TSIMMIS project. The key point of our approach for information integration is we convert all types of data into RDF data using the reference ontology, and only perform queries locally to the resulting data stored in the central repository, without going to the data sources.

We currently support three kinds of source data in this layer: RDF data can be directly passed to the articulation layer. Data and ontologies in the UML/XMI format are rewritten by the Data Translator using the InterDataWorking approach described in [12], before passed to the articulation layer. Data sources based on HTML are wrapped and written as RDF data using the reference ontology, thus the resulting RDF data needs no articulation and is directly stored into the repository. Please note that instance data and ontologies are handled uniformly – ontologies are just another kind of data.

* The work is supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory.

† http://www.semanticweb.org/ontologies/

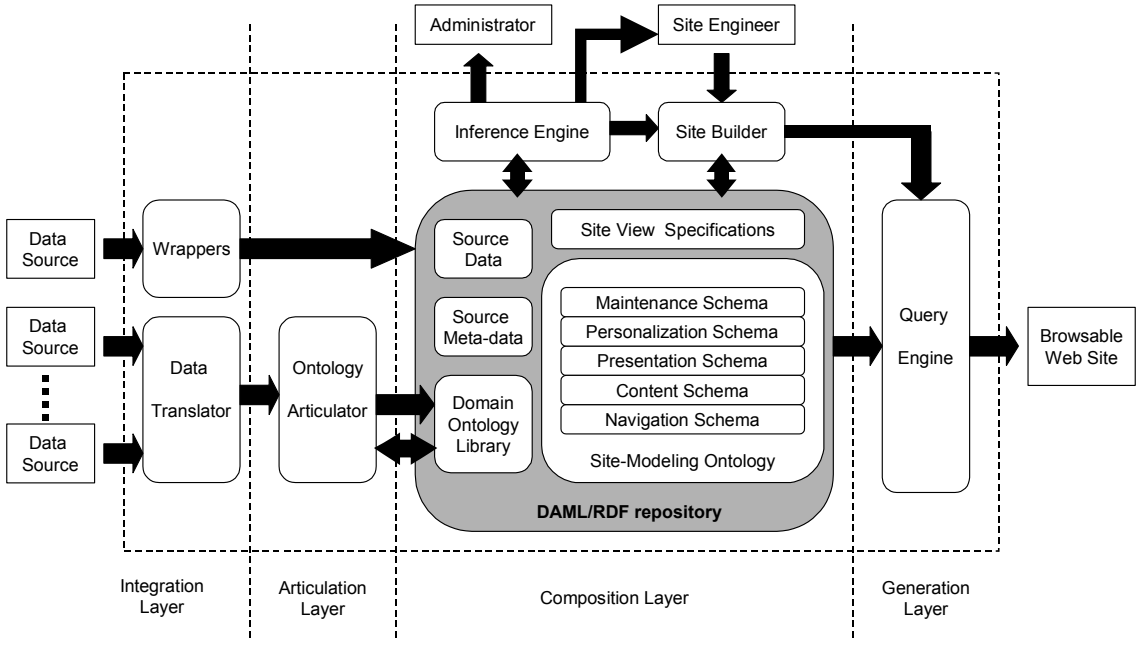


Figure 1. OntoWebber System Architecture

Articulation layer. The articulation layer resolves the *semantic differences* between the different data sources. Even if all source data have been converted into the RDF format. To be able to use the data for the site generation we need to relate the incoming data to the reference ontology of OntoWebber. Since different data providers may use different vocabularies (domain ontologies) to annotate their data, ontology articulation [13] bridges the semantic gap by establishing mapping rules between the concepts and relationships described in source ontologies to those in the reference ontology. Then the data can be queried based on the reference ontology of the OntoWebber system.

Composition layer. At this stage, the reference ontology and RDF data are available, together with articulation rules that relate the source data to the reference ontology. The ontologies for site modeling are a set of predefined schemas using DAML+OIL, available in the central repository as well. Thus, a particular *site view* consisting a set of Web pages can be created from the underlying data. A *site view specification* is a set of site models describing different aspects of a site view based on the site modeling ontologies (see Section 3). Later the site view specification is exported to the query engine (located in the next layer) to be instantiated as Web pages in the desired format. Site models can be constructed using provided software components and are materialized in DAML+OIL. Initially, a default site view is instantiated from a predefined site view specification for general public access. Other site views can be created for specific user or user group by defining their own site view specifications. Furthermore, by declaratively modeling personalization and maintenance of a site, we can achieve these tasks after the site generation phase.

Generation layer. A browsable Web site can be generated by instantiating the corresponding site view with data in the repository. This is done by the query engine, which queries the site view specification for the specific site view to be generated, at the same time queries the data to produce Web pages in desired format. There is a continuum of possibility

of query compilation to materialize Web pages for the site view. By declaratively specifying models, we can achieve all possibilities, i.e. full compilation, partial compilation, or interpretation of Web pages, and choose the optimal compilation strategy depending on various factors such as site requirements, user characteristics, etc.. Details are discussed in Section 4.2.

2.2 Web Site Design Methodology

The design of a Web site is an iterative process [3][14]. Each cycle goes through the following steps:

(1) Requirements analysis. This involves the detailed analysis of objectives of the site, data characteristics, user requirements, etc.. These aspects are the foundation for the site modeling process.

(2) Domain ontology design. The site modeling process starts from designing the default domain ontology, which also serves as the reference ontology for ontology articulation. Analysis of the data helps to extract the common elements to be included in the default domain ontology. The objectives and usage of the Web site will also influence the scope and complexity of the ontology.

(3) Site view design. A *site-view graph* is a graphical representation of three aspects of a site view, i.e., navigation, content, and presentation. The design of the site-view graph is determined by factors like characteristics, preferences, and requirements for targeted users of the site.

(4) Personalization design. Based on user analysis, different personalization elements need to be defined, including categorical information about the user, such as age, browser type, etc., and user requirements such as what operations are expected when changes occur on certain data. If some data elements of interest are missing from the site-view graph designed in step 3, we also need to go back and refine the graph.

(5) Maintenance design. Here we will not dealing with functionality maintenance, which relates to software engineering issues, such as debugging and empowering the software. We only focus on the data maintenance aspect of a Web site. Data maintenance involves manipulating data when certain data changes. Therefore, we need to find out all the anticipated changes of the data, and the corresponding actions to be performed.

3 Modeling of Web Site

From a data management perspective, a Web site here can be considered as a collection of data, including site modeling schemas, site models (i.e., instance of modeling schemas), and source data (i.e., instance of site models). Site model is a notion we use to define all the models we used in the site modeling process, each represents a different aspect of the Web site. There are altogether six types of site models which are shown in Figure 2. To facilitate the processing of these models for Web site management, these six types of site models can further be classified into two categories, site-specific and site-view-specific. If a site model for a particular Web site is *site-specific*, that means there is only one of this type of models for the Web site. Domain model, personalization model and maintenance model all belong to this category. For instance, a personalization model captures all the information about users of a Web site, therefore only one personalization model is needed for any Web site. On the other hand, there can be multiple *site-view-specific* models for a particular Web site. These models

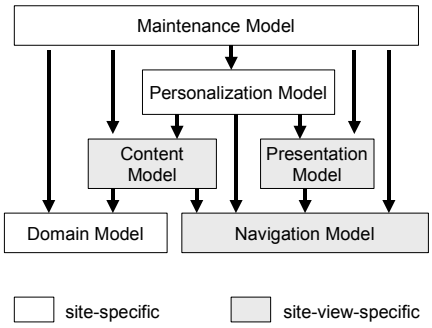


Figure 2. Site Models and Their Relationship

are only specific to a particular site view, tailored for a particular user. Navigation model, content model and presentation model belong to this category. Take content model as an example, for a particular Web site, there could be many content models, though each is associated with a specific site view. Put it another way, a particular site view specification contains a navigation model, a content model, and a presentation model. And a Web site contains multiple site view specifications.

The relationship of the site models is also shown in Figure 2. There is an arrow between two models if the source model refers to the destination model as part of its operational data. Specific to a particular site view, the navigation model specifies the navigational structure of the site view without concerning what content will be associated with primitive elements of the structure. Based on the domain model, content model then relates concepts in the domain to the primitives in the navigation model. The primitives in the navigation model can also be associated with appropriate presentation styles by the presentation model. Specific to the Web site, the domain model defines all the concepts and their properties and relationships in the domain. The personalization model handles the update of individual-dependent data according to user preferences over navigation, content and presentation aspects of their own site views. All these models are part of the operational data for the maintenance model, which not only manages source data, but the other models as well.

The distinct separation of these site models facilitates the conceptual modeling process. Designers can focus on each aspect of the site design at a time without bothering with detailed dependencies on different aspects other than those explicitly specified in the model. Models can also be reused easily, such as the reuse of favorite presentation style with different content and navigation models. The declarative specification of these models also makes it much easier to change any aspect of the site, simply by defining rewriting rules for the models.

The vocabulary (ontologies) for describing site models is a set of pre-defined site modeling schemas using DAML+OIL. Table 1 shows the relationship between models, the schemas used to define them, and meta-schemas (schemas used to define the modeling schemas).

Table 1. Relationship between models and schemas

Site model	Site modeling schema	Meta-schema
Domain model	DAML+OIL	DAML+OIL
Navigation model	Navigation schema	DAML+OIL
Content model	Content schema (and upper ontology)	DAML+OIL
Presentation model	Presentation schema	DAML+OIL
Personalization model	Personalization schema (and upper ontology)	DAML+OIL
Maintenance model	Maintenance schema (and upper ontology)	DAML+OIL

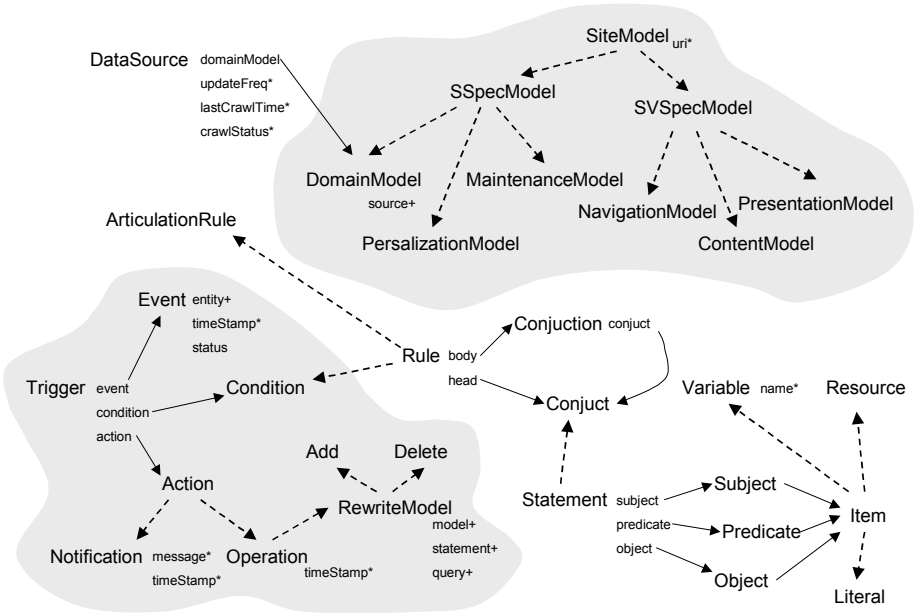


Figure 3. Upper Ontology for Site Modeling

(Note that in this paper, we will use this graphical representation to describe schemas and models, just for illustration purpose. The corresponding serialization in DAML+OIL should be straightforward. Nodes in the graph stands for classes in the ontology, the properties of the class are listed beside it. Solid arrows are used for specifying property values as instances of the pointed class, and the dashed arrows represent sub-class relationship. Asterisk sign (*) indicates the value of the property is a literal, and plus sign (+) means the property value is an instance of a certain class but solid arrow to that class is omitted for readability of the graph.)

To illustrate the process of modeling a Web site, we will present all site modeling schemas and a set of site models described using corresponding schemas for an example site view. The site view can be further instantiated with the data collected from research communities distributed on the Web, and serves as a simplified version of the SWCP.

3.1 Upper Ontology

Before we discuss all site modeling schemas and example site models, we need to define an upper ontology which contains all the necessary concepts either not captured by any of the schemas (e.g., data sources), or will be shared among multiple schemas (e.g., triggers). Figure 3 shows the graphical representation of the upper ontology.

As can be seen in the upper ontology, we define the six types of *site models* as first class objects. This makes models describing the Web site part of the processible data. Management of Web site can thus be reduced to the management of site models. These site models belong to two distinct categories, SSpecModel (*site-specific* model), and SVSpecModel (*site-view-specific* model), as we have discussed before. In the upper ontology, we also explicitly define data sources, rules, and triggers, which will be used later in defining schemas for individual site models.

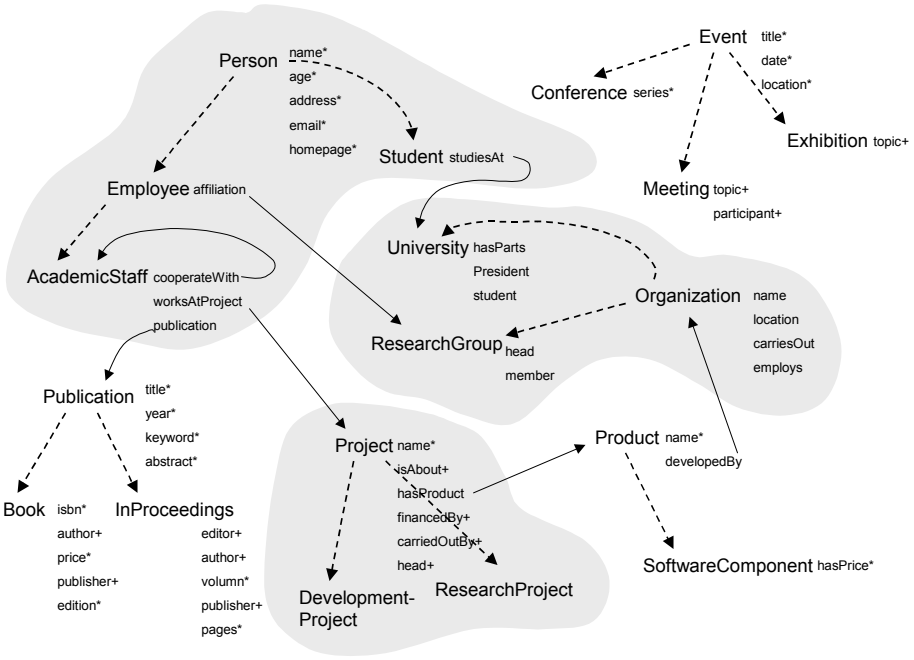


Figure 4. The domain model for SWCP

3.2 Domain Modeling

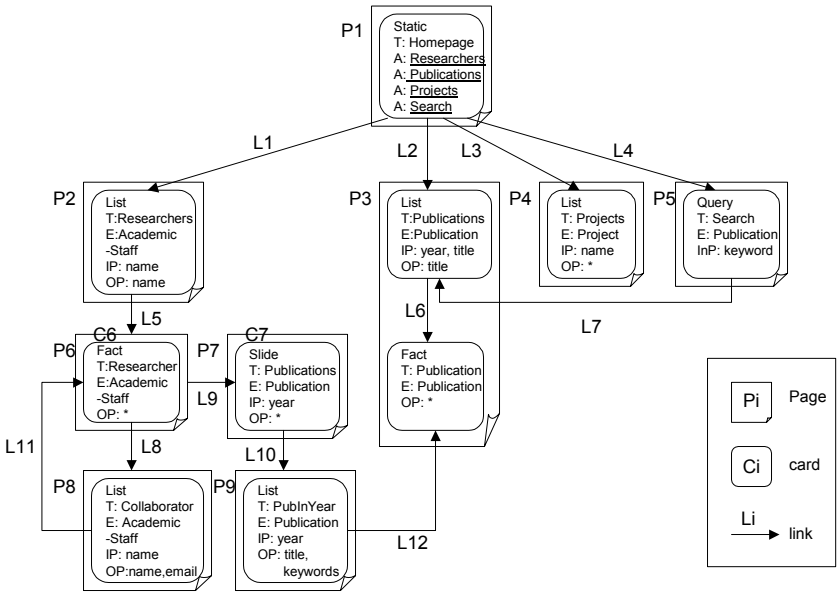
The domain model is actually an ontology constructed by analyzing the collected data in the domain, and extracting common concepts, their properties and relationships. It is used as the reference ontology in ontology articulation for mapping source ontologies to it, and as a foundation for modeling other aspects of a particular site view. The schema for domain modeling is DAML+OIL. The domain model for the example site view of SWCP is shown in Figure 4 (Note some classes and properties such as constraints are omitted due to space limit).

3.3 Site View Modeling

To facilitate the process of modeling navigation, content, and presentation of a particular site view, we have designed a graphical representation called a *site-view graph* to incorporate these three aspects of a site view. By designing a site-view graph, three models of the site view specification can be generated based on the graph, and later guide the instantiation of Web pages in desired format from the underlying data.

3.3.1 Site-view Graph

The site-view graph is a simplified conceptual model to describe hypertext on the Web. It contains a minimal set of design primitives for composing basic information structures in a typical Web site.



(T: title E: entity IP: Indexed Properties OP: Output Properties InP: Input Properties * : all properties)

Figure 5. An example of a site-view graph

Design primitives. The basic elements of a site-view graph are cards, pages and links. A *card* is the minimal unit of a site-view graph. A *page* contains one or more cards and corresponds to a physical Web page. *Links* are used to connect cards to form the navigational structure of the site-view graph. Pages are connected only through the links attached to their cards. The semantics of these design primitives are defined in the schemas for navigation, content and presentation. Each schema categorizes and attaches necessary properties to these design primitives. For instance, a card is attached with properties about coming and outgoing links in navigation schema, entity property in content schema, and font property in presentation schema.

To see how these design primitives fit together to form a complete site-view graph, an example of a site-view graph for the SWCP is presented in Figure 5. The details about the graph will be explained in the rest of the section when we describing each model in site view specification.

Web information structures. The typical information structures on the Web can be categorized into three basic types [8]:

- *Sequential*. A linear form of information flow, the simplest and most common structure. The linkage from page P6 to P7 in Figure 5 is an example.
- *Hierarchical*. A hierarchical structure involves having a page linking to lower level pages of detail. An example could be the root page P1 links to next level of pages P2 to P5 in Figure 5.
- *Associative*. This structure involves nonlinear navigation, fundamentally any structure that is not sequential or hierarchical. L7 in figure 5 is an example where the retrieval of a list of publications as search result from database helps to form the linkage between the two pages.

The information structure of a site view is usually a combination of the above structures. The provided design primitives are able to form all these basic information structures, which are the building blocks of a site view.

Minimalist Approach. Web sites that are too complicated for their intended user will likely frustrate the user and make site maintenance a nightmare. Sophistication could add value if applied appropriately, but still it depends on the nature of the Web site, and the intended user. When modeling a site view, we explicitly offered a set of guidelines, in favor of a predictable and consistent user interaction and ease of maintenance.

A minimalist design includes the following ingredients. A site-view graph always starts from a default root page, with links to the first level of pages. Content information is categorized and aggregated by cards and pages. Only one type of content is contained in each card. Navigation is made possible only through links. Moreover, a site map is generated out of the site-view graph, which can be presented in a separate page or as part of the root page. This site map makes the structure of the site visible to users, and gives the location information so they know where they are and where they can go.

3.3.2 Navigation modeling

The navigation model of a site view is a description of the site-view graph with respect to how the cards and pages are connected through links, without concerning what semantics will be associated with these primitives. The schema for navigation modeling is shown in Figure 6.

We classified cards into two categories, dynamic cards and static cards. Dynamic card contains content that depends on the changes of source data, i.e., the query used to generate the content needs to be reevaluated if source data changes. A further classification of dynamic card is the following four types of card. Each represents a typical way to structure information within a card:

- *Fact Card*. Only one instance of the entity will be shown with specified output properties in the card.
- *List Card.* A list of instances of the entity will be shown, with indexes on key properties (i.e., some literal properties of the class). And each instance will be shown with specified output properties.

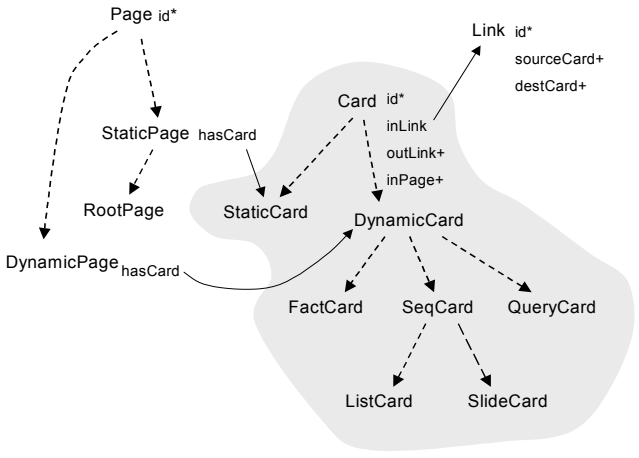


Figure 6. Navigation Schema

- *Slide Card.* A sequence of instances of the entity will be shown with specified output properties in the card, one instance at a time, and hyperlinks are created to browse nearby instances (in the order given by specified key properties) in the sequence.
- *Query Card.* A set of input properties needs to be filled out to search for entities satisfying these criteria.

The list-card and slide-card are both a type of sequence-card, which means the content of these cards is a sequence of instances. Details about how content is generated and presented in the card are discussed in content modeling (see Section 3.3.3). Static card contains content which is source data independent, such as static text, and images. A common example is the root page of a site view, which is always a static card, with predefined anchor texts leading to the next level pages. Pages are also classified into dynamic and static types according to the types of cards they contain.

Navigation model of a site view can be defined using the given schema. As an example, a portion of the navigation model for the example site view is presented in Figure 7.

3.3.3 Content Modeling

The content model associates meanings to design primitives in the site-view graph. The schema used for content modeling is shown in Figure 8. It basically specifies two aspects of content modeling. One is how to present the content in a rendered card (part of a Web page), the other is how to generate the content for a specific card.

How to present the content in a card can be explained by meanings of the attached properties to different card classes in the content schema. Each card has a property 'title' that can be used when rendering the card in Web pages. For a static card, we define all types of static elements that can be contained in the card, i.e., text, image, and anchor.

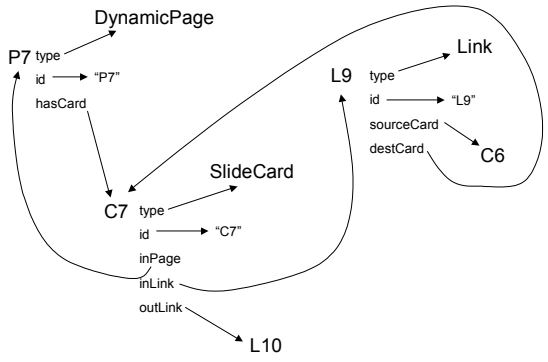


Figure 7. Navigation model for example site view

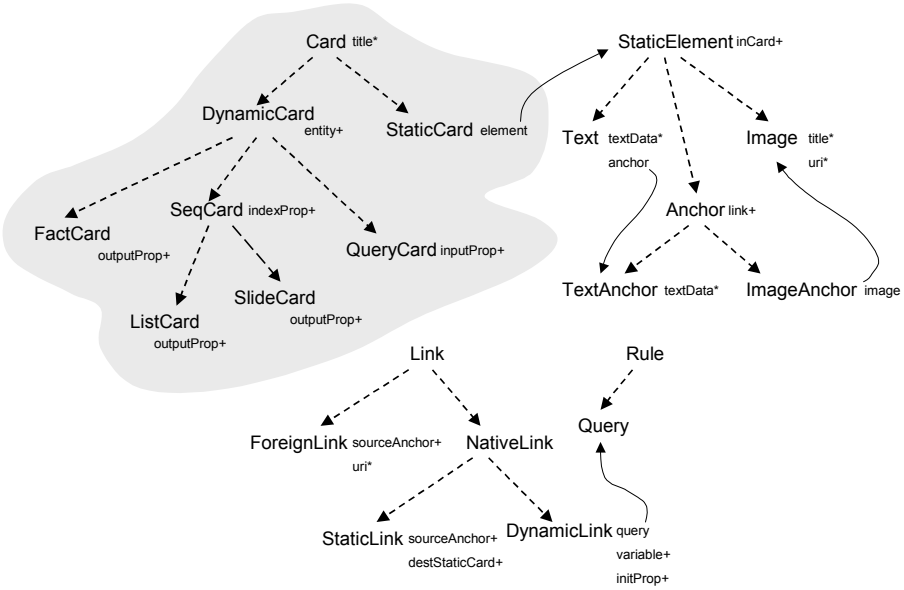


Figure 8. Content Schema

For a dynamic card, a property 'entity' takes a value as a class in the domain model. The content of the card is instantiated with instances of this class. The 'outputProp' property of a card specifies what properties of each instance are listed on the card. For sequence-card, the 'indexProp' property specifies the set of indexed properties used in ordering the listed instances. A constraint on this property (which is omitted in Figure 8) is it must take values that are literals (i.e., can not be type of 'resource'). Consider an example in Figure 5, card C7 is a slide-card, which is a type of sequence-card, it has the following listed properties: title 'Publications', which can be the name of the card when it is rendered in a Web page; entity 'Publication', which means only instances of class Publication can be contained in the card; indexed property 'year' (note we can have more properties listed to form a multi-key index), so the order of listed publication instances has to be determined by values of their 'year' property; output property '*', this means values of all the properties of publication instances are shown in the card.

The generation of content (i.e., instances of entity property of a card) can be realized by attaching certain properties to link classes. Links can be either foreign (linking to a page outside the current Web site) or native (linking to a Web page inside the current Web site). Native links can be either static or dynamic. Static links connect cards without any information flow, while dynamic links always connect dynamic cards, where the content in the destination card is determined by information passed from the source card. To instantiate the destination card with desired content, we associate three properties to a link: a *query* property, which can be assigned with a query the execution of which produces the content of the card; a *binding-variable* property, which indicates variables in the query which will be instantiated with data values passed from the source card; and *initiating-property* property, which helps to create a hyperlink in the source card. The query is a type of 'Rule' class, and the binding-variable is a type of 'Variable' class. A query will first be rewritten with binding variables replaced with data values passed from source card, then executed to produce the instances.

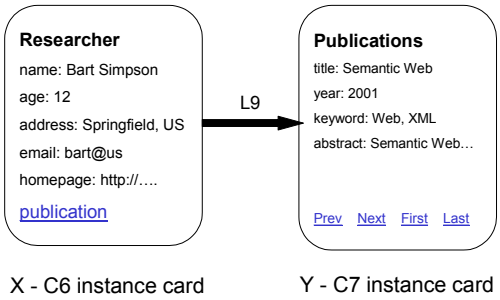


Figure 9. An example of content instantiation

An example of content instantiation is shown in Figure 9, where we extracted the portion of site-view graph (see Figure 5) containing card C6, C7 and link L9. In this example, we name the C6 instance card as X, and similarly, the C7 instance card as Y. Suppose X is already instantiated with an instance of class 'AcademicStaff', whose URI is 'Bart' (we use first name of the instance as the URI only for illustration purpose). And now we need to instantiate Y, which is intended to have a slide show of all instance publications of 'Bart'.

The way the instantiation is done is by specifying appropriate values to the query and binding-variable properties of link L9. The query[‡] attached with link L9 is:

```
FORALL A,P <-
      A[type->AcademicStaff] and A[publication->P]
```

The binding-variable value of link L9 is 'A'. To add a hyperlink in X, the initiating property takes the value as 'publication'. Then the query is evaluated after replaced the variable 'A' with the URI 'Bart'. Given destination card is a slide-card, a set of Web pages is instantiated with each instance in the query result (assume we are instantiating Web pages statically). These Web pages are linked together by pre-defined hyperlinks in a slide-card, such as 'Prev' and 'Next', and ordered by the indexed properties of the card C7. Finally, a static card containing an anchor which links to the first of these Web pages is created and added into X. The name of the anchor is given by the initiating property, which is 'publication'.

Another different type of query is that for link L7 in Figure 5. Because the source card is a query-card, the instances to be contained in the destination card cannot be compiled statically. The variable binding has to take place at run-time when user specifies the search criteria. The query for L7 is

```
FORALL P,K,T <-
      P[type->Publication] and P[keyword->K] and
      P[title-> T]
```

[‡] Queries and rules in this paper are written in TRIPPLE notation. TRIPPLE is an inference engine we are developing for the OntoAgents project. Note that O[P->V] stands for a statement in RDF (O,P,V).

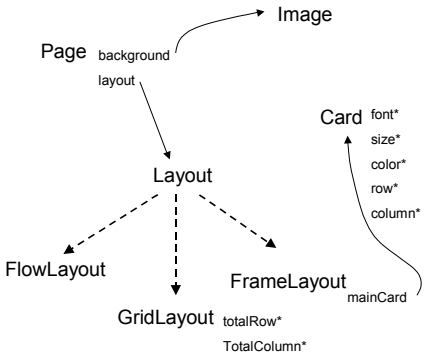


Figure 10. Presentation schema

The binding variable is 'K'. And initiating property is NULL, since no need to have hyperlink in the search page. After replacing variable K with values entered by the user, the query will return titles of all the publication instances with the specified keywords. An instance of the destination card will then be instantiated dynamically as a list of titles of publications.

3.3.4 Presentation Modeling

The third aspect of a site view is the presentation model. By associating presentation elements to design primitives in the site-view graph, the presentation model specifies the look-and-feel of the Web pages generated from the site view. The schema for creating presentation models is shown in Figure 10.

The background of the page can be chosen as an instance of images. Card and page both have style elements like font, color, etc., with the elements of card, if present, overriding those of the containing page. The layout of cards in a page can be one of the three types. The *flow layout* (default layout) arranges all the cards in a row, the *grid layout* maps these cards to certain position on the screen, and the *frame layout* places one card, denoted by mainCard property, in the static frame, and other cards in the dynamic one.

3.4 Personalization Modeling

Personalization in our approach includes providing personalized content collection, navigational experience, and presentation style through adapting the site view to the needs of users. This is accomplished by manipulating all three models of the site view. The schema for personalization modeling is shown in Figure 11. Users are explicitly modeled by three properties, i.e., capacity, interest and request.

Capacity property describes basic information about the user, such as age, preferred browser type, connection speed, etc.. The capacity of a user can be used to assign user to certain predefined groups, and adjust presentation styles for better online experience. *Interest* aspect of the user includes the three models of the site view of the user. These models specify the user's site view and can be rewritten to specify a new site view. And *request* property defines triggers which will be fired if certain conditions are satisfied, the actions of the trigger is either update the site view by model rewriting, or notify user by messages.

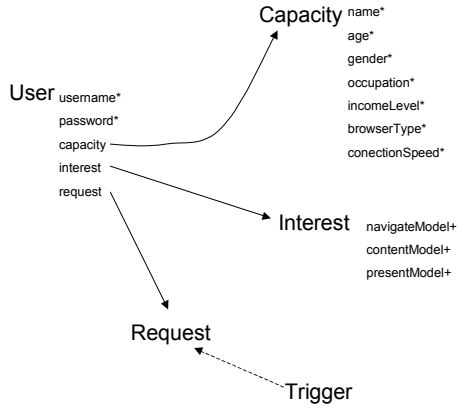


Figure 11. Personalization schema

Basically, two types of personalization can be provided by the system. The *fine-grained personalization* is achieved by defining the personalization model using the above schema. A *coarse-grained personalization* can also be used, by assigning user to specific user group. For each user group, a particular site view and personalization model is constructed. This can be modeled by defining similar properties for user groups as for users in the above schema and add relationship between user and user group. In the course-grained personalization, the site view of the user will not be updated as often as in the fine-grained personalization, since it only changes when group view changes. This helps to reduce workload of the system considerably.

3.5 Site Maintenance Modeling

Maintenance of a Web site typically falls into two categories, content maintenance and functionality maintenance. The later can be further classified into corrective, adaptive, and perfective maintenance [14]. Here we will focus on the content maintenance aspect, since the functionality part is more of a software-engineering issue, while what we are interested is the data management of a Web site.

From data management point of view, Web site maintenance can be regarded as a manipulation of data when certain data changes. Therefore, we come to a simple schema for maintenance modeling, which is shown in Figure 12.

Administrator is the target object of maintenance rules, and will update the source data, meta-data, and site view specifications according to the fired triggers. There are basically two types of maintenance rules.

User-oriented rules. Administrator is a super user, who has the authority to initiate actions that influence users and user groups with certain properties. It can be achieved by rewriting the personalization model. An example of these rules could be “if any instance of Book about Semantic Web (e.g. title or keyword contains the phrase) has been published, re-compute the site views of users who are working on a project about DAML.

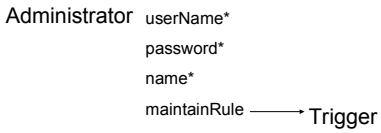


Figure 12. Maintenance Schema

Site-oriented rules. Administrators can also perform operations on meta-data, which provides basic information about data sources (the frequency of updates, crawl status, etc.) and about the Web site itself (number of users and user groups, different versions of ontologies, etc.). This is basically handled by rewriting the maintenance model. For instance, a rule of this type could be "if source A has changes weekly, and today is six days after the most recent crawling of the source site, then schedule the crawling for today".

4 Web Site Generation

The generation of a browsable Web site is an instantiation of a particular site view. It can be described as a two-phase process. First, integrity constraints are verified over the site view specification. Second, Web pages are materialized by querying the source data based on the specified site models.

4.1 Constraint Verification

Constraint verification on a traditional Web site is a difficult task. For example, the checking of whether each page is reachable from the root page are usually performed by manually following each link in all the pages, which takes much effort especially when the Web site contains a large number of pages. The constraints of the Web site generated using our approach can be easily verified. Since ontologies (i.e., site schemas and models) are explicitly specified using DAML+OIL, constraint verification becomes a direct application of semantics of the ontologies. On the other hand, it is also part of the reasoning facility provided by the inference engine, as we define rules and verify them against the ontologies. Note that a complete formalization of the ontologies is undesirable as it takes considerable amount of effort, and offers no obvious benefit.

There are mainly three types of integrity constraints to be verified against each of the three models in the site view specification.

Structural constraints. They dictate all the legal patterns of navigation in the site-view graph. These are verified against the navigational aspect of the site view. Examples of the constraints could be expressed as the following rules:

(a) Every dynamic card has at least one incoming link.

```
<- FORALL C C[type->DynamicCard] ->
   EXISTS L L[type->Link] and C[inLink->L] and L[destCard->C]
```

(b) Every card is reachable from the root page, this is expressed by defining a property called 'reachable' using recursion, and using this property to check the constraint.

```
FORALL X,Y X[reachable->Y] <-
      X[type->Card] and Y[type->Card] and
      EXISTS L X[inLink->L] and Y[outLink->L]
FORALL X,Y,Z X[reachable->Z] <-
      X[type->Card] and Y[type->Card] and Z[type->Card]
      and X[reachable->Y] and Y[reachable->Z]
<- FORALL P,X P[type->Rootpage] and X[type->Card]
      -> EXISTS Y Y[type->Card] and P[hasCard->Y] and
      X[reachable->Y]
```

Semantic constraints. The content model of a site view is validated based on semantic information. Example constraints could be:

(a) The entity associated with a query-card must match the entity associated with its destination card, since the search result should relate to the same entity.

```
<- FORALL X,Y,L,E1,E2 X[type->QueryCard] and Y[type->Card] and
      L[sourceCard->X] and L[destCard->Y] and X[entity->E1] and
      Y[entity->E2] -> E1 = E2
```

(b) The value of 'initProp' property of a dynamic link should be one of the properties of the entity associated with the destination card (see Section 3.3.3).

```
<- FORALL L,I,D,E L[type->DynamicLink] and L[initProp->I]
      and L[destCard->D] and D[entity->E] -> I[domain->E]
```

Presentational constraints. The look-and-feel of Web pages can be different on different platform, browser, and even depends on the connection speed. An example of how these presentation elements influence the site view could be: do not allow usage of background image and frame layout in the site view given a low connection speed.

```
<- FORALL P,U,C,S U[type->User] and U[capacity->C] and
   C[connetionSpeed->'Low'] and P[type->Page] ->
   NOT EXISTS X P[background->X] and
   NOT EXISTS Y P[Layout->Y] and Y[type->FrameLayout]
```

Constraints involving multiple site models are also possible. An example is to verify that queries produce a full text version of a site view. This requires defining rules to check both the static elements in the content model and page elements in the presentation model.

4.2 Site View Instantiation

After integrity constraints verification has been done, models in the site view specification will be compiled into Web pages to produce a browsable Web site. A typical instantiation is carried out by the query engine which generates HTML pages with data from the repository, based on the navigation and content model, and produce CSS style-sheets based on the presentation model.

Because all models are specified declaratively, there exists a continuum of possibilities in page compilation, which did not exist in any prior system. The possibilities range from full pre-generation of HTML pages at compile time (full compilation), to partial compilation (e.g.

a Java Servlet builds the HTML page, and instantiate pages at runtime out of data stored in database), to full interpretation (the models are interpreted by an interpreter, which creates the all HTML output at runtime).

Different solutions have different application areas. Pre-generation of the HTML code is desirable if the load of the Web site is extremely high and it is too expensive to access a database when a request comes in. Unfortunately pre-generation is also the most inflexible solution – a change to the source data may require a large number of the site views to be re-instantiated. An example of a full pre-generation system is Strudel. Partial compilation is desirable, if the load of the Web site is in balance with the need to reflect updates of the database quickly in the generated HTML code. Torri is an example of a Web site management system realizing partial compilation. The models defined for the Web site are used to generate JSP scripts. Finally, full interpretation of the models is the most inefficient, but also the most flexible way of Web site creation. The MyYahoo.com Web site [10] can be regarded as an interpretative system. Changes to the layout and the selection of modules are directly reflected in the personalized view of the Web site.

5 Related Work

Given the amount and complexity of the Web content, research has been conducted in the large context of extending database techniques for data on the Web, particularly with the goal of facilitating the creation and maintenance of data intensive Web sites. A few systems, such as ARANEUS[11], AutoWeb[4], Torri[2], have been developed using a model driven approach, which is adapted from classical database and hypermedia design methodologies. These systems have their own data models, query languages, and sets of CASE tools to facilitate the process of wrapping, modeling, generation and querying. But the common theme is a high-level description of a Web site by distinct orthogonal dimensions. Those dimensions include the modeling of information content, page composition, navigation, and presentation. Personalization by means of user modeling and business rule management has also been added in later systems. However, none of the approaches deal with integration of heterogeneous data sources, which is what OntoWebber is explicitly designed for. Since in OntoWebber ontologies and site models for different aspects of the site are both expressed in RDF, they can be rewritten and queried statically or dynamically, which is another feature not present in these systems, because they do not have a unified data model like RDF and do not construct ontologies for every aspects of site modeling.

Other systems like Strudel[5] and its variant Tiramisu[1], address the problem of data integration, and establish a separation over the Web site data management, content and structure specification, and visual presentation. The emphasis of their approach is the declarative specification of a Web site's structure and content. By defining a declarative query language, they showed how the generation of a Web site can be automated by querying the data graph to construct a site graph (i.e., site view), and how integrity constraints of the generated site can be enforced by reasoning over the site structure. However, they do not have fine-grained modeling hierarchies for each aspects of a Web site as in OntoWebber. Site view in these systems is simply a data graph containing all the navigation and content information. The presentation style is hard coded in the HTML templates. While in OntoWebber the site view is specified as three distinct models to separate the aspect for navigation, content, and presentation. The rewriting and reusing of these models eases the maintenance work and

enables flexible personalization. Since strudel does not concern the aspects of site maintenance and personalization, it is actually only an implementation tool, not a management system.

AI approaches like SEAL[9] focus on the presentation of portals based on a domain ontology. However, SEAL does not enable the creation of a site by modeling the site itself, and also offers no support for the integration of heterogeneous data sources. Although SEAL provides a set of tools (e.g. the Java based rule engine SiLRI), the underlying site still needs to be created and programmed in a conventional manner.

The fundamental difference between the OntoWebber approach and previous approaches is the integration of three different aspects into a coherent framework: (1) integration of heterogeneous data sources based on a formal model for semi-structured data, (2) explicit ontologies, which help to structure, create and generate the site, and (3) a rigorous modeling methodology, which helps to create reusable models.

6 Conclusion and Future Work

We proposed a model-driven ontology-based system architecture for creating data intensive Web sites and portals. Our approach combines the advantages of different technologies: Semi-structured data technology is used to integrate heterogeneous data sources; Declarative models help to define a Web site without hard-coding design into static or dynamic Web pages; Ontologies are used to provide access to the underlying data and guide the modeling process.

Future work includes the development and integration of all the software components of the system. We have already developed a number of tools, such as Web crawler, ontology articulation tools, data translation tool, ontology construction tool, and RDF query and storage facilities. However, the graphical Web site modeling tool and the inference engine are still under development. Furthermore, a number of problems to be considered could be:

Management of evolving ontologies. Different versions of ontologies (i.e., schemas) might be incompatible with each other. How to manage evolving ontologies and retain consistency and usability of ontologies is a necessity for the robustness of a Web site.

Optimization strategies for site generation. Evaluation and performance measurements of dynamic and static Web site generation needs to be investigated. And a set of optimization strategies can thus be defined to guide the cost-effective generation of Web site.

Adaptation to handle dynamic services. Currently approach of OntoWebber only deals with static information sources. With UDDI[§] and Microsofts.NET initiative, more and more dynamic Web services will be available, which will be integrated into portals and Web sites. OntoWebber needs to define appropriate ontologies to be able to handle dynamic services as well.

References:

[1] Corin R. Anderson, Alon Y. Levy, Daniel S. Weld: Declarative Web Site Management with Tiramisu. WebDB (Informal Proceedings) 1999: 19-24.

[2] Stefano Ceri, Piero Fraternali, Stefano Paraboschi: Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. VLDB 1999: 615-626.

[§] http://www.UDDI.org

[3] Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites WWW9 Conference, Amsterdam, May 2000.

[4] Piero Fraternali, Paolo Paolini: A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. EDBT 1998: 421-435.

[5] Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, Dan Suciu: Declarative Specification of Web Sites with Strudel. VLDB Journal 9(1): 38-55 (2000).

[6] Gamma, E., Helms, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[7] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS". In *Proceedings of the AAAI Symposium on Information Gathering*, pp. 61-64, Stanford, California, March 1995.

[8] Charles J. Lyons, Essential Design for Web Professionals, Prentice Hall, 2000.

[9] A. Mädche, S. Staab, N. Stojanovic, R. Studer, Y. Sure. SEAL - A Framework for Developing SEmantic portALs. In: BNCOD 2001 - 18th British National Conference on Databases. Oxford, UK, 9th - 11th July 2001, LNCS, Springer Verlag, 2001.

[10] Udi Manber, Ash Patel, John Robison: Experience with Personalization on Yahoo! Communications of the ACM Vol. 43, No. 8 (August 2000), Pages 35-39.

[11] G. Mecca, P. Merialdo, P. Atzeni, V. Crescenzi The (Short) Araneus Guide to Web-Site Development - Second Intern. Workshop on the Web and Databases (WebDB'99) in conjunction with SIGMOD'99, May 1999.

[12] Sergey Melnik, Stefan Decker: A Layered Approach to Information Modeling and Interoperability on the Web. ECDL 2000 Workshop on the Semantic Web. 21 September 2000, Lisbon Portugal.

[13] Prasenjit Mitra, Gio Wiederhold, Martin L. Kersten: A Graph-Oriented Model for Articulation of Ontology Interdependencies. *Proceedings of the 7th International Conference on Extending Database Technology, EDBT 2000*, March 2000 Springer Verlag.

[14] Tomas A. Powell, David L. Jones, Dominique C. Cutts, Web Site Engineering Beyond Web Page Design, Prentice Hall, 1998.

[15] Steffen Staab, Jürgen Angele, Stefan Decker, Michael Erdmann, Andreas Hotho, Alexander Mädche, Hans-Peter Schnurr, Rudi Studer, York Sure. Semantic Community Web Portals. In: WWW9 / Computer Networks (Special Issue: WWW9 - Proceedings of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, May, 15-19, 2000), 33(1-6): 473-491. Elsevier, 2000.

Indexing a Web Site with a Terminology Oriented Ontology

E. Desmontils & C. Jacquin
IRIN, Université de Nantes
2, Rue de la Houssinière, BP 92208
F-44322 Nantes Cedex 3, France
{desmontils,jacquin}@irin.univ-nantes.fr
http://www.sciences.univ-nantes.fr/irin/indexGB.html

Abstract. This article presents a new approach in order to index a Web site. It uses ontologies and natural language techniques for information retrieval on the Internet. The main goal is to build a structured index of the Web site. This structure is given by a terminology oriented ontology of a domain which is chosen a priori according to the content of the Web site. First, the indexing process uses improved natural language techniques to extract well-formed terms taking into account HTML markers. Second, the use of a thesaurus allows us to associate candidate concepts with each term. It makes it possible to reason at a conceptual level. Next, for each candidate concept, its capacity to represent the page is evaluated by determining its level of representativeness of the page. Then, the structured index itself is built. To each concept of the ontology are attached the pages of the Web site in which they are found. Finally, a number of indicators make it possible to evaluate the indexing process of the Web site by the suggested ontology.

keywords : Information Retrieval on Internet, Indexing Web Pages, Ontologies, Semantic Indexing.

1 Introduction

Searching for information on the Internet means accessing multiple, heterogeneous, distributed and highly evolving information sources. Moreover, provided data are highly changeable: documents of already existing sources may be updated, added or deleted; new information sources may appear or some others may disappear (definitively or not). In addition, the network capacity and quality is a parameter that cannot be entirely neglected. In this context, the question is: how to search for relevant information on the Web more efficiently? Many search engines help us in this difficult task. A lot of them use centralized databases and simple keywords to index and to seek the information. Within such systems, the recall[1] is often relatively high. Conversely, the precision[2] is weak. An intelligent agent supported by the Web site may greatly improve the retrieval process ([4], [1]). In this context, this agent knows its

[1] Recall is defined as the number of relevant documents retrieved divided by the total number of relevant documents in the collection

[2] Precision is defined as the number of relevant documents retrieved divided by the total number of documents retrieved

pages content, is able to perform a knowledge-based indexing process on Web pages and is able to provide more relevant answers to queries. In information retrieval processes, the major problem is to determine the specific content of documents. To highlight a Web site content according to a knowledge, we propose a semi-automatic process, which provides a content based index of a Web site using natural language techniques. In contrast with classical indexing tools, our process is not based on keywords but rather on the concepts they represent.

In this paper, we firstly present the general indexing process (section 2). After having exposed the characteristics of used ontologies (section 3), we will indicate how the representativeness of a concept in a page is evaluated (section 4) and, finally, how this process is evaluated itself (section 5).

2 Overview of the indexing process

The main goal is to build a structured index of Web pages according to an ontology. This ontology provides the index structure. Our indexing process can be divided into four steps (figure 1:

1. For each page, a flat index is built. Each term of this index is associated with its weighted frequency. This coefficient depends on each HTML marker that describes each term occurrence.

2. A thesaurus makes it possible to generate all candidate concepts which can be labeled by a term of the previous index. In our implementation, we use the Wordnet thesaurus ([23]).

3. Each candidate concept of a page is studied to determine its representativeness of this page content. This evaluation is based on its weighted frequency and on the relations with the other concepts. It makes it possible to choose the best sense (concept) of a term in relation to the context. Therefore, the more a concept has strong relationships with other concepts of its page, the more this concept is significant into its page. This contextual relation minimizes the role of the weighted frequency by growing the weight of the strongly linked concepts and by weakening the isolated concepts (even with a strong weighted frequency).

4. Among these candidate concepts, a filter is produced via the ontology and the representativeness of the concepts. Namely, a selected concept is a candidate concept that belongs to the ontology and has an high representativeness of the page content (the representativeness exceeds a threshold of sensitivity). Next, the pages which contain such a selected concept are assigned to this concept into the ontology.

Some measures are evaluated to characterize the indexing process. They determine the adequacy between the Web site and the ontology. These measures take into account the number of pages selected by the ontology, the number of concepts included in the pages... The index is built as a XML file ([28]) and is independent of Web pages.

Our process is semi-automatic. It enables the user to have a global view of the Web site. It also makes it possible to index a Web site without being the owner of these pages. We do not regard it as a completely automatic process. Adjustments should be carried out by the user. The counterpart of this automatisation is, obviously, a worse precision of the process. Lastly, compared to the annotation approach, our indexing process improves information retrieval: it

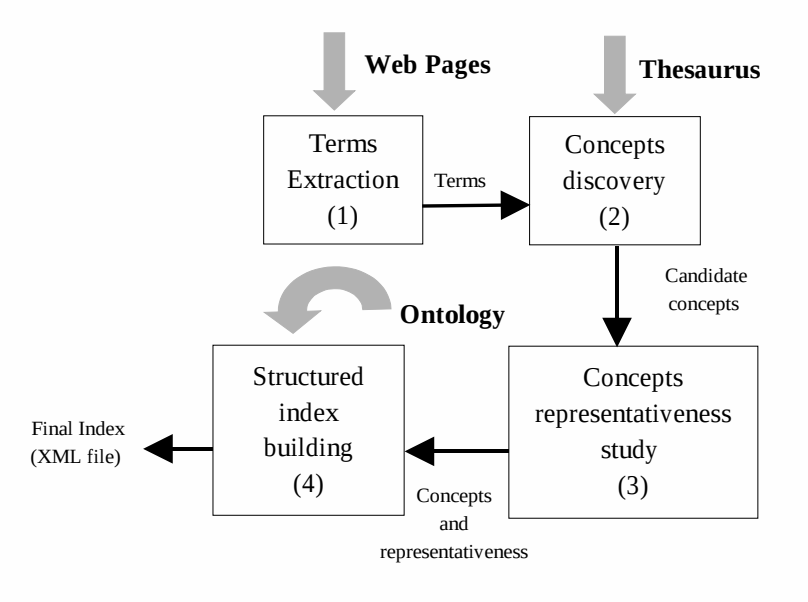


Figure 1: The indexing process

makes it possible to reach directly the pages concerning a concept. By contrast, the annotation approach requires to browse all the pages of the Web site to find this same information. Now, we will study two significant elements: the ontology and the method to evaluate the concepts.

3 Terminology oriented ontologies

3.1 Ontology definition

The term ontology comes from philosophy. In this context, its definition is: «*systematic explanations of the existence*». Futhermore, researchers in Knowledge Engineering give other more suitable definitions with their concerns. In this context, their definitions are strongly dependent on the author's point of view and on his use of ontologies [12, 13]. Some have a formal point of view and work on abstract models of ontologies while others have a more pragmatic approach.

We have chosen this definition of ontology: "*an ontology provides the common vocabulary of a specific domain and defines, more or less formally, terms meaning and some of their relationships*" ([11]). In our context, we thus call ontology a hierarchy of concepts defines in a more or less formal way. For instance, figure 2 shows an extract of the SHOE ontology concerning the american universities.

3.2 Terminology oriented ontology

The concepts of ontologies are usually represented only by a single linguistic term (a label). However, in our context, this term can be at the same time ambiguous (it represents several candidate concepts) and not always unique (existence of synonyms). As a result, within the framework of texts written in natural language, it is necessary to determine the whole set of the synonyms (candidate labels) to define in a single way a concept. Such process can

```
<?xml version="1.0" encoding="ISO-8859-1"
      standalone="no"?>
<!DOCTYPE ontology SYS-
TEM "http://.../onto.dtd">
<ontology id="university-ont" version="2.1"
          description="...">
  <def-category name="Department"
                isa="EducationOrganization"
                short="university depart-
ment"/>
  <def-category name="Program"
                isa="EducationOrganization"
                short="program"/>
  <def-category name="ResearchGroup"
                isa="EducationOrganization"
                short="research group"/>
  <def-category name="University"
                isa="EducationOrganization"
                short="university"/>
  <def-category name="Activity"
                isa="SHOEEntity"
                short="activity"/>
  <def-category name="Work"
                isa="Activity"
                short="work"/>
  <def-category name="Course"
                isa="Work"
                short="teaching course"/>
  ...
</ontology>
```

Figure 2: Extract of the SHOE ontology concerning the american universities

be found in a manual way in OntoSeek ([14]) or in a semi-automatic way in Mikrokosmos ([25]).

In our context, an ontology is a set of concepts each one represented by a term (a label) and a set of synonyms of this term, and a set of relationships connecting these concepts by the specific/generic relationship, the composition relationship,... Currently, the only relationship we take into account is the “isa” relationship. We call this type of ontology a terminology oriented ontology. Note that our ontologies do not reflect all the inherent aspects to formal ontologies ([11]). Our ontologies are close by their structure to those used in the SHOE project ([21]). Moreover, we choose XML format ([28]) to store our ontologies and our indexing results. The used DTD is rather similar to the SHOE DTD but we made modifications and extensions to this last.

We thus propose a process which makes it possible to determine all the candidate labels of a concept. This process is based on a thesaurus and uses a number of heuristics similar with those proposed by the Mikrokosmos project. The general principle of these heuristics is to try to make a correspondance between the paths according to the “isa” relationship in the

ontology and the paths of hypernyms in the thesaurus. According to the "matching degree", a more or less large confidence is given to such or such set of synonyms (concept). Let us note that experiments using the relationship of composition have not improved the results.

The user can manually finish the disambiguation process of the labels. Indeed, the process can not always select in an unquestionable way the good set of synonyms. The definitions of the sets of candidate ynonyms are presented in order to help to this final choice.

However, the process gives results rather satisfactory since it chooses the good sense for nearly 75% of the labels associated with the concepts for the ontology of the Universities of SHOE ([21]) and for 95% of the label after several modifications (contradictions with the used thesaurus were deleted).

These evaluations were determined with ontologies for which the whole set of the labels associated with the concepts was manually disambiguated. Of course, this disambiguation process depends on the thesaurus used (in our case Wordnet).

4 Index building

The other important part of our process is the indexing process and the evaluation of the importance of a concept in a HTML page. There are two essential steps: (1) terms extraction from Web pages and calculus of the weighted frequency and (2) determination of candidate concepts and the calculus of the representativeness of a concept.

4.1 Terms extraction

The well-formed terms extraction process starts by (1) removing HTML markers from Web pages, (2) dividing the text into independent sentences, and (3) lemmatizing words included in the page. Next, Web pages are annotated with part of speech tags using the Brill tagger ([3]). As a result, each word in a page is annotated with its corresponding grammatical category (noun, adjective...). Finally, the surface structure of sentences is analyzed using term patterns (Noun, Noun+Noun, Adjective+Noun...)[7] to provide well-formed terms . For each selected term, we calculate its weighted frequency. The weighted frequency takes into account the frequency of the term and especially the HTML markers which are linked with each of its occurrences. We can notice that the frequency is not a main criterion. Indeed, we work with pages which are of rather restricted size compared to large corpora used in NLP (Natural Language Processing). The influence of the marker depends on its role in the page. For example, the marker "TITLE" will give a considerable importance to the term (*10) whereas the marker "B" (for bold font) has a quite less influence (* 2). The table 1 gives the weight of the most significant markers (the markers weights were determined in an experimental way [10]). In a Web page containing n different terms, for a given term T_i (with i in $1..n$), the weighted frequency $F(T_i)$ is determined as the sum of the p weights of HTML markers associated with the p term occurences. The result is then normalized. This calculus is shown in formula (1) and (2) where $M_{i,j}$ corresponds to the HTML marker weight associated with the jth occurrence of the term T_i.

$$F(T_i) = \frac{P(T_i)}{max_{k=1..n}(P(T_k))} \quad (1)$$

```
<?xml version="1.0" encoding="ISO-8859-1"
      standalone="no"?>
<!DOCTYPE ontology SYSTEM "http://.../onto.dtd">
<ontology id="university-ont" version="3.0">
  <def-category name="Course" short="teaching course"
                isa="Work">
    <sense name="Course" no="1" origin="WN"
           definition="..." convenience="1.0">
      <synset>class#4,course of instruction#1,
              course of study#2,course#1</synset>
    </sense>
  </def-category>
  <def-category name="Department"
                short="university department"
                isa="EducationOrganization">...
  </def-category>
  <def-category name="University" short="university"
                isa="EducationOrganization">
    <sense name="University" no="3" origin="WN"
           definition="..." convenience="1.0">
      <synset>university#3</synset></sense>
  </def-category>
  <def-category name="Program" short="program"
                isa="Information">
    <sense name="Program" no="4" origin="WN"
           definition="..." convenience="1.0">
      <synset>course of study#1,curriculum#1,program#4,
              syllabus#1</synset></sense>
  </def-category>
  <def-category name="ResearchGroup"
                short="research group"
                isa="EducationOrganization">
    <sense name="ResearchGroup" no="0" origin="TECH"
           definition="" convenience="1.0">
      <synset>research group#0</synset></sense>
  </def-category>
  <def-category name="Activity" short="activity"
                isa="HumanActivity">...
  </def-category>
  <def-category name="Work" short="work"
                isa="Activity">...
  </def-category>...
</ontology>
```

Figure 3: Extract of the terminology oriented ontology concerning the american university

$$P(T_i) = \sum_{i=1}^{p} (M_{i,j}) \tag{2}$$

HTML marker description	HTML marker	Weight
Document title	<TITLE></TITLE>	10
Keyword	<meta name="keywords" ... content=...>	9
Hyper-link	<A HREF=...></A>	8
Font size 7	<FONT SIZE=7></FONT>	5
Font size +4	<FONT SIZE="+4"></FONT>	5
Font size 6	<FONT SIZE=6></FONT>	4
Font size +3	<FONT SIZE="+3"></FONT>	4
Font size +2	<FONT SIZE="+2"></FONT>	3
Font size 5	<FONT SIZE=5></FONT>	3
Heading level 1	<H1></H1>	3
Heading level 2	<H2></H2>	3
Image title	<IMG ... ALT="...">	2
Big marker	<BIG></BIG>	2
Underlined font	<U></U>	2
Italic font	<I></I>	2
Bold font	<B></B>	2
...	...	...

Table 1: Higher coefficients associated with HTML markers

Table 2 shows some results extracted from an experiment on a Web page. Terms are sorted according to the weighted frequency coefficient.

4.2 Page concepts determination

During the term extraction process, well-formed terms and their weighted frequency coefficient were respectively extracted and calculated. The well-formed terms are different forms representing a particular concept (for example “chair”, “professorship”...). In order to determine not only the set of terms included in a page but also the set of concepts in a page, a thesaurus is used. Our experiments use the WordNet thesaurus ([23]). The process to generate candidate concepts is quite simple: from extracted terms, all candidate concepts (all senses) are generated using a thesaurus. A sense is represented by a list of synonym (this list is unique for a given concept). Then for each candidate concept, the representativeness is calculated according to the weighted frequency and the cumulative similarity of the concept with the other concepts in the page. This last one is based on the similarity between two concepts.

We first define the similarity measure between two concepts which makes it possible to evaluate the semantic distance between these two concepts. This measure is defined relatively to a thesaurus and to the hypernyms relationship. In our context, we use the similarity measure defined by [29]. They propose a similarity measure related to the edge distance in the way it takes into account the most specific subsumer of the two concepts, characterizing their commonalities, while normalizing in a way that accounts for their differences. Their measure is shown in formula 3 where c is the most specific subsumer of c_1 and c_2, $depth(c)$ is the number of edges from c to the taxonomy root, and $depth_c(c_i)$ with i in $\{1, 2\}$ is the number of edges from c_i to the taxonomy root through c.

Terms	Weighted frequency
uw	1.00
cse	0.59
uw cse	0.45
computer	0.41
university	0.37
seattle	0.30
article	0.30
science	0.26
research	0.24
professor	0.24
...	...
computer science	0.18
...	...
university of washington	0.16
...	...
program	0.15
...	...
news	0.12
...	...
information	0.09
...	...
message	0.01
...	...

Table 2: Extracted terms and their weighted frequency (sorted according to the weighted frequency). Results coming from http://www.cs.washington.edu/news/

$$sim(c_1, c_2) = \frac{2 * depth(c)}{depth_c(c_1) + depth_c(c_2)} \tag{3}$$

This measure performs a little worse than the Resnik's measure ([26]) but better than the traditional edge-counting measure (see related works for more details).

For evaluating the relative importance of a concept in a page, we define its cumulative similarity. The cumulative similarity measure associated with a concept in a page, noted $\widehat{sim}$, is the sum of all the similarity measures calculated between this concept and all the other concepts included in the studied page. In this formula, a specific concept is unified with the corresponding synset (set of synonyms) in WordNet. The measure is shown in formula 4[3], where l_k synsets are associated with a term T_k, and there are m terms in the studied Web pages.

$$\widehat{sim}(synset_i(T_k)) = \sum_{j \in [1,k-1] \cup [k+1,m]} \sum_{l=1}^{l_j} sim(synset_i(T_k), synset_l(T_j)) \tag{4}$$

[3] $\widehat{sim}(synset_i(T_k))$ is normalized

In this calculus, all similarities are not been taken into account in order to discriminate the results: a threshold is applied. Finally, we determine a representativeness coefficient which determines the representativeness of a concept in a document. The coefficient is a linear combination of the weighted frequency and of the cumulative similarity of a concept (formula 5)[4]. This coefficient is the major one to qualify answer to a request. The empirical values for α and β are respectively 2 et 1.

$$representativeness(synset_i(T_k)) = \frac{\alpha * F(synset_i(T_k)) + \beta * \widehat{sim}(synset_i(T_k))}{\alpha + \beta} \quad (5)$$

The table 3 shows the effect of the representativeness on the concepts order (terms found in the page are in bold font). Some concepts are higher in the table 3 than in the table 2. For instance, news#1 (weighted frequency 0.12, representativeness 0.51) or information#1 (weighted frequency 0.1, representativeness 0.59). This is a good result for a page related to a news page. If we analyse the result more in details, the concepts: news#4 and news#2 have a representativeness equal to 0.49. This is not very different from the degree of news#1 which is equal to 0.51. The explanation is that Wordnet includes too much fine-grained sense distinctions. In fact, in the thesaurus, the three previous concepts have all the same subsumers. Then, an automatic process cannot distinguish these three concepts. Wordnet was built by linguist and is not always effective in NLP [25].

5 Associating concepts and synsets

At this point, we have on the one hand a terminology oriented ontology and on the other hand candidate concepts with their representativeness coming from HTML pages. In the next step, candidate concepts are matched with concepts of the ontology. If a concept is in the ontology and in a Web page, the URL of this page and its representativeness are added to the ontology.

To evaluate the appropriateness of an ontology according to a set of HTML pages, five typical coefficients are calculated. These coefficients are normalized. The first four coefficients define:

- the rate of concepts directly involved in HTML pages, called *the Direct Indexing Degree or DID*;
- the rate of concepts indirectly involved in HTML pages (calculated by the way of the generic/specific relationship), called *the Indirect Indexing Degree or IID*;
- the rate of pages concerned with the ontology concepts, called *the Ontology Cover Degree or OCD*, which gives the number of Web pages that involve at least one concept of the ontology;
- the Mean of the Representativeness of the candidate Concepts (MRC).

These coefficients (DID, IID, OCD, MRC) are evaluated for different thresholds applied on the representativeness (0 to 1 with a step equals to 0.02). For each coefficient its weighted

[4]The representativeness is normalized. $F(synset_i(T_k)$ is the normalized sum of all the weighted frequency related to $synset_i(T_k)$.

Concepts	Weighted frequency	Representativeness
uw#0	1.0	1.0
award#2, accolade#1, honor#1, honour#2, laurels#1	0.20	0.7
computer#1, data processor#1, electronic computer#1, information processing system#1	0.41	0.68
information#1, **info**#1	0.1	0.59
cse#0	0.59	0.59
university#2	0.37	0.58
course of study#1, **program**#4, curriculum#1, syllabus#1	0.15	0.53
calculator#1, reckoner#1, figurer#1, estimator#1, **computer**#2	0.41	0.51
news#1, **intelligence**#4, tidings#1, word#3	0.12	0.51
news#2	0.09	0.49
news#4	0.09	0.49
voice#6	0.01	0.51
voice#2, vocalization#1	0.01	0.51
message#2, **content**#2, subject matter#1, substance#6	0.01	0.51
language#1, linguistic communication#1	0.01	0.51
article#3, clause#2	0.30	0.5
submission#1, **entry**#4	0.01	0.5
subject#1, topic#1, theme#1	0.01	0.5
university#3	0.37	0.42
...	...	...

Table 3: Extracted concepts after the calculus of the representativeness degree (sorted according to the representativeness). Results come from http://www.cs.washington.edu/news/

mean (WM) is calculated. For instance, formula 6 presents the calculus of the weighted mean for the direct indexing degree (DID).

$$\overline{DID} = \sum_{i=0}^{1} (i * DID_i) \tag{6}$$

This calculus privileges the concepts which are more representative of the pages. A representative ontology of a site has the weighted mean nearly equal to 1. This evaluation depends on the thesaurus used because it depends on the used relationships. Finally, the global evaluation of the indexing process (OSAD: Ontology-Site Adequacy Degree) is a linear combination of these weighted means. Currently, the coefficients are evaluated in an experimental way. The equation 7 gives the present evaluation where s is a Web site and o an ontology. The experiment shows that a value of 0.3 for the representativeness gives good results. Below this threshold, too many concepts with a low representativeness are kept. For this threshold, the discrimination of concepts is relatively effective (the larger the Web pages are, the more effective is the process).

$$OSAD_{s,o} = \frac{\overline{IID}_{s,o}}{2} + \overline{DID}_{s,o} + 2 * \overline{OCD}_{s,o} + 2 * \overline{MRC}_{s,o} \tag{7}$$

The figure 4 presents indexing results related to the Web site: “http://www.cs.washington.edu/” (1315 HTML pages). This is the site of the department of computer science of the washington university. It was chosen because of its a priori adequacy with our ontology. However, the Ontology-Site Adequacy Degree (OSAD) is not very high (56%). The explanation is that the used ontology (the SHOE ontology with some extensions and modifications) does not cover all the studied domain. For instance, the studied site has numerous personal Web pages which are rarely indexed by the ontology. Figure 5 presents an extract of the structured index.

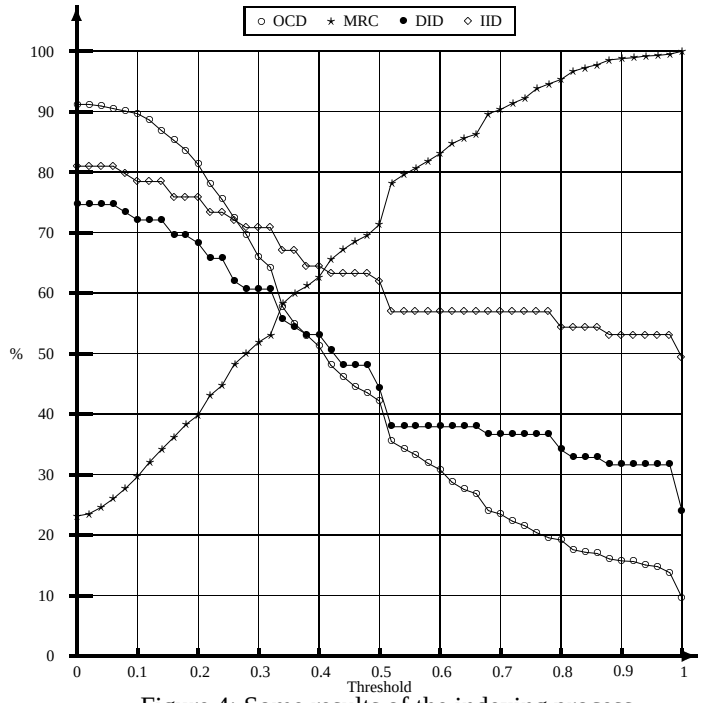


Figure 4: Some results of the indexing process

The indexing process can highlight concepts, which do not match with concepts of ontologies. In this case, we may search for ontologies related to this index. In the future, we will be able to start again the indexing process when the content of the site evolves or when ontologies are updated. This process can only be executed on modified pages.

The evaluation process enables us to evaluate the adequacy between the pages of the site and the ontology and thus to adopt various strategies depending on the coefficients value:

1. the coefficients are correct: the structured index is kept and exploited;
2. the coefficients are not correct:
   (a) the pages which are not suitable are deleted (the OCD and/or the MRC coefficient are low);
   (b) the ontology is updated (the DID coefficient is low);
   (c) a new ontology is chosen and the index is built again (the whole set of coefficients is low);

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE ontology SYSTEM "http://.../onto.dtd">
<ontology id="university-ont" version="3.0" description="">
 <def-category name="University" short="university"
               isa="EducationOrganization">
   <sense name="University" no="3" origin="wn" convenience="1.0">
     <synset>university#3</synset>
     <page name="http://www.cs.washington.edu/info/contact/"
           frequence="0.5" representativeness="0.4"/>
     <page name="http://www.cs.washington.edu/info/aboutus/"
           frequence="0.54" representativeness="0.49"/>
     <page name="http://www.cs.washington.edu/education/courses/590m/"
           frequence="0.4" representativeness="0.4"/>
     <page name="http://www.cs.washington.edu/outreach/"
           frequence="0.28" representativeness="0.34"/>
     <page name="http://www.cs.washington.edu/mssi/"
           frequence="0.5" representativeness="0.43"/>
     <page name="http://www.cs.washington.edu/general/overview.html"
           frequence="0.87" representativeness="0.81"/>
     <page name="http://www.cs.washington.edu/education/courses/599/"
           frequence="0.4" representativeness="0.35"/>
     <page name="http://www.cs.washington.edu/workforce/tnt/"
           frequence="0.25" representativeness="0.35"/>...
    </sense>
 </def-category>
 <def-category name="Department" short="university department"
               isa="EducationOrganization">
   <sense name="Department" no="1" origin="wn" convenience="1.0">
     <synset>department#1,section#11</synset>
     <page name="http://www.cs.washington.edu/education/courses/444/"
           frequence="0.29" representativeness="0.31"/>
     <page name="http://www.cs.washington.edu/lab/facilities/la2.html"
           frequence="0.5" representativeness="0.41"/>
     <page name="http://www.cs.washington.edu/ARL/"
           frequence="0.21" representativeness="0.32"/>
     <page name="http://www.cs.washington.edu/"
           frequence="0.33" representativeness="0.37"/>
     <page name="http://www.cs.washington.edu/desktop_refs.html"
           frequence="0.5" representativeness="0.43"/>
     <page name="http://www.cs.washington.edu/news/jobs.html"
           frequence="0.44" representativeness="0.46"/>
     <page name="http://www.cs.washington.edu/admin/newhires/faq.html"
           frequence="0.29" representativeness="0.32"/>
     <page name="http://www.cs.washington.edu/info/videos/index.html"
           frequence="0.39" representativeness="0.38"/>
     <page name="http://www.cs.washington.edu/affiliates/corporate/"
           frequence="0.5" representativeness="0.51"/>...
   </sense>
 </def-category>...
</ontology>
```

Figure 5: Extract of the SHOE ontology concerning the american universities

6 Exploitation of our approach for query answering

Most of search engines use simple keywords to index web pages. Queries are often made up of a list of keywords connected by logical operator (“and”, “or”...). In our context, we use the terminology oriented ontology and the structured index in order to improve the query answering process. Queries are not only processed at the terminological level but also at the conceptual level. This approach provides several improvements:

1. a user’s query is expanded: terms are transformed into concepts;
2. logical operators have a richer semantics than in the simple keywords world;
3. the answers are more suitable to a user’s query.

The query expansion is thus improved by the use of ontologies. Often, when a user proposes a query which contains terms connected by logical operators, these terms are often ambiguous. In our approach, terms are replaced by their associated concepts. The candidate concepts are first selected in the ontology. Then, the other concepts of the query and the logical operators are studied. Finally, if a term is still associated with several candidate concepts, the user’s assistance is required. If set of query terms are not associated with any concepts at the end of this process, they are regarded as not relevant for the site. According to the logical operator, either they are suppressed from the query or the query has no response.

The conceptually expanded query can be exploited to seek pages corresponding precisely to its content. The ontology makes it possible to improve the interpretation of the used logical operators. Currently, in one hand, the “and” and “or” operators have the same interpretation as in the traditionnal keywords approach. In the other hand, the “no” and “near” operators have a different semantics. For a query containing a “no” operator, we add to the concerned concept, all the concepts which are more specific than this concept according to the “isa” relationship. So, all pages containing these concepts will be rejected. The “near” operator is not related to the distance between words (number of words between two words) as in the classical approach. But, it is related to a semantic distance between concepts according to the similarity measure [29] used to calculate the representativeness coefficient. In our context, the “near” operator becomes an unary operator and makes it possible to add to the query all the concepts semantically connected to the targeted concept and in its neighbourhood.

7 Related works

Our choices differ from related works especially from work on annotation of Web page like KA2 ([9], [2]), SHOE ([21]) or WebKB([22]). These two projects annotate manually Web pages using semantic tags. SHOE proposes a set of *Simple HTML Ontology Extension* to annotate Web pages with ontology-based knowledge concerning page contents. In this context, an agent can use this knowledge to manage effectively information requests.

In all the cases, the goal is to use semantic information to improve the information retrieval. However, in these approaches, annotations are strongly linked to document. The author of pages progressively indicates handled knowledge where it appears. The problem is that any modification or new generation of the pages requires to remake entirely or partly the annotations. Nevertheless, the precision of this process is extremely fine. Moreover, the methods based on annotation are manual or semi-manual (an user interface helps the user to

annotate the document [16]). Therefore, they are very time expensive and can be carried out only by specialists ([15]).

However, this manual process is time expensive, complex, and information and knowledge are mixed. The information management difficulty is thus increased ([15]). In addition, semantically annotated documents are not today and perhaps may be never available on the Web. These two projects work on restricted domain and scaling up to the entire Web is a titanic task ([15]). Moreover, in this context, all Web page builders have to accept to annotate their own pages. The consensus needed by this protocol is far to be widely admitted and is at the opposite of the Web philosophy. Another project is the "WebKB" project ([22]). It proposes another manual process to annotate Web pages using an ontology represented with a conceptual graph ([27]), which is built using a linguistic thesaurus. Even if the used language is different from the two previous projects, annotations are also included in the HTML pages. Moreover, the thesaurus is only used to extend the ontology. It is not used to automatically index natural language documents.

Like in OntoSeek project ([14]), our approach adds linguistic attributes to ontologies using the WordNet thesaurus to improve our semi-automatic Web site knowledge discovery. Guarino calls this process a *disambiguation process*. However, the manual process OntoSeek uses ontologies not to define the knowledge of a Web site but to find user's data in a large classical database of Web pages. Another project proposes a similar process: the Mikrokosmos project ([25]) to provide a knowledge base for machine translation process. This process is another semi-automatic process (the user can improve manually the disambiguation results). It studies several heuristics. The most important are an hierarchical heuristics and a similarity heuristics. The hierarchical heuristics uses the generic/specific relationship in the ontology and the relationship of hyperonymy in the thesaurus. For [25], the hierarchical heuristics seems to be the more effective to select senses. Therefore, we choose to use this heuristics and to improve it.

Some projects of the KDD (Knowledge Discovery in Databases) community are interested by extracting knowledge from Web sites. [8] apply techniques of KDD to keywords which are attached to the documents and which are then regarded as attributes. These mining techniques use statistical analysis to discover association rules and interesting patterns over keywords distributions and associations. Other researchers [18] use terms automatically extracted from documents to characterize the document and to find associations which connect the terms to the documents. Another approach is to apply KDD techniques after the use of information extraction techniques, which transform information located in texts into a structured database [6]. Other approaches [20] mixe NLP techniques and KDD techniques to extract automatically information from documents. They do not use keywords as attribute but use concepts which are acquired by the way of a thesaurus. The approach of the last authors seems the most interesting because they do not work any more with simple keywords but with the concepts included in documents. Compared to KDD techniques like [20], we also work on conceptual level instead on the simple keywords level. But we take the option to have linguistic processing much finer and especially we privilege an a priori knowledge on the studied domain (one or several ontologies). [20] use a priori knowledge on the studied domain (a thesaurus) exclusively to extract the concepts of the pages. In our approach, the concepts are also extracted from the pages using a thesaurus, but the indexing process itself is also based on an ontology of the domain. [24] asserts besides that for an effective extraction of knowledge, a priori knowledge on the studied domain (for example ontologies) is essential.

Many measures of similarity are defined in related works. For [19], the information shared

by two concepts is indicated in an "isa" taxonomy by the most specific concept that subsumes them. The semantic similarity of two concepts in a taxonomy is the distance between the nodes corresponding to the items which are compared (edge-counting). The shorter the path from one node to another is, the more similar they are. Given multiple paths, one takes the length of the shortest one.

A widely acknowledged problem ([26]) with this approach is that it relies on the notion that links in the taxonomy represent uniform distances (but it is most of the time false). [26] describes an alternative way to evaluate semantic similarity in a taxonomy based on the notion of information content. All links in a taxonomy are weighted with an estimated probability (concept occurrences in corpora), which measures the information content of a concept. The main idea is: the more concepts share information, more similar they are. The information shared by two concepts is indicated by the information content of the concepts that subsumes them in the taxonomy. The probability P of a concept c is based on the probability associated with the concept plus the probability associated with all its descendant concepts. $P(c)$ is then used to calculate the information content of a concept c which is equal to $-log(P(c))$.

8 Conclusions

In this paper, we have presented a semi-automatic process to index a Web site by its content. This process builds a structured index coming from an ontology and pages of a Web site. After the construction of a flat index where all terms have a weighted frequency, we determine candidate concepts associated with these terms. For each concept, a representativeness coefficient is calculated. Finally, the most representative concepts in a Web page are selected, and those which belong to the ontology are kept. The final structured index is organized according to the ontology. With each ontology concepts a set of Web pages is associated from where the potential concepts were extracted.

This process comprises a number of advantages on the traditional indexing methods (only based on keyword retrieval) and even on the methods of Web site annotation:

1. selected pages contain not only the keywords but also the required concepts ;
2. these concepts are representative of the topics treated in selected pages ;
3. terms which are responsible of the page selection are not always those of the request but can be synonyms ;
4. pages can comprise not only the required concepts but also more specific ones ;
5. the importance of a concept depends not only on its term frequency but also on the HTML markers which describe it and on its relations with the other concepts of the page...

The indexing process can be used not only for retrieving information but also for valueing the appropriateness of a Web site with regard to a domain or a knowledge. This latter case enables us to classify a Web site in a hierarchical index of a classical search engine (Yahou !, Excite...). Note that such hierarchies can be themselves considered as general ontologies ([17]).

Currently, other Web sites on american universities are indexed in order to compare their results to those of the Washington university. In order to improve the indexing results, we may also improve the coverage degree of the ontology on our studied domain. We study also

other relationships than the generic/specific relationship in order to improve the process of concepts extraction. We have developed a measure according to the composition relationship, but we must also evaluate it in an experimental way.

The results presented in this paper can be used in various applications. They are currently being incorporated within the Bonom Multi-agent system ([5], [4]) to search for relevant information on the Internet. The system involves different types of agents among which "site agents" which encapsulate information sources. The methods we propose are implemented within the site agents. They greatly improve the site analysis process and the query answering process.

References

[1] N. Ashish and C. A. Knowblock, "Semi-Automatic Generation Internet Information Sources", In 2nd IFCIS Conference on Cooperative Information Systems (CoopIS), Charleston, SC, 1997.

[2] V. R. Benjamins, D. Fensel, A. Gomez-Perez, S. Decker, M. Erdmann, E. Motta, and M. Musen. "Knowledge Annotation Initiative of the Knowledge Acquisition Community KA2". In Proceedings of the 11th Banff knowledge acquisition for knowledge-based system workshop, Banff, Canada, 1998, pp. 18-23.

[3] E. Brill, "Transformation-based error-driven learning and natural language processing: a case study in Part-of-speech Tagging". Computational Linguistics, vol. 21, 1995, pp. 543-565.

[4] S Cazalens, E Desmontils, C Jacquin, and P Lamarre, "A Web Site Indexing Process for an Internet Information Retrieval Agent System", International Conference on Web Information Systems Engineering (WISE'2000), IEEE Computer Society Press, Hong-Kong, 19-20 June, 2000, pp. 245-249.

[5] S. Cazalens and P. Lamarre, "An organization of Internet agents based on a hierarchy of information domains", In Proceedings MAAMAW, Yves Demazeau and Francisco J. Garijo editors, may 2001

[6] J. Cowie and W. Lehnert. "Information extraction". In Communications of the ACM, number 1, volume 39, january 1996.

[7] B. Daille, "Approche mixte pour l'extraction de terminologie : statistique lexicale et filtres linguistiques", PHD Thesis, Paris 7, 1994.

[8] R. Feldman and I. Dagan. "Knowledge discovery in textual databases (KDT)". In First international conference on knowledge discovery (KDD'95), Montreal, august 1995.

[9] D. Fensel, S. Decker, M. Erdmann, and R. Studer. "Ontobroker: Or How to Enable Intelligent Access to the WWW". In Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'98), Banff, Canada, 1998.

[10] J. Gamet. "indexation de pages web". Rapport de DEA informatique, université de Nantes, 1998.

[11] A. Gomez-Perez. "Développements récents en matière de conception, de maintenance et d'utilisation des ontologies". In Proceedings of colloque Terminologie et intelligence artificielle de Nantes, 10-11 mai 1999, revue terminologies nouvelles, pp. 9-20.

[12] T. Gruber, "A Translation Approach to Portable Ontology Specification". In Knowledge Acquisition journal, vol 5, pp 199-220, 1993.

[13] N. Guarino. "Some Organizing Principles for a Unified Top-Level Ontology". In Spring Symposium series on ontological engineering, pp 57-63, 1997

[14] N. Guarino, C. Masolo, and G. Vetere, "OntoSeek: Content-Based Access to the Web", IEEE Intelligent Systems and Their Applications, Elsevier Science, 14(3), 1999, pp. 70-80.

[15] J. Heflin, J. Hendler, and S. Luke, "Applying Ontology to the Web: A Case Study", In International Work-Conference on Artificial and Natural Neural Networks (IWANN), 1999.

[16] J. Kahan, M. Koivunen, E. Prud'Hommeaux and R.R. Swick "Annotea: An Open RDF Infrastructure for Shared Web Annotations", In proceedings of the WWW'10 conferences, Hong Kong 2001

[17] Y. Labrou and T. Finin, “Yahoo! as an Ontology - Using Yahoo! Categories to Describe Documents”, In Proceedings of CIKM’99, Kansas City, MO, Oct. 1999, pp. 180-187.

[18] S. Lin and al. “Extracting classification knowledge of internet documents with mining term associations: a semantic approach”. In International ACM-SIGIR conference on research and development in information retrieval (SIGIR-98).

[19] J. H. Lee, M. H. Kim, and Y. J. Lee, “information retrieval based on conceptual distance in IS-A hierarchies”, journal of documentation, 49(2) , 1993, pp 188-207.

[20] S. Loh, L.k. Wives and J Palazzo M de Oliveira. “Concept-based knowledge discovery in texts extracted from the web”. In journal SIGKDD explorations, number 1, volume 2, pp 29-39, 2000.

[21] S. Luke, L. Spector, and D. Rager. “Ontology-Based Knowledge Discovery on the World-Wide-Web”. In Proceedings of the workshop on Internet-based information system, AAAI’96, Portland, Oregon, 1996.

[22] P. Martin and P. Eklund, “Embedding Knowledge in Web Documents”, In Proceedings of the 8th International World Wide Web Conference, Toronto, Canada, May 11-14, 1999 (http://www8.org).

[23] G. A. Miller, “WordNet: an Online Lexical Database”, International Journal of Lexicography, 3(4), 1990, pp. 235-312.

[24] D. Mattox, L. Seligman and K. Smith, “Rapper: a wrapper generator with linguistic knowledge”, In ACM workshop on information and data management, 2000.

[25] T. O’Hara, K. Mahesh, and S. Niremburg, “Lexical Acquisition with WordNet and Microkosmos Ontology”, workshop on “usage of WordNet in natural language processing systems”, 8 pages, Coling-ACL’98

[26] P. Resnik, “Semantic similarity in a taxonomy : an information-based measure and its application to problems of ambiguity in natural language”, journal of artificial intelligence research, 11, July 1999, pp. 95-130.

[27] J. F. Sowa, “Conceptual Structures, Information Processing in Mind and Machine”, Addison Wesley Publishing Company, 1984

[28] W3C. “Extensible Markup Language (XML) 1.0”. W3C Recommendation, Reference: REC-xml-19980210, 10 February 1998, http://www.w3.org/TR/REC-XML

[29] Z. Wu and M. Palmer, “verb semantics and lexical selection”, In Proceedings of the 32nd annual meeting of the association for computational linguistics, Las Cruses, New Mexico, 1994

[30] B. Yuwono and D. L. Lee. “WISE: A World Wide Web Resource Database System”. IEEE Transactions on Knowledge and Data Engineering, 8(4), 1996, pp. 548-554.

A semantic model for specifying data-intensive Web applications using WebML

Sara Comai Piero Fraternali
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza L. Da Vinci, 32
I-20133 Milano, Italy
comai,fraterna@elet.polimi.it

Abstract. WebML (Web Modelling Language) is a language for the design of data-intensive Web sites. It is supported by visual tools allowing the definition of the conceptual data organization and of the pages and links of the actual hypertext(s) which constitute a Web application. In this paper we describe a semantic model for WebML hypertexts by means of Statecharts. Statecharts provide a formal description of the clicking behavior and page data fill of WebML applications. The proposed semantic model has guided the implementation of the WebML runtime and the construction of advanced specification checking functions embedded in the WebML design tools. In particular, developers are supported in the identification of design and runtime problems caused by non-determinism, racing conditions and deadlocks.

1 Introduction

Web applications have spread in every sector of the human activity, well beyond the boundaries of document-oriented systems, for which the Web has been initially conceived.

The enormous demand for Web-enabled applications, both novel or resulting from the re-engineering of existing systems, coupled to the chronic lack of skilled IT personnel, puts forth a dramatic request for better software engineering practices, similar to those adopted in more mature software fields, like database and object-oriented development.

For improving productivity, a broader coverage of the tasks of Web site development is imperative, because the vast majority of Web development tools available on the market still concentrate only on design and implementation, paying little attention to requirement analysis and conceptual modelling [7]. Therefore, implementing and maintaining a large Web site is still a very human-intensive and error-prone activity, which does not benefit from the availability of a formal development process, supported by modelling notations and CASE tools.

To cope with these requirements, the research community has proposed several approaches for the so-called model-driven design of Web sites [1, 5, 8, 9, 11], which share the idea of leveraging semi-formal notations to express the data structure and hypertext topology of a Web site and of using conceptual-level specifications to drive the design and implementation.

WebML [3] is one of the proposals for the conceptual specification and automatic implementation of Web sites. A WebML specification is directed labelled graph, internally represented as an XML document written according to the WebML DTD, which describes the topology of one or more hypertexts conceived to publish information on a set of application objects. The WebML language is backed by a suite of software tools, which transforms visual WebML specifications into server-side page templates and database queries, which implement the desired Web site.

Differently from previous proposals, which were mostly introduced informally and by examples, WebML anchors Web site specifications to a sound formal basis, by associating a formal semantics to the semi-formal visual notation. This paper introduces WebML's formal semantics, which is based on the use of STATECHARTS [10] to express the dynamic behavior of a Web site.

As a consequence of establishing a formal model, Web site specifications acquire an unambiguous meaning and lend themselves to automatic checking for correctness or desired properties. Moreover, the formal semantics can be used as a yardstick to evaluate the correctness of CASE tools generating running Web sites from WebML specifications, because the runtime behavior of the generated site must obey the expected behavior expressed by the formal semantics.

2 Overview of WebML

A WebML specification consists of two major components:

- The *structure model*, describing the conceptual organization of the application data;
- One or more *hypertexts* (*site views* in the WebML jargon) defined on top of the structure model, which express the organization and linking of pages used to publish the application data.

The approach adopted by WebML is data-driven: first the structure of the data is described, then, on the basis of such structure, the hypertext is defined, as explained in the following subsections. For further details about the syntax of WebML the reader may refer to [3] and to the Web site `http://webml.org`.

2.1 Structure model

The structure model describes the conceptual data organization, and is compatible with the Entity-Relationship data model, used in conceptual database design, and with UML class diagrams, used in object-oriented modelling. The fundamental elements of the structure model are *entities*, defined as containers of data elements, and *binary relationships*[1], defined as semantic connections between pairs of entities. Entities have attributes representing the properties of the real world objects and relationships are characterized by named relationship roles (i.e., the two directions in which a binary relationship can be traversed) and cardinality constraints associated to each role.

[1]WebML presently supports only binary relationship without attributes; work on supporting content units defined over generalized n-ary relationships and relationship attributes is ongoing.

Example I: Figure 1 shows a simple structure schema for the publication of an hypertext describing data about books: the rectangles in the graph represent entities, while edges represent relationships (for brevity, relationship roles names are omitted). In the example, each book is written by one or more authors and has a unique publisher; moreover, each book may be associated with zero or more reviews.

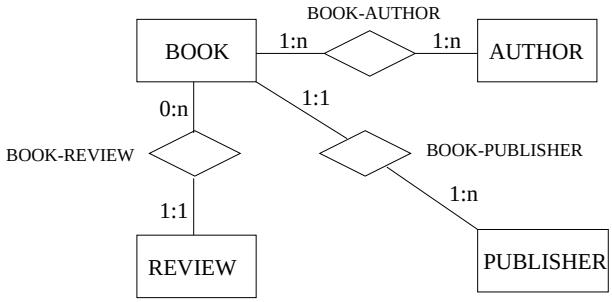


Figure 1: Example of structure schema

2.2 Hypertext

A WebML *hypertext* consists of a set of *pages*, depicted as rectangles, connected by *non-contextual links*, represented by oriented arcs. The content of a page is expressed by means of *content units*. Different kinds of unit are provided by WebML, denoted by different symbols. Units may be connected by *contextual links*, also graphically depicted by means of oriented arcs (See Figure 3). We describe first content units; then, we clarify the use of contextual and non-contextual links.

Units publish information about the objects of the structure schema: each unit is defined over a *master object*, an entity or a relationship role[2], which gives content to the unit.

WebML offers six predefined content units to assemble read-only hypertexts (additional units are available for content management applications):

- *Data units*: they are used to publish a set of attributes of a single object (e.g. the data of a single book). The graphical representation of WebML data units is shown in Figure 2.a.
- *Index units*: they are used to represent sorted lists of objects, where each object is denoted by some representative attributes (e.g. an index of authors may show the first name and last name of each author). Index units are typically linked to a data unit, which shows the details of the object selected from the index (e.g. the data of the selected author). The graphical representation of index units is shown in Figure 2.b.
- *Multidata units*: they show multiple objects together, by repeating the presentation of several, identical data units[3] (e.g., all the books written by an author). See Figure 2.c for the graphical representation.

[2] A relationship role univocally determines a source entity and a destination entity, based on the direction in which the relationship is considered.

[3] In the following sections, multidata units will be treated as a finite set of data units, and therefore will not be considered explicitly.

- *Scroller units*: they provide the commands to scroll over an ordered set of objects. They are generally connected to a data unit showing the current item of the sequence. The graphical representation is shown in Figure 2.d.
- *Filter units*: they allow the user to specify search criteria by means of a search form. Typically, a filter unit is connected to an index unit showing the result of the search (e.g. the user inserts the category of a book, and the list of books belonging to this category is shown). The graphical representation of filter units is shown in Figure 2.e.
- *Direct units*: they associate one object to a single other object along a one-to-one or many-to-one relationship (possibly the identity relationship). They are generally connected to a data unit showing the unique target of the one-to-one or many-to-one relationship (e.g., the data of a book may be connected through a direct unit to the data of its unique publisher). The graphical representation is shown in Figure 2.f.

Figure 2: Graphical representation of WebML units

Example II: Consider for example the hypertext depicted in Figure 3. It contains three pages: the home page, the books' index page and the book page. The home page is empty (we suppose that it contains only unmodeled, presentation-oriented content) and is connected by a link to the books' index page, which contains two units: a filter defined over books (BookFilter) allows one to search all books based on some keywords (e.g. with respect to their category), and is linked to an index unit (BookIndex), which represents the list of books matching the search criteria expressed in the filter unit. The books' index is connected to a data unit in a separate page (BookPage). This page contains several pieces of information, which are shown when the user clicks on an entry in the index of books: the data of the selected book (BookData), the data of its publisher (PublisherData), the index of its authors (AuthorIndex), and a scroller unit defined over the book's reviews (ReviewScroller), which allows the user to orderly browse the book's reviews, displayed one by one in a data unit (ReviewData). A direct unit (Book2Publisher) is interposed between the BookData unit and the PublisherData unit, to associate the book to its unique publisher.

An important difference exists between non-contextual links connecting pages and contextual links between units: the former are a mere navigational device used to change page, the latter imply the transportation of *navigation context* from the source to the destination unit. Navigation context is information passed from one unit to another one in order to make the second unit computable from the data in the structure layer. For example, in Figure 3 the link exiting the home page is a non-contextual link and does not carry any information; instead, the link between the BookIndex unit listing a set of books and the BookData unit showing the data of a particular book is contextual: it must carry the identifier of the book selected in the index, for the data unit to be computable. Note that, as shown in this example,

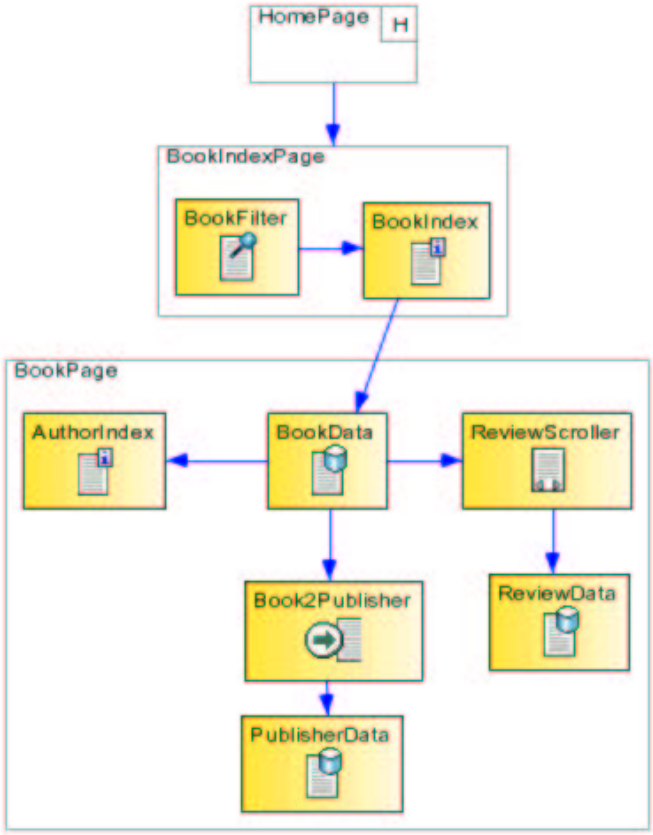


Figure 3: Example of WebML pages

when the source and the destination units of a contextual link belong to different pages, also navigation between pages is performed.

WebML units are both producers and consumers of navigation context. For example, an index unit typically produces the identifier of the object selected from the user; however, it may also consume context, e.g., to display a list of objects connected by a relationship to an input object. For example, in Figure 3 the AuthorIndex unit, listing the authors of a particular book, needs the OID of the current book to be computed: indeed, according to the schema of Figure 1, given the current book, the target objects of the relationship between book and author can be identified.

The following table illustrates the input and output context of the different WebML units.

Unit	Input parameters	Output parameters
Data unit	Selected instance (OID of the current instance)	Current instance
Index unit	Owner of the relationship[1], Optional predicate[2]	Selected item, Owner of the relationship[1]
Multidata unit	Owner of the relationship[1], Optional predicate[2]	Selected item (possibly all the items), Owner of the relationship[1]
Filter unit	Owner of the relationship[1], Optional predicate[2]	New Predicate, Owner of the relationship[1]
Scroller unit	Owner of the relationship[1], Optional predicate[2]	Selected item, Owner of the relationship[1]
Direct unit	Owner of the relationship	Target of the relationship [3]

[1] When a unit is defined over a relationship role, the OID of an instance of the source entity participating to the relationship (called the relationship's owner, in the WebML jargon) is required.

[2] When the unit is preceded by a filter unit, a predicate is passed to compute the result set of the search.

[3] The target of a one-to-one or many-to-one relationship is the unique object associated to the owner of the relationship.

As shown in the example of Figure 3, a WebML page typically contains several units linked in a network topology to produce the desired communication effect. In order to specify how the context is propagated along the chains of linked units, WebML permits the designer to declare links (both contextual and non-contextual) as *automatic* or *clickable*. The former are "automatically clicked" by the WebML runtime system, to propagate context from the source to the destination unit of the link even in absence of user's action. The latter do not exhibit such behavior, but the user must explicitly activate the link for context propagation to occur.

When links are automatic the output parameters of the unit wherefrom the link exits may need proper initialization: the output of an index or scroller unit is initialized to the first instance of the underlying entity or relationship; the output predicate of a filter unit is initialized to "true", to select all objects of the underlying entity or relationship.

For example, in Figure 3, when the BookPage is accessed, the OID of the book to be displayed is passed to the book data unit by its incoming contextual link. Then, propagation of context occurs inside the BookPage page. If all the links between units in BookPage are automatic, context information flows from unit to unit without the user's intervention: the OID of the selected book flows to the subsequent units, thus showing also the index of authors, the publisher's data, the first review and the scroller commands to access the other reviews. The first review is chosen by default by the system, which initializes the output parameter of the scroller unit.

Conversely, if the links exiting the book data unit are defined as clickable, when the BookPage is accessed only the data of the selected book are shown; then the user must click on the provided anchors (one for each link) to transfer the output context and see also the other data in the page. Notice the importance of the automatic links in practical applications: they allow to automatically display information bound to the current data. A more sophisticated example showing the use of automatic and clickable links will be presented in the next section.

3 Semantics of WebML

In the previous section we introduced the syntax and the main characteristics of WebML; now we describe its semantics.

Before introducing a formal description of the behavior of a dynamic Web site, we extend the hypertext of Figure 3 in order to show some particular behaviors and problems that highlight the benefits of having a semantic model.

The WebML specification of Figure 4 extends the previous hypertext with the authors' index page.

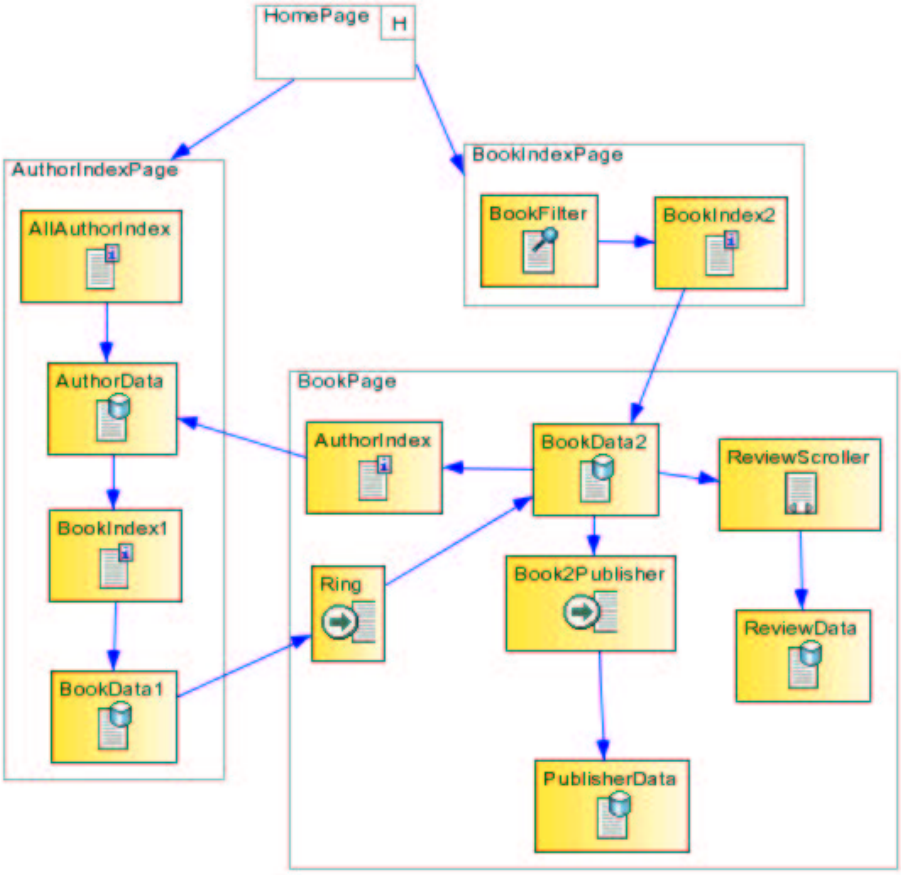


Figure 4: Example of a Web site

Let us carefully analyze such page. It contains several kinds of information to be shown: the index of all the authors (AllAuthorIndex), the data of a selected author (AuthorData), the list of the books of such author (BookIndex1) and the data of one book selected from such list (BookData1). Note that the two first links transport the identifier of the selected author, while the third link transports the identifier of the selected book. Depending on how the links between units are specified, i.e. automatic or clickable, this page behaves differently. Suppose that all the links between the units be automatic, i.e. the first click of every link is automatically done by the system without any intervention of the user: when this page is accessed it displays the list of all the authors, the data of the *first* author of the index (chosen as default by the system), the list of the books of such author and, finally, the data of the *first* book of the book index (chosen as default by the system). All these data are automatically shown. Then, the user may select a different author from the author index or a different book from the book index: in both cases all the data related to the units following the considered index change accordingly. That is, if a new author is selected from AllAuthorIndex the data of the new author are shown, together with his/her books and the data of the first of such

books; if a new book of the author is selected from BookIndex1 the data of the selected book are displayed. Notice how the automatic links allow to define default choices without leaving empty parts in the page.

Consider now the case where the links exiting the index units are defined as clickable and the link exiting the author data unit is still automatic: the behavior of the same page would be different, since the first choice of each index is not performed by the system, but for each index the system waits until the user clicks on an item. This means that, when the page is entered the first time it contains only the list of the authors; then, when the user selects one of such authors, his/her data together with the list of his/her books are shown; finally, when the user selects one of the books of the author also the data of the books are shown. Then, the user may e.g. select a new book from BookIndex1: as a consequence the data of the new selected book are displayed as in the previous case. Instead, if the user selects a new author from AllAuthorIndex, the data of the new author are displayed together with his/her books' list but no data about a particular book are shown until the user explicitly selects one item. Notice that in this case the page content changes, i.e., initially only one unit is populated, then after user's clicking two other units are filled and so on.

So, depending on the kinds of link (automatic or clickable) the behavior of the page is different, since information may be automatically displayed or not displayed at all, and page composition may change after user clicking. For complex pages containing several units the same page may have different configurations at runtime depending on the kinds of links and on user's behavior; such configurations can be properly described by a semantic model.

Let us consider another important aspect that need to be considered. From the authors' index page, it is possible to reach the page displaying further information about the selected book (a direct unit is used to represent the identity relationship, i.e., the current book itself) and from this page it is possible to go back to the authors' index page, by selecting one of the book's authors. When a page may be reached from different pages, the page must be correctly computed for every single access. When designing a Web site several pages could in fact be reused for displaying the same kind of information.

Let us focus on the authors' index page again: when it is reached from the book page the contextual link enters the second unit (AuthorData) of the chain. In this case the first unit in the chain (AllAuthorIndex) may cause some problems. The list of all the authors can always be displayed, independently of how the page is accessed, since this unit does not receive any input context. But what about its outgoing link? If it is clickable, the system waits until the user selects a new item: so, if the page is accessed from the book page the remaining part of the page is computed for the author selected in the book page and it is not changed until a new author is selected from the AllAuthorIndex. Instead, what does it happen if the outgoing link is automatic? Does the system automatically display the data of the first author of such index or the data of the author selected in the book page?

To answer questions like this we need a semantic model, describing the precise behavior of the hypertext, whose interpretation may become difficult for complex sites. Then, on the basis of the semantics the system can be actually implemented and the correctness of the specification can be automatically checked. To formalize the semantics we adopt Statecharts, which allow to easily describe any dynamic system behavior. Indeed, each page of the site can be represented as a state. Intuitively, when we navigate through the different pages we change state. We can change page by clicking on the anchors provided by the current page: the action of clicking represents the event which makes the system change its current state. In a similar

way, also the content of the pages may be represented by concurrent states, each representing the behavior of a single unit: the content of a unit is shown depending on the possible events automatically generated by the system (e.g. when there are automatic links to be followed) or by possible selections performed by the user. In the sequel we formally describe how to map a generic WebML specification into a Statechart describing its semantics. We first provide some preliminary definitions. Then, we define how to map pages into states and how to map units contained inside a page into concurrent states. Finally, we will see that this model allows to analyze the behavior of the system in critical cases, where for example non-determinism or racing conditions arise.

3.1 Preliminary Definitions

The concepts of a WebML hypertext introduced in the previous sections can be formally described as follows:

Definition 1: (**WebML hypertext**): a WebML hypertext is a triple (U, P, L) where U is a set of units, P is a set of pages, and L is a set of links. U, P and L are such that: 1) links in L connect either two pages in P or two units in U; 2) units in U are contained in pages in P; 3) one page in P is defined as the home page.

In the sequel we represent links between units with the pair (u_i, u_j) and links between pages with the pair (p_i, p_j).

Units of a page are classified based on the topology of the links that connect them:

Definition 2: (**Access, depending, and stand-alone units**) Let H=(U, P, L) be a WebML hypertext. Let $u \in U$ be a unit contained in page $p \in P$. Then, u is an *access unit* if it has incoming contextual links originating from outside of p; it is a *depending unit* if it has incoming contextual links originating from units inside p; it is a *stand-alone unit* if it has no incoming contextual links.[4]

We now introduce the variables and alphabets for events (E), conditions (C) and actions (A) needed for mapping WebML concepts to Statecharts:

Definition 3: (**Variables and E[C]/A alphabets**) Let H=(U, P, L) be a WebML hypertext. Let u_i (i=1 . . . n) be the units in U, l_i (i=1 . . . m) be the links in L and p_i (i=1 . . . q) be the pages in P. Then, we define the following variables, events, conditions and actions:

[4]Note that a unit may have multiple incoming links, and thus be both an access and a depending unit. The actual link used at runtime to access a page determines the role of the unit.

Type	Name	Description	NULL value
Variable	access_link_p_i	It refers to the contextual link through which page p_i has been accessed.	yes
Variable	recomputable_u_i	It is a boolean variable stating if the content of unit u_i can be re-calculated for display or not.	yes
Variable	input_context_u_i	It contains the input context of unit u_i.	yes
Variable	output_context_u_i	It contains the output context of unit u_i.	yes
Event	output_context_u_i_available (i=1...n)	It denotes that the content of unit u_i has been calculated and its output context is available in variable output_context_u_i.	—
Event	clicked_on_anchor_l_j (j=1...m)	It denotes that the user has clicked on the anchor corresponding to link l_j.	—
Condition	access_link_$p_k(u_i)$	It checks if there exists an access link entering unit u_i in page p_k.	—
Action	initialize_output_u_i (i=1...n)	It initializes all the output parameters of unit u_i.	—

As customary in Statecharts we use the polymorphic symbol ϵ to denote both the empty event, used to specify automatic transitions, and the empty action.

3.2 Page configuration

We first define how to map the pages of a generic WebML hypertext into a Statechart: given a WebML hypertext all the pages are mapped into states and all the links (both non-contextual and contextual) are mapped into transitions among such states as follows:

Definition 4: (**WebML hypertext Statecharts**) Let H=(U, P, L) be a WebML hypertext. Then, the corresponding *WebML hypertext Statecharts* is obtained as follows:

- For each $p_i \in P$ a top-level state S_p_i is created;
- For each non-contextual link $l_i = (p_j, p_k) \in L$ a transition from S_p_j to S_p_k is created with
  - E[C]/A=ϵ[$true$]/access_link_p_k:=NULL if the link is automatic,
  - E[C]/A=clicked_on_anchor_l_i[$true$]/access_link_p_k:=NULL if the link is clickable.
- For each contextual link $l_i = (u_t, u_v) \in L$ with $u_t \subset p_j$ and $u_v \subset p_k$, a transition from S_p_j to S_p_k is created with
  - E[C]/A=clicked_on_anchor_l_i[$true$]/output_context_u_t_available; access_link_p_k:=l_i if the link is clickable,
  - E[C]/A=output_context_u_t_available[$true$]/access_link_p_k:=l_i if the link is automatic.
- Page_p_{home} is the initial state.

Example III: The WebML hypertext of Figure 4 is mapped into the hypertext statechart shown in Figure 5: the four pages are mapped into four states and all the links among such pages are mapped into transitions. In particular, from the home page two non-contextual links

(transitions 1 and 2) depart, which are activated when the event of clicking on the corresponding anchors occurs. Since the links are non-contextual no access link is set for the following pages. From the authors' index page a link departs toward the book page: it is activated when the user clicks on the anchor of the link (transition 3) provided in correspondence of the book data unit (BookData1), setting the current link as active for the book page. This last operation is necessary when the page can be accessed through different links in order to consider the correct incoming link. This transition notifies also that the output context of BookData1 has been computed and is available to be used to compute the new page. Analogously, two contextual links, one from the books' index page to the book page (transition 4) and one from the book page to the authors index page (transition 5) are obtained.

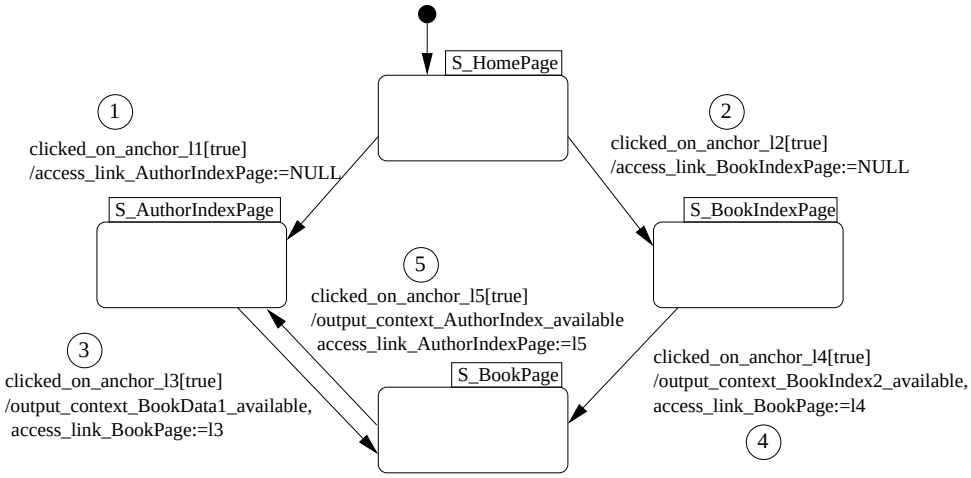


Figure 5: WebML hypertext statechart

3.3 Unit Configurations

Once the pages have been mapped into states, the content of each page can be described. Given a page of the WebML hypertext, the units in it are mapped into a set of concurrent states, each describing the behavior of a single unit.

Intuitively, each unit can be either in a *disabled* state, where no data are shown, or in an *enabled* state where its content is displayed according to the input context. At page entry, a unit is disabled by default. Then, one ore more transitions may lead to the enabled state. From this state one or more transitions are defined, either to go back to the disabled state or to re-enter the enabled state, possibly changing the unit content (see Figure 6). The kinds of events, conditions and actions of the transitions depend on on the fact that the unit is an access, a standalone or a depending unit.

Definition 5: (**WebML unit Statechart**) Let H=(U, P, L) be a WebML hypertext. Let $p \in P$ be a page containing one or more units and S_p be its corresponding state. Then, for each unit u_k contained in p the following states are introduced:

- A concurrent state S_u_k nested at the first level of S_p is created;
- A state $S_{disabled}\_u_k$ nested inside S_u_k is created and set as initial state;

- An state $S_{enabled}\_u_k$ nested inside S_u_k is created containing the entry action recomputable_u_k:=false;

A set of transitions between $S_{disabled}\_u_k$ and $S_{enabled}\_u_k$ are introduced as follows:

- If u_k is an *access unit*, then, for each contextual link $l_j = (u_j, u_k) \in L$ with $u_j \not\subset p$ a transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is added with

  E[C]/A=ϵ[output_context_u_j<>NULL AND access_link_p(u_k)]
  /input_context_u_k:=output_context_u_j;
  access_link_p:=NULL.

  The transition states that at page entry the output context of the source unit of the incoming link becomes the input context of the access unit.

- If u_k is a *standalone unit*, then a transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is added with E[C]/A=ϵ[$true$]/ϵ. The transition states that a standalone unit is automatically enabled at page entry.

- If u_k is a *depending unit*, then for each contextual link $l_j = (u_j, u_k) \in L$ with $u_j \subset p$

  – The following transitions or actions are added to the state of the *source* unit u_j, which feeds navigation context to the depending unit u_k:

    * A new ring transition on $S_{enabled}\_u_j$ is added with:
      E[C]/A=clicked_on_anchor_l_j[$true$]/ recomputable_u_j:=true, $j = 1, \ldots, n$.
      This transitions expresses that a unit feeding another unit inside the same page may need re-computation (this happens if there is a cycle of links leading back to the unit).
      If the outgoing link l_j is clickable action output_context_u_j:=NULL is added to all the other transitions entering $S_{enabled}\_u_j$; this expresses that for clickable links there is the need of cleaning the output context, when the destination unit is enabled.
      If the outgoing link l_j is automatic, the action initialize_output_u_j is added to all the other transitions entering $S_{enabled}\_u_j$. This expresses that for automatic links there is the need of properly initializing the output context, when the destination unit is enabled.
    * Action output_context_u_j_available is added as entry action in $S_{enabled}\_u_j$ to activate the depending units.
    * Actions output_context_u_j:=NULL; output_context_u_j_available are added to the transitions from $S_{enabled}\_u_j$ to $S_{disabled}\_u_j$. These transitions indicate that the unit cannot be computed and therefore also their depending units must be inhibited by setting the passed context to NULL.

  – The following transitions are defined for the depending unit u_k:

    * A transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is created with:
      E[C]/A=output_context_u_j_available[output_context_u_j<>NULL AND NOT access_link_p(u_k) AND recomputable_u_k] /input_context_u_k:=output_context_u_j.
      This transitions expresses that the depending unit is enabled when the output context of its feeding unit becomes available and is not null, the unit has not

been directly accessed from outside the page and is re-computable (i.e., its content has not already computed in the context propagation, e.g., due to link cycles).

* A ring transition on $S_{enabled}\_u_k$ is added having:
E[C]/A=output_context_u_j_available[output_context_u_j<>NULL AND recomputable_u_k]
/input_context_u_k:=output_context_u_j. This transition ensures that if the input context changes also the output context is re-calculated.
* A transition from $S_{enabled}\_u_k$ to $S_{disabled}\_u_k$ is created with:
E[C]/A=output_context_u_j_available[output_context_u_j=NULL]/ϵ. This transition states that the unit is disabled if due to some event (e.g., a user click on a preceding index in the same page) and to the context propagation rules the input context of the depending unit becomes null.

Note that access and standalone units have no transition from the enabled to the disabled state, because for such units it is not possible to change their content once they have been calculated. For example, in the page containing the index of all the books' authors, such index is immediately shown at page entry and no event can change its content.

Example IV: Consider the authors' index page of Figure 4: Figure 6 expresses the hypertext of its first two units, i.e. AllAuthorIndex and AuthorData. For the other units the mapping is applied in an analogous way. Here we suppose that the link between the units be automatic.

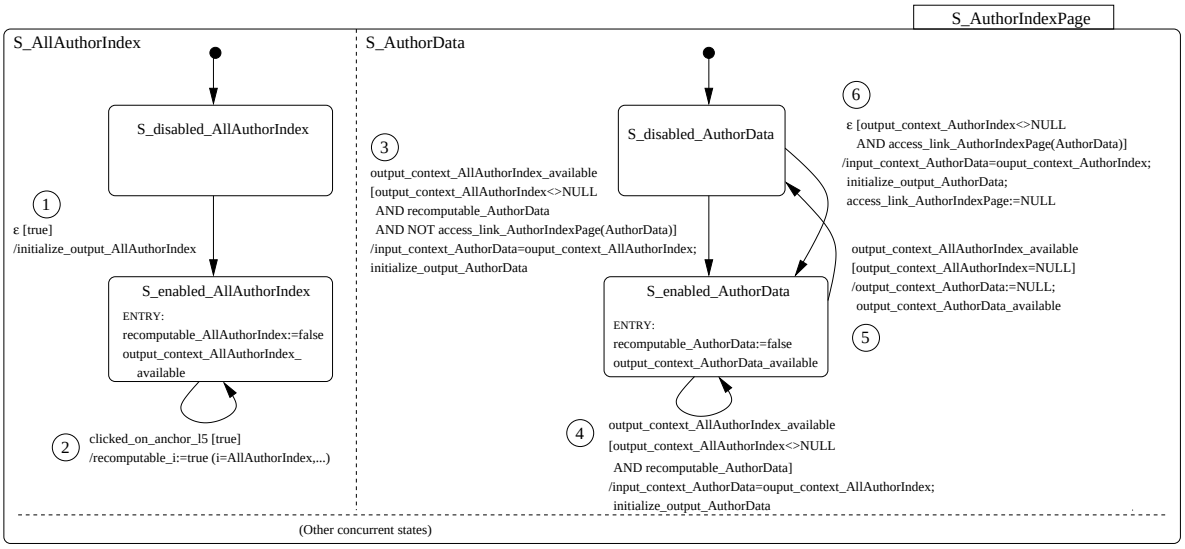


Figure 6: WebML unit configuration statechart

For each unit a concurrent state is created, having two internal states (disabled and enabled).

The first unit (AllAuthorIndex) is a standalone unit and therefore it is automatically enabled (see transition 1). Since its outgoing link is automatic, it also initializes its output with default values: in our system it automatically selects as output the identifier of the first author listed in the index. When the enabled state is entered the unit is set as non-recomputable, i.e.

its content cannot be automatically recomputed[5], and its output context is rendered available to the depending units. The ring transition on the enabled state (transition 2) is triggered when the user selects a new item from the index unit: all the units can then be recomputed according to the new choice (for the index unit we may for example highlight the current choice, for the data unit connected to the index we must show the new selected author, and so on).

The second unit (AuthorData) is both a depending unit, since it depends from the AllAuthorIndex unit, and an access unit, when the page is accessed from the book page. Due to the two access methods, its corresponding state embodies two different transitions (transitions 3 and 6) for passing from the disabled to the enabled state.

As a depending unit the unit is enabled when the output context of the AllAuthorIndex unit is available (transition 3), i.e., every time a new selection in the author index is made by the user or possibly by the system itself. It is fired only if the output context is valid, if the unit has not been already computed, and if there are not any active access links which must be calculated first. Then, it sets its input context to the value of the output context of the unit from which it depends and initializes its output context. Once in the enabled state, it shows the data of the current author and, as in the previous case, it is set as non-recomputable and renders its output available to the following depending unit. From the enabled state two transitions exit: if the user selects a new author from the index a new output context is available: if the context is valid the current state is re-entered and recomputed for the new context (transition 4), i.e., the new current author is shown, otherwise the unit must not be displayed and the unit returns to the disabled state (transition 5). In this latter case the output context of the unit is set to NULL and becomes available to its depending units which cannot be displayed.

As a depending unit the AuthorData unit is enabled when the page variable *access_link* is active for its incoming link (transition 6). Notice that if the page is entered from the book page such variable has been set as active by its incoming link (see transition 4 in Figure 5). If the context is valid the output context is properly initialized and variable *access_link* is unset. Notice that when the page is accessed from the book page only transition 6 is enabled, while transition 3 cannot be triggered. Now we are able to answer the question we issued in Section 3 about the behavior of the page when the outgoing link of the AllAuthorIndex unit is automatic: when the page is accessed from the book page, according to our semantics, the system displays the data of the author selected in the book page and not the first author of the AllAuthorIndex unit.

3.4 Checking the Consistency of WebML Specifications

The specification of WebML semantics through Statecharts allows the designer to better grasp the application behavior and to predict potential critical cases, e.g., non-deterministic and deterministic conflicting transitions, racing conditions, deadlocks and so on. Here, we only show a simple case by means of an example.

Example V: Consider the WebML page of Figure 7: two indexes allow the reader to display information on a certain book. The user may select either a bestselling book from the first index or a recent book using the second one, and the data about the selected book are shown in the data unit. If the two links between the index units and the data unit are defined as

[5]The problem of recomputing the content of a unit becomes relevant in case of cycles among units where all the links are defined as automatic.

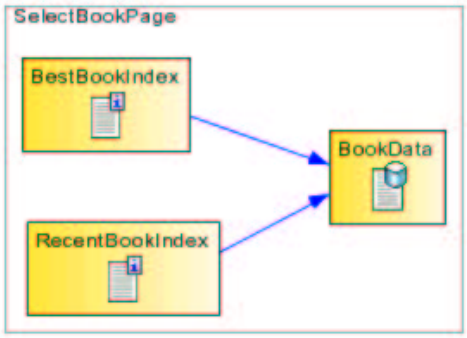


Figure 7: Example of WebML page with racing condition

automatic, the selected item for both indexes is initialized, which results in an unpredictable navigation context to be passed to the data unit (see statechart in Figure 8).

In a typical implementation, propagation of context along links takes places according to some implementation-dependent order, and thus, since the result of applying the two transitions depends on their execution order, a racing condition arises [10].

Notice that, although automatic links are very useful for the specification of automatic behaviors, their use must be carefully controlled to avoid unpredictable behaviors as shown in this example.

The Statechart semantic model has been applied to other, more complex, WebML primitives, including AND-OR nested pages, data entry forms, and update operations. Many subtle behavioral issues have been clarified before implementation with the aid of the illustrated approach.

4 Implementation

The proposed semantic model has set the basis for the implementation of the WebML tool suite.

Figure 9 represents the architecture of WebML, which can be divided into three layers:

1. The *Design Layer*: it includes WebML Control Center, which is the core software element of the WebML architecture, supporting the visual specification of Web sites. Designers use Control Center to input the data structure, the hypertexts diagrams, and presentation directives[6]. WebML specifications are stored as XML documents, which feed the WebML code generator. The output of the code generator is a set of page templates and unit descriptors, which enable the execution of the application in the runtime layer. A page template is a template file (e.g., a JSP file), which expresses the content and mark-up of a page in the mark-up language of choice (e.g. in HTML, WML, etc.). A unit descriptor is an XML file, which expresses the dependencies of a WebML unit from the data layer (e.g., the name of the database and the code of the SQL query from which the population of an index must be computed). Both the templates and the unit descriptors are produced

[6]Presentation directives are expressed as XSL style sheets, which apply to XML documents conforming to the WebML DTD.

by a set of translators coded in XSL and executed by a standard XSL processor. WebML Control Center provides also an interface to the Data Layer to assist the designer in mapping an abstract WebML structure schema to an existing data repository (e.g. a relational database).

2. The *Runtime Layer*: it includes a stack of software components, which produce the actual pages of the application from page templates. Presently, WebML runs on top of any existing JSP 1.1 execution engine, enriched with a thin layer of Java classes decoupling the processing of WebML units from the access API of the data layer. This layer is responsible of extracting the data from the data repository (WebML RunTime) and of formatting it to compose the actual page (WebML TagLib).

3. The *Data Layer*: it includes the repository of data necessary to instantiate the page templates. The inputs to the Data Layer are requests from the runtime layer for data access. The output is the requested content. Presently, WebML can access data stored in any JDBC-compliant relational database and in XML documents.

Three components of the architecture illustrated above have been influenced most by the work on WebML semantics:

- WebML Control Center: the WebML design tool has been extended with a module responsible of checking the consistency of the WebML specifications and of producing warnings and error reports. Checking rules are coded in XSL and enforced by a standard XSL processor. They embody several conservative correctness checks for alerting the designer of potentially dangerous hypertext configurations, e.g., unit and link mismatches, racing conditions and deadlocks.

- The WebML template generator: it embodies the page, unit, and link behavior specified in the Statecharts semantic model. E.g., the sequence of operations needed to correctly generate the passage of context among units and the navigational logics (automatic and clickable links) are encoded at this level.

- The WebML runtime, which insulates the WebML templates from the data source. It has been revised in several aspects to adhere to the described formal semantics. Indeed, this module actually executes the operations specified in the page templates and unit descriptors, by querying the needed data and checking the actual presence of the data in the data source.

The presence of the Statecharts formal semantics has permitted WebML developers to examine on the paper alternative execution options for WebML constructs, and to compare the behavior of the implementation with the expected hypertext execution semantics.

5 Related Work

WebML [3] is one of a family of proposals for the model-driven development of Web sites, which includes also other approaches, e.g., Araneus [11], and Strudel [5]. Like Araneus and Strudel, WebML allows to define the site's structure and content: in the former, the Entity-Relationship model is used to describe the data structure and a conceptual model is used to define the site's hypertext; the latter relies on a data model for semi-structured information and

sites are specified through queries expressed in the StruQL language over the semi-structured data model.

WebML shares several features also with the languages for hypermedia applications, such as HDM - Hypermedia Design Model [9], OOHDM - Object Oriented HDM [12] and RMM - Relationship Management Methodologies [9], from which its basic notations and concepts derive. However, w.r.t. such models WebML has been simplified in order to be effectively supported by CASE tools, and new features specific to data-intensive Web applications have been integrated.

W.r.t. all such approaches (Araneus, Strudel and the hypermedia models), WebML pages and units may be structured in complex ways by means of linking and nesting and exhibit a more sophisticated navigation context semantics, which permits one to define a wide spectrum of page configurations and interactive page-fill behaviors. Indeed, by linking the different kinds of units it is possible to obtain a variety of navigation modes and by defining links as automatic rather than clickable also the content filling of the pages at runtime can be designed. Therefore also our semantics, focusing on the description of such features is quite sophisticated and is not associated only to the simple navigation among pages.

In literature navigation semantics has already been described by means of formal methods: in [13] Petri Nets are used to describe hypertext systems; in [15] and [6, 14] Statecharts are employed to describe navigation browsing. Statecharts are a more powerful mean for the description of reactive systems, since they allow the specification of hierarchical structure; therefore they seem to be suitable for the specification of hypermedia applications requiring synchronization control across different levels of the hierarchical structure.

In [15] Statecharts are used to describe the behavior of hypertext networks: however, the focus is on the specification of system interface behaviors, e.g., related to buttons, frames and so on. Instead, in [6, 14] a model based on Statecharts, called HMSB - Hypermedia Model Based on Statecharts, is used to specify both the structural organization and the browsing semantics of hypermedia applications. Here the focus is on synchronization of multimedia data (i.e. text, audio, animations, images and so on). An environment, called HySChart, supporting the authoring of structured hyperdocuments based on the HMSB model has been proposed, which can be used also as a front-end for Web applications. Compared to the HMSB model, the WebML semantic model addresses structured, data-intensive hypertexts, which do not consist of page instances connected by links, but of page templates, composed by content units which retrieve data from a data layer (e.g. a relational database). For this reason the navigation semantics of WebML results in a more complex specification. Moreover, w.r.t. [6, 14], Statecharts in WebML are not used as a notation for specifying the Web application, but they merely represent the method to formally describe its semantics: in fact, the WebML tools provide a more intuitive graphical notation for the specification of an hypertext, and rely on the formal semantics to provide an efficient specification checker for such hypertext model.

Finally, a different approach based on a generalization of Statecharts is used in OOHDM, which employs ADVcharts [2]. ADVcharts use notations from Petri Nets and statecharts and are used to provide a formal semantics of Abstract Data Views, a concept for designing interactive user interfaces. Diversely from our approach, ADVcharts address the formal specification of dynamic aspects of user interfaces, which are seen as composition of simple visual objects.

6 Conclusions

In this paper we have presented a semantic model for the WebML site design language. The model relies on the mapping of WebML constructs (pages, units, and links) into a Statechart. This mapping caters for all the design primitives of WebML, which are able to formally express the clicking (and automatic) behavior of complex, real-life data-intensive Web applications. The proposed semantics has been extensively used as a reference in the implementation of the WebML design and runtime tools. In the future, further aspects of the WebML language, designed to cope with sophisticated application requirements (e.g., nested pages, update operations, data entry units) will also be given a formal semantics using Statecharts, to formally investigate their properties and runtime behavior and direct their integration in the WebML design and execution environment.

References

[1] A. Bongio, S. Ceri, P. Fraternali, A. Maurino: Modeling Data Entry and Operations in WebML. WebDB (Informal Proceedings) 2000: 87-92

[2] L.M.F. Carneiro, D.D. Cowan, C.J.P. Lucena. ”ADVcharts: a Graphical Specification for Abstract Data Views”. CASCON’93, Toronto, Canada, pp. 84-96 October, 1993.

[3] S. Ceri, P. Fraternali, A. Bongio. ”Web Modeling Language (WebML): a Modeling Language for Designing Web Sites”. Computer Networks, 33, pp. 137-157 (2000).

[4] S. Ceri, P. Fraternali, A. Maurino, S. Paraboschi: One-to-One Personalization of Data-Intensive Web Sites. WebDB (Informal Proceedings) 1999: 1-6

[5] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, D. Suciu. ”Overview of Strudel - A Web-Site Management System”. Networking and Information Systems 1(1): 115-140 (1998).

[6] M.C. Ferreira De Oliveira, M.A.S. Turine, P.C. Masiero. ”A Statechart-based Model for Modeling Hypermedia Applications”. ACM TOIS, April, 2001.

[7] P. Fraternali. ”Tools and Approaches for Developing Data-Intensive Web Applications: A Survey”. ACM Computing Surveys 31(3): 227-263 (1999)

[8] F. Garzotto, P. Paolini, D. Schwabe. ”HDM - a Model-based Approach to Hypertext Application Design”. ACM Transaction on Information Systems 11(1), January, 1-26, 1993.

[9] T. Isakowitz, W. Sthor, P. Balasubramanian. ”RMM: a Methodology for Structured Hypermedia Design”. CACM, 38(8), pp. 34-44 (1995).

[10] D. Harel, A. Naamad. ”The STATEMATE Semantics of Statecharts”. TOSEM 5(4): 293-333 (1996).

[11] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, G. Sindoni. ”The Araneus Web-Base Management System”. SIGMOD Conference 1998: 544-546

[12] D. Schwabe, G. Rossi. ”The Object-Oriented Hypermedia Design Model”. Communications of the ACM 38, 8, 45-46, 1995.

[13] P. Stotts, R. Furuta. ”Petri-Net-Based Hypertext: Document Structure with Browsing Semantics”. TOIS 7(1): 3-29 (1989)

[14] M. A. S. Turine, M. C. Ferreira de Oliveira, P. C. Masiero. ”HySCharts: A Statechart-Based Environment for Hyperdocument Authoring and Browsing”. Multimedia Tools and Applications 8(3): 309-324 (1999).

[15] Y. Zheng, M. Pong. ”Using Statecharts to Model Hypertext”. ECHT 1992: 242-250, 1992.

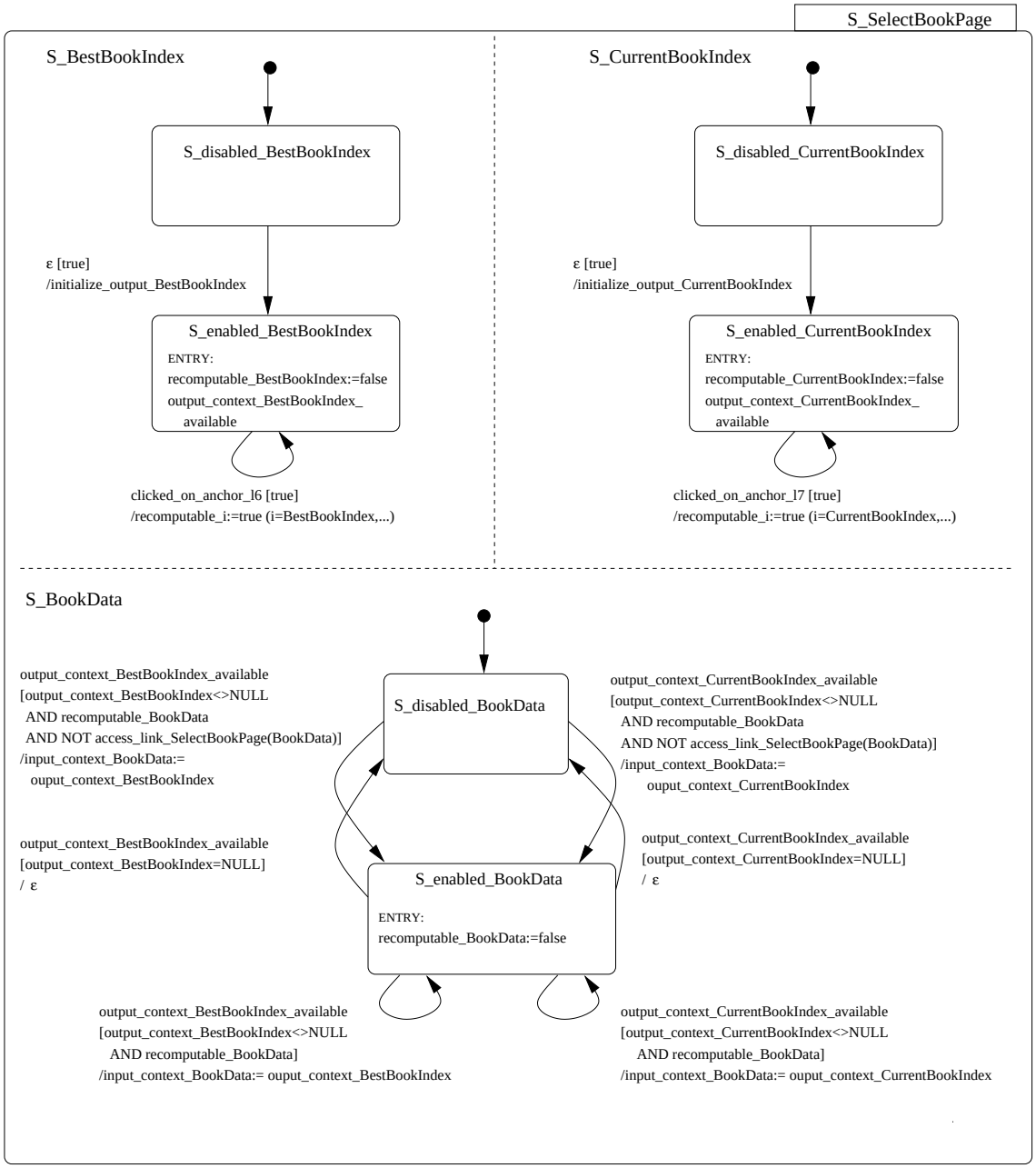


Figure 8: Statechart of a page with racing condition

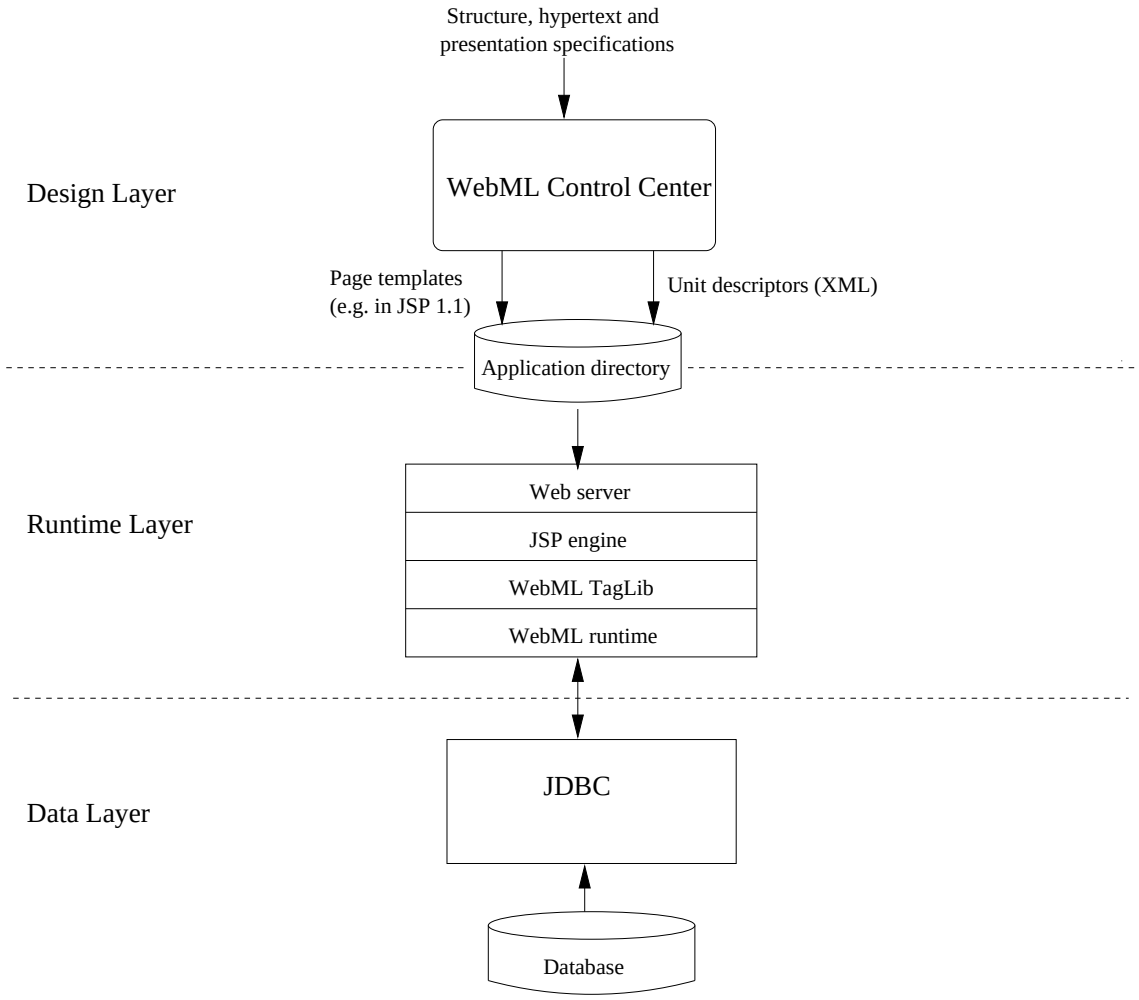


Figure 9: The WebML architecture

Development of a Simple Ontology Definition Language (SOntoDL) and Its Application to a Medical Information Service on the World Wide Web

Rolf GRÜTTER and Claus EIKEMEIER
Institute for Media and Communications Management, University of St. Gallen, Blumenbergplatz 9, CH-9000 St. Gallen, Switzerland

Abstract. It is the vision of the protagonists of the Semantic Web to achieve a set of connected applications for data on the World Wide Web (WWW) in such a way as to form a consistent logical web of data. Therefore, the Semantic Web approach develops languages for expressing information in a machine-processable form. Particularly, the Resource Description Framework (RDF) and RDF Schema (RDFS) are considered as the logical foundations for the implementation of the Semantic Web. This paper documents the development of a Simple Ontology Definition Language (SOntoDL). The development is part of a project aimed at the implementation of an ontology-based semantic navigation through a glossary of an evidence-based medical information service on the WWW. The latest version of SOntoDL is integrated with the RDF/RDFS framework thereby providing for the foundation of a Semantic Web of evidence-based medical information.

1. Introduction

One of the hot topics in medical informatics is the handling of medical terminologies. Thereby, the challenge is twofold. Firstly, the conflicting targets of a concept representation that is close to the real-world and, at the same time, easy to handle by healthcare professionals, e.g., for the coding of diagnoses or indexing of medical subjects, must be solved [17]. Secondly, as existing terminologies like MeSH (Medical Subject Headings), SNOMED (Standardized NOmenclature of MEDicine [19]), and UMLS (Unified Medical Language System [18]) show, it is not evident that medical terminologies provide enough conceptual expressiveness to allow for an easy handling from a formal-logical point of view. Particularly, the mentioned terminologies do not distinguish between generic (IS-A) and partitive (PART-OF) relations [15]. However, a formal-logical foundation is pivotal if terminologies should be processable by computers.

Whereas in closed healthcare settings knowledge engineering, i.e., the discipline that deals with the formal representation of terminologies, has quite a tradition, the advent of the World Wide Web (WWW) brought about an increase in its scale and scope. The increased scale refers to the spread of the WWW which is literally world-wide. The increase in scope refers to the

extension of knowledge engineering methods to non-medical domains, such as corporations. This trend is reflected by the recent Semantic Web initiative [1].

It is the vision of the protagonists of the Semantic Web to achieve „a set of connected applications for data on the WWW in such a way as to form a consistent logical web of data“ ([1], p. 1). Therefore, the Semantic Web approach develops languages for expressing information in a machine-processable form. Particularly, the Resource Description Framework, RDF [14] and RDF Schema, RDFS [5] are considered as the logical foundations for the implementation of the Semantic Web.

This paper documents the development of a Simple Ontology Definition Language (SOntoDL). The development is part of a project aimed at the implementation of an ontology-based semantic navigation through a glossary of an evidence-based medical information service on the WWW. The terminology on which the ontology is based refers to the field of clinical epidemiology. The latest version of SOntoDL is integrated with the RDF/RDFS framework thereby providing for the foundation of a Semantic Web of evidence-based medical information.

2. Application Domain: The Evimed Project

The Evimed project (www.evimed.ch) was initiated in April 1998. It aims at providing general practitioners in the German speaking countries with relevant and reliable information for daily practice, thereby supporting the practitioners in making appropriate medical decisions. To achieve this, a group of physicians who are trained in evidence-based medicine systematically reviews published studies with respect to practical relevance and trustworthiness. These reviews are published together with the links to the original articles in the Journal Club of Evimed (Figure 1).

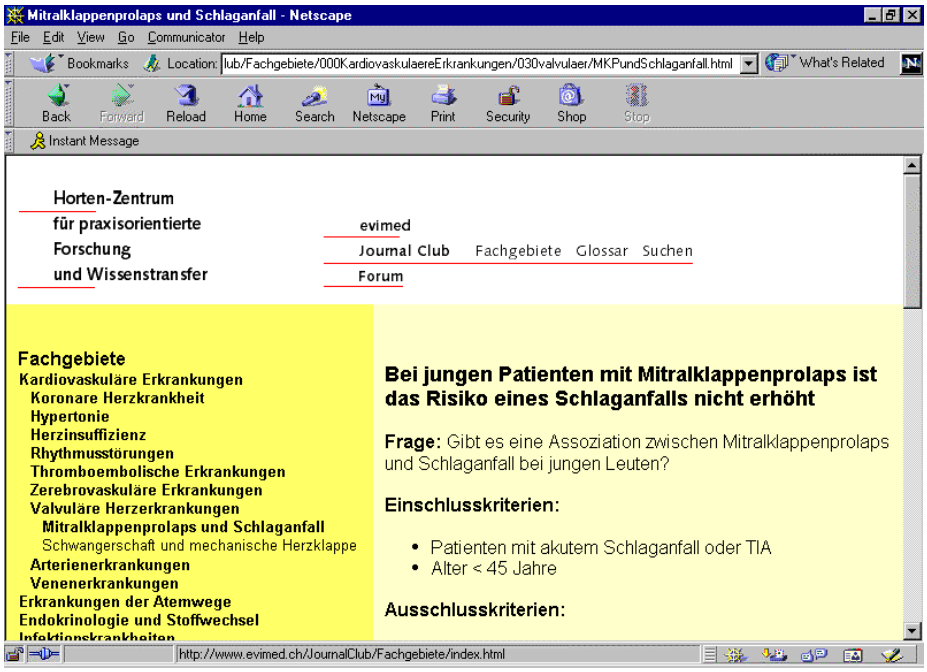


Figure 1. Journal club of the Evimed website

The Journal Club is the core of the Evimed website. Currently (i.e., in July 2001) it stores 400 reviews of selected articles published in various biomedical journals. The reviews are categorized according to 22 specialties and can either be browsed or be accessed via a (syntactic) search engine. The website includes a separate section with articles on the subject of Evidence-Based Medicine (EBM). Evimed also includes a glossary with currently 25 definitions of EBM-specific terms from the field of clinical epidemiology. Further services include free access to Medline, links to literature and other EBM-related sources (e.g., a calendar of events related to further education in EBM), and a guest book offering the possibility to post comments and to subscribe for a free newsletter. Taking into account the objection of practitioners who do not feel comfortable with translating foreign languages, all text of the website is in German.

As mentioned in the Introduction, the development of SOntoDL is part of a project aimed at the implementation of an ontology-based semantic navigation through the glossary of Evimed. The ultimate vision of the project is to effectively meet the information needs of the practitioners who are often not familiar with terms from the domain of clinical epidemiology.

3. Methods: The Concept of the Ontology

The applied conceptual framework refers to the concept of the ontology as defined by Gruber [8]. According to Gruber, an ontology is a specification of a conceptualization, i.e., a formal description of the concepts and their relations for a „universe of discourse“. The universe of discourse refers to the set of objects which can be represented in order to represent the (propositional) knowledge of a domain. This set of objects and the describable relations among them are reflected in a representational vocabulary. In an ontology, definitions associate the names of objects in the universe of discourse with human-readable text, describing what the names mean, and formal axioms constrain the interpretation and well-formed use of the ontology. In short, an ontology consists of the triple (vocabulary, definitions, axioms). Formally, an ontology is the statement of a logical theory.

This conceptual framework has been applied in order to define a simple ontology definition language. The three major steps of the development process are described in the following section.

4. Results: Towards A Simple Ontology Definition Language

The development process of SOntoDL was guided by the following requirements:

(1) The language format must allow for an easy integration with the WWW.
(2) Since the knowledge base of the given domain evolves with time, the language must allow for an easy extension of the ontology without requiring the modification of the program that processes the ontology (i.e., the inference engine).
(3) The language should be applicable not only to the given application domain but also to additional domains.
(4) The language should support the physically dissociated (i.e., distributed) maintenance of representational vocabulary and human-readable definitions, as the two do not require the same frequency of updates and the respective editors may not be the same.

(5) The language should allow for the representation of complex, non-hierarchical knowledge structures.
(6) The language should allow to distinguish between generic (IS-A) and partitive (PART-OF) relations thereby providing enough conceptual expressiveness to support an easy handling of the representational vocabulary from a formal-logical point of view.
(7) The language should be integrated in a common logical framework for connected applications on the WWW thereby taking advantage of a range of tools (hopefully) being developed.

Not all of these requirements were specified prior to the development of SOntoDL. Instead, the list was completed during the development process. The latter can be structured according to the three basic approaches, i.e., intuitive approach, generic approach, and integration into the RDF/RDFS framework.

4.1. Intuitive Approach

The intuitive approach builds on a simple mapping of the paper-based concept hierarchy as provided by the healthcare professionals of Evimed onto an Extensible Markup Language (XML) document tree, whereby the names of the concepts are represented as tag names (Figure 2). By using the emerging WWW standard XML as the core technology for SOntoDL the requirement (1) is met.

```
<EBM_Ontologie>
      ...
      <Stat_Kennzahlen>
            <Kennzahlen_Therapie>
                  <Ereignis>
                        <Ereignisrate>
                              <EER>
                                    <CER>
                                          <ARR/>
                                          <RRR>
                                                <NNT/>
                                          </RRR>
                                    </CER>
                              </EER>
                        </Ereignisrate>
                  </Ereignis>
            </Kennzahlen_Therapie>
            ...
      </Stat_Kennzahlen>
</EBM_Ontologie>
```

Figure 2. Intuitive approach to a simple ontology definition language. Note that the XML representation is incomplete and not ready for a processing by a software program. For complete XML documents cf. Figures 4 and 6.

The intuitive approach yields the advantage of comprehensiveness and easy readability by men. As a main disadvantage the representational vocabulary is fixed and cannot be extended without modifying the Document Type Definition (DTD) (not shown). In other words, it makes

no sense to define a DTD at all (the definition of a DTD for a class of XML documents is optional). The intuitive approach defines a particular ontology language, namely for the domain of evidence-based medicine, rather than an ontology *definition* language and cannot be applied to other domains. Due to its obvious weakness, the intuitive approach – while initially intended – has not been implemented.

4.2. Generic Approach

The generic approach defines an ontology definition language by an XML DTD (Figure 3). Different from the intuitive approach, most of the element types denote generic concepts such as `item`, `identifier`, and `description`. „Generic“ means in this context that the concepts do not refer to a particular application domain (e.g., EBM). In addition, since the `ontology` includes zero, one or more `item` (denoted by the symbol `*`) and each `item`, in turn, includes zero, one or more `item`, the ontology can be arbitrarily extended. This way, the requirements (2) and (3) are met in addition to (1).

```
<!ELEMENT ontology (item*)>
<!ATTLIST ontology version CDATA  #FIXED "1.1">
<!ELEMENT item (identifier+,description, item*)>
<!ATTLIST item myID NMTOKEN #IMPLIED>
<!ELEMENT identifier (#PCDATA)>
<!ATTLIST identifier language (english | german | french) #REQUIRED>
<!ATTLIST identifier format (short | long) #REQUIRED>
<!ELEMENT description (#PCDATA)>
<!ATTLIST description language (english | german | french) #REQUIRED>
<!ATTLIST description implementation (html|url) #REQUIRED>
```

Figure 3. Generic approach to a simple ontology definition language

The generic version of SOntoDL has been applied in order to implement a prototype of an ontology-based semantic navigation through the glossary of an evidence-based medical information service on the WWW [9]. At this small scale, it proved to be well defined. However, when the language was applied to a different domain, i.e., the NetAcademy (www.netacademy.org), its limitations became clear: Since the ontology is implemented as a single XML file, the document soon gets very large and hard to handle. Worse, all updates, be it of the representational vocabulary (element type `identifier`) or of the human-readable definitions (element type `description`), must be made in this central document. In order to anticipate these disadvantages (and to meet requirement 4), the option to implement the description as an URI reference [2] instead of a CDATA section was added to the language (attribute `implementation`). Even though this extension of SOntoDL is marginal, it has a major impact on its applicability. Particularly, it is now possible to integrate external resources into the domain ontology. In other words, external resources can be annotated with a representational vocabulary by simple URI references.

4.3. Integration into the RDF/RDFS Framework

The generic version of SOntoDL takes advantage of the intrinsic structuring capabilities of XML documents, i.e., the representational vocabulary is implemented as a hierarchy of items together with their identifiers and descriptions, whereby each item corresponds to a node of the document tree. This pragmatic approach has been chosen since the Evimed vocabulary was provided as a hierarchy of concepts. The disadvantage of this approach is its limitation to a (mono-) hierarchical representation and the inability to represent more complex knowledge structures. In order to overcome this limitation, SOntoDL is integrated into the RDF/RDFS framework, that is the potential de facto standard for connected applications on the WWW. Thus, along with this third development step, the requirements (5), (6) (see below), and (7) are met in addition to (1) - (4).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:o="http://ontoserver.aifb.uni-karlsruhe.de/schema/rdf">
  <rdf:Description ID="Item">
    <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf resource="http://www.w3.org/2000/01/rdf-
     schema#Resource"/>
  </rdf:Description>
  <rdf:Description ID="childOf">
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Irreflexive"/>
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Asymmetric"/>
    <rdfs:domain rdf:resource="#Item"/>
    <rdfs:range rdf:resource="#Item"/>
    <o:isInverseRelationOf rdf:resource="#parentOf"/>
  </rdf:Description>
  <rdf:Description ID="siblingOf">
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Irreflexive"/>
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Symmetric"/>
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Transitive"/>
    <rdfs:domain rdf:resource="#Item"/>
    <rdfs:range rdf:resource="#Item"/>
  </rdf:Description>
  <rdf:Description ID="parentOf">
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Irreflexive"/>
    <rdf:type resource="http://ontoserver.aifb.uni-
     karlsruhe.de/schema/rdf#Asymmetric"/>
    <rdfs:domain rdf:resource="#Item"/>
    <rdfs:range rdf:resource="#Item"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 4. Integration of SOntoDL into the RDF/RDFS framework

In view of its conception, the generic approach to SOntoDL is closely related to RDF/RDFS, and a re-definition mainly required the effort to become familiar with RDF/RDFS. The concept corresponding to the XML DTD is the RDF *Schema*. Therefore, SOntoDL is re-defined as an application-specific extension to RDFS (Figure 4). In addition to the generic namespaces `rdf` and `rdfs` (referring to the RDF Schema), the namespace `o` [16] is used. The latter refers to a schema that provides an ontology meta-layer for the representation of axioms. These can be used to type relations (i.e., properties in terms of RDF/RDFS) thereby providing the basis for the implementation of integrity constraints for the ontology. While the re-definition of the so far latest version of SOntoDL has been completed, it has only been partially applied to the EBM domain in order to test its applicability. Figure 5 shows the Directed Labeled Graph (DLG) representation of a sample ontology item defined by SOntoDL.

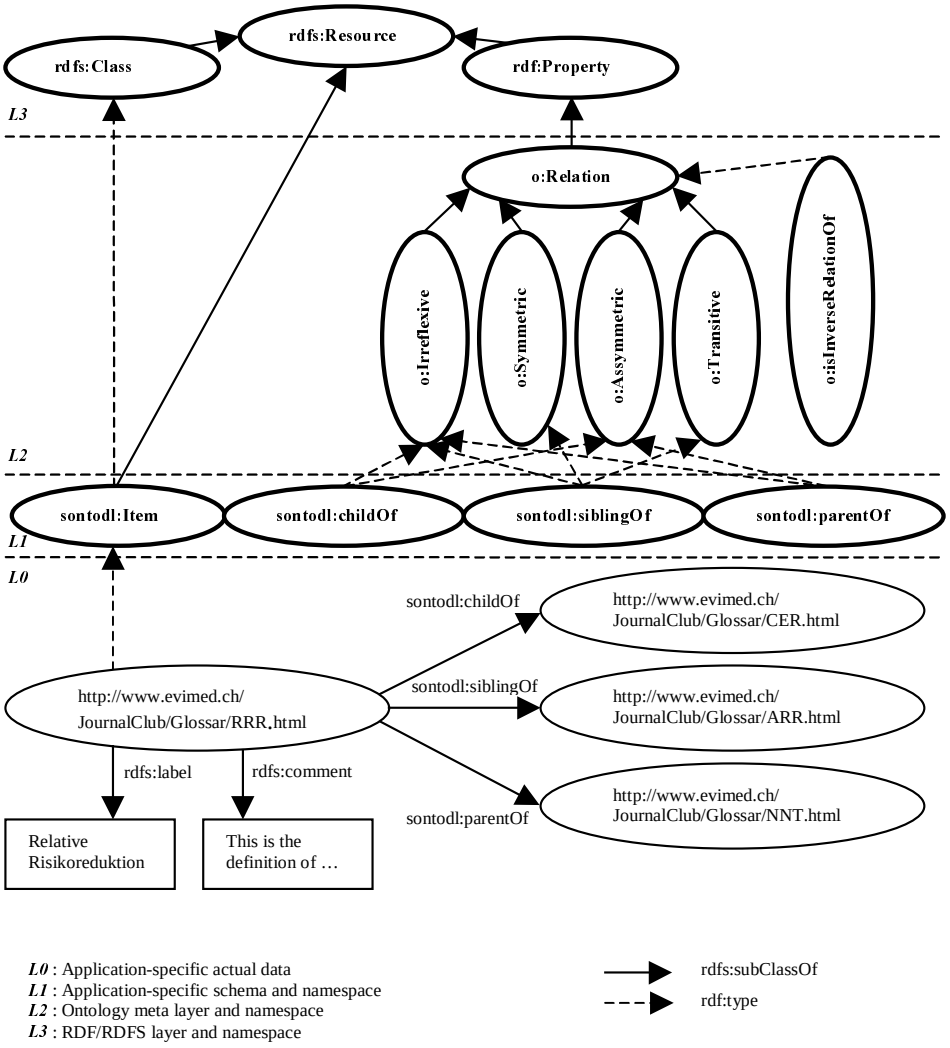


Figure 5. DLG representation of an Evimed ontology item defined by SOntoDL. Note that the `sontodl` namespace refers to SOntoDL which is defined as an application-specific extension to RDFS as shown in Figure 4. The resources on the application-specific actual data layer refer to four glossary items of the Evimed website. The serialization of the DLG representation in RDF syntax is shown in Figure 6.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:dc="http://dublincore.org/documents/1999/07/02/dces/"
 xmlns:odoc="http://ontoserver.aifb.uni-karlsruhe.de/schema/ontodoc"
 xmlns:o="http://ontoserver.aifb.uni-karlsruhe.de/schema/rdf"
 xmlns:sontodl="http://.../schema/sontodl">
  <rdf:Description about="">
    <dc:Title>Evimed Ontology</dc:Title>
    <dc:creator>
      <rdf:Bag>
        <rdf:li>Rolf Gruetter</rdf:li>
        <rdf:li>Claus Eikemeier</rdf:li>
      </rdf:Bag>
    </dc:creator>
    <dc:date>2000-10-25</dc:date>
    <dc:format>text/xml</dc:format>
    <dc:description>An ontology on evidence-based medicine.</dc:description>
    <dc:subject>Ontology, Evidence-based medicine</dc:subject>
    <odoc:url>http://...</odoc:url>
    <odoc:version>2.0</odoc:version>
    <odoc:last_modification>2000-10-25</odoc:last_modification>
  </rdf:Description>
  <rdf:Description about="http://www.evimed.ch/JournalClub/Glossar/RRR.html">
    <rdf:type resource="http://.../schema/sontodl#Item"/>
    <rdfs:label xml:lang="de">Relative Risikoreduktion</rdfs:label>
    <rdfs:comment xml:lang="en">This is the definition of the labeled concept
     by Evimed.</rdfs:comment>
    <sontodl:childOf
     resource="http://www.evimed.ch/JournalClub/Glossar/CER.html"/>
    <sontodl:siblingOf
     resource="http://www.evimed.ch/JournalClub/Glossar/ARR.html"/>
    <sontodl:parentOf
     resource="http://www.evimed.ch/JournalClub/Glossar/NNT.html"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 6. Serialization of the DLG representation of an Evimed ontology item defined by SOntoDL in RDF syntax. Note that additional schemata, referred to by the namespaces dc and odoc, are used in order to represent ontology meta-data.

5. Discussion

There is quite a number of related approaches to ontology definition languages (cf. Section 6). Nevertheless, the presented approach follows its own conception. The reason therefore is that the need for an ontology definition language arose during a particular project in a particular application domain (cf. Section 2). As a consequence, initially a pragmatic approach was chosen which was step by step refined as reported in Section 4. The core of the initial conception was to keep the language as simple as possible and to avoid unnecessary overkill (therefore, an unreflected application of one of the languages mentioned in Section 6 was not considered).

During the project, this conception was partially weakened, primarily in favor of a more general applicability.

As mentioned, the latest version of SOntoDL resulting from the integration into the RDF/RDFS framework unfolds its full potential only if increasingly tools are available that support RDF/RDFS. In case another technology for the implementation of the Semantic Web, such as Topic Maps [3] outsmarts RDF/RDFS, SOntoDL has to be adapted.

6. Related Work

SHOE (Simple HTML Ontology Extensions) provides distributed ontologies consisting of categories and relationship rules [10]. Thereby, the categories provide for the classification of instances. They are organized hierarchically and support multiple inheritance. The relationship rules are implemented as Horn clauses. The instances (i.e., individual constants in terms of Horn-rules) are represented as URLs/URIs. This is similar to the approach as presented in this chapter, where the human-readable definitions, as part of the ontology, are likewise represented by URI references. SHOE was originally specified in SGML (as is HTML) (before the definition of XML) but is meanwhile also specified as an XML DTD.

XOL (XML-based Ontology Exchange Language) is a language for specifying and exchanging ontologies [13]. XOL is specified in an XML-based syntax (kernel DTD). It uses a frame-based semantic model, i.e., OKBC-Lite. An XOL file consists of a module-header definition and one or more class, slot and individual definitions. The module-header definition provides meta-information of the ontology, such as the name and version. The class definitions provide the classes and subclasses of the defined individuals. The slot definitions are strings that encode the official names of the entities. Each slot definition refers to a class name. The individual definitions provide the names, documentations, instance-of information, and slot-values of the defined individuals. As a disadvantage, XOL does not re-use the core semantics of RDF/RDFS. Hence, „pure“ RDF/RDFS applications cannot process even the core object-model definitions.

The Ontobroker application answers queries based on a facts base and an ontology base [7]. The facts base stores instance information („values“ in terms of the query interface) which is extracted from annotated HTML pages, for instance, of a corporate Intranet, which is different from the hereby presented approach, where the HTML pages, i.e., the resources, are externally annotaded by RDF descriptions. The ontology base stores a set of ontologies. Each ontology includes a concept hierarchy („classes“), a set of slot definitions („attributes“), and a set of rules. The rules implement integrity constraints for the ontology. Ontologies are defined as F-logic statements.

An approach to representing ontologies in RDF/RDFS, similar to [16] (and to the latest version of SOntoDL), is pursued with OIL [12, 6]. OIL uses description logics for the definition of concepts and relations and proposes an ontological meta-layer that is combinable with the herein applied axiom categorization proposed by [16].

Closely related is the recent approach as pursued by DAML (DARPA Agent Markup Language) [11]. The goal of the DAML program is to create technologies that enable software agents to dynamically identify and understand information sources, and to provide interoperability between agents in a semantic manner. Particularly, an agent markup language developed as an extension to XML and RDF should allow users to provide machine-readable semantic annotations for specific communities of interest. According to the initiators of the DAML program and

similar to Ontobroker, objects in the Web will be marked to include descriptions of information they encode, of functions they provide, and/or of data they can produce. In addition, DAML should allow for an „ontology calculus“ similar to the relational calculus that makes DataBase Management Systems (DBMS) possible.

References

1. Berners-Lee, T.: Semantic Web Road map. (1998) Retrieved July 6, 2001 from the World Wide Web: http://www.w3.org/DesignIssues/Semantic.html
2. Berners-Lee, T., Fielding, R., Irvine, U.C., Masinter, L.: Uniform Resource Locators (URI): Generic Syntax. (1998) Retrieved July 6, 2001 from the World Wide Web: http://www.ietf.org/rfc/rfc2396.txt
3. Biezunski, M., Bryan, M., Newcomb, S.R.: ISO/IEC FCD 13250:1999 – Topic Maps. Retrieved July 6, 2001 from the World Wide Web: http://www.ornl.gov/sgml/sc34/document/0058.htm
4. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E. (2000). Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. Retrieved July 6, 2001 from the World Wide Web: http://www.w3.org/TR/REC-xml
5. Brickley, D., Guha, R.V.: Resource Description Framework (RDF). Schema Specification 1.0. W3C Candidate Recommendation 27 March 2000. Retrieved July 6, 2001 from the World Wide Web: http://www.w3.org/TR/rdf-schema/
6. Broekstra, J., Klein, M., Decker, S., Fensel, D., van Harmelen, F., Horrocks, I.: Enabling Knowledge Representation on the Web by Extending RDF Schema. The Tenth International World Wide Web Conference (WWW10), May 1-5, 2001, Hong Kong. ACM 1-58113-348-0/01/0005. Retrieved July 9, 2001 from the World Wide Web: http://www10.org/cdrom/papers/291/index.html
7. Decker, S., Erdmann, M., Fensel, D., & Studer, R. (1999). Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al. (Eds.): Semantic Issues in Multimedia Systems. Proceedings of DS-8 (pp. 351-369). Boston: Kluwer Academic Publisher.
8. Gruber, T.:. What is an Ontology? (1997) Retrieved July 6, 2001 from the World Wide Web: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
9. Grütter, R., Eikemeier, C., & Steurer, J. (2001). Up-scaling a Semantic Navigation of an Evidence-based Medical Information Service on the Internet to Data Intensive Extranets. In Proceedings of the 2nd International Workshop on User Interfaces to Data Intensive Systems (UIDIS 2001). Los Alamitos, California, USA: IEEE Computer Society Press.
10. Heflin, J., Hendler, J., & Luke, S. (1999). SHOE: A Knowledge Representation Language for Internet Applications. Technical Report CS-TR-4078, University of Maryland, College Park.
11. Hendler, J.: DAML: The DARPA Agent Markup Language Homepage. Retrieved July 6, 2001 from the World Wide Web: http://www.daml.org
12. Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., Harmelen, F.V., Klein, M., Staab, S., Studer, R.: The Ontology Interchange Language OIL: The Grease Between Ontologies (Tech. Rep., 2000). Dep. of Computer Science, Univ. of Manchester, UK/ Vrije Universiteit Amsterdam, NL/ AIdministrator, Nederland B.V./ AIFB, Univ. of Karlsruhe, DE. (http://www.cs.vu.nl/˜dieter/oil/)
13. Karp, P.D., Chaudhri, V.K., Thomere, J.: XOL: An XML-Based Ontology Exchange Language (Tech. Rep., Version 0.3, 1999)
14. Lassila, O., Swick, R.R.: Resource Description Framework (RDF). Model and Syntax Specification. W3C Recommendation 22 February 1999. Retrieved July 6, 2001 from the World Wide Web: http://www.w3.org/TR/REC-rdf-syntax/
15. Schulz, S., Romacker, M., Hahn, U.: Ein beschreibungslogisches Modell für partitive Hierarchien in medizinischen Wissensbasen. In: Greiser, E.; Wischnewsky, M. (Hrsg.): Methoden der Medizinischen Informatik, Biometrie und Epidemiologie in der modernen Informationsgesellschaft. MMV Medien & Medizin Verlag, München (1998) 40-43
16. Staab, S., Erdmann, M., Maedche, A., Decker, S.: An Extensible Approach for Modeling Ontologies in RDF(S). Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB), Universität Karlsruhe (2000). Retrieved July 6, 2001 from the World Wide Web: http://www.aifb.uni-karlsruhe.de/~sst/Research/Publications/onto-rdfs.pdf

17. Straub, H.: Four Models of Concept Architectures. In: Grütter, R. (ed.): Knowledge Media in Healthcare: Opportunities and Challenges. Idea Group Publishing, Hershey/London (2002) (to appear).
18. United States National Library of Medicine. Unified Medical Language System (UMLS). Retrieved July 6, 2001 from the World Wide Web: http://www.nlm.nih.gov/research/umls/umlsmain.html
19. Wingert, F.: SNOMED Manual. Springer-Verlag, Berlin (1984)