Bulletin of the Technical Committee on

Data Engineering

December 2003 Vol. 26 No. 4 **IEEE Computer Society**

Letters

Special Issue on Making the Semantic Web Real

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Distribution

IEEE Computer Society
1730 Massachusetts Avenue
Washington, D.C. 20036-1992
(202) 371-1013
jw.daniel@computer.org

Letter from the Editor-in-Chief

The Current Issue

My (perhaps naive) view of how we progress from the world of human understanding (involving semantics) to the world of automatic data processing of one form or another (involving syntax) is to capture more of the human world of semantics as syntax. This is a complex activity, and requires that people agree on a number of important issues, such as (i) how to express the syntax, (ii) how the words (character strings) relate to one another and to various computer operations. Historically, the database community has been a bit spoiled. The syntax was given to us very early by Ted Codd. The ”words” were defined only locally, and were provided from well defined business data processing that in many cases was already being done.

Now we have the world wide web to deal with. One need not buy into some grandiose vision (a point made by Manola in this issue) to see that as we incrementally provide more ”syntax”, and fill in information, e.g. via ontologies or via data modelling, we will provide an ever more useful ”semantic web”. The test of success, in its most basic economic sense, will be if and when companies can make money providing web services that win them financial success. I am convinced that this will happen. Much of the activity and research described in this issue of the Bulletin provides strong evidence that progress is being made. A final caution- the semantic web is a huge challenge, so be patient. Looked at another way, it is a major research opportunity!

I’d like to thank Umesh Dayal, Harumi Kuno, and Kevin Wilkinson for undertaking this very ambitious issue on the semantic web. I think Bulletin readers will find there is much to learn from this newly emerging and increasingly important part of the database field.

David Lomet
Microsoft Corporation

TC on Data Engineering: Election of Chair for 2004-2005

TC on Data Engineering: Election of Chair for 2004-2005

The Chair of the IEEE Computer Society Technical Committee on Data Engineering (TCDE) is elected for a two-year period. The mandate of the current Chair, Erich Neuhold, terminates at the end of 2003. Hence is time to elect a Chair for the period January 2004 to December 2005. Please vote before January 25, 2004 using the ballot on the next page.

The Nominating Committee, consisting of Masaru Kitsuregawa, Paul Larson, and Betty Salzberg, is nominating Erich Neuhold for a second term as Chair of TCDE. Erich’s position statement and a short biography are included below. The Committee invited nominations from members of the TCDE but received no other nominations.

Paul Larson
Nominating Committee Chair

Position Statement and Biography

ERICH NEUHOLD
Professor, Department of Computer Science
Darmstadt University of Technology
Wilhelminenstrasse 7
D-64283 Darmstadt, Germany
neuhold@ipsi.fhg.de

Biography

Erich Neuhold is Professor of Computer Science at Darmstadt University of Technology and Director of the Fraunhofer Institute for Integrated Publication and Information Systems (IPSI) also in Darmstadt, Germany. He has been Professor at the University of Stuttgart and the Technical University of Vienna and he has also worked in research and management positions for IBM and Hewlett Packard both in Europe and the USA. His areas of expertise in databases include distributed databases, object-oriented databases, databases for the Internet (e.g. semi-structured data and XML), information retrieval, information visualization and their applications in digital libraries, cultural heritage and e-commerce.

He has published four books and about 190 papers. His work has appeared, among others, in the VLDB Journal, Information Systems, Acta Informatica and in many conferences as, for example, VLDB, ICDE, MMDB, ADL and DL. He has served in all capacities on many conference committees and was PC Chair and General Chair of ICDE. He currently holds the chairs of the IEEE-CS Technical Committee on Data Engineering and the Technical Committee on Digital Libraries. He is also the Chair of the ICDE Steering Committee and also member of the JCDL at ECDL Steering Committees. He is a Senior Member of IEEE.

Position Statement

If reelected I will continue my work to increase the visibility and attractiveness of the TCDE and ICDE for researchers and, most important, for industry members. Database research had a tremendous influence on DB technology and has gained continuously by feedback from the practitioners. Such cooperation has to be strengthened wherever possible. One of the possibilities here is to explore ways to broaden ICDE (International Conference on Data Engineering) by either industry oriented tracks or associated workshops that gear to the practitioners and users and feed back their problems and experiences to the research community. I will continue to achieve this goal by working closely together with the I CDE Steering Committee. Another goal of my chairmanship is to establish more joint events with data base societies and action groups in other countries. Beside this the Technical Committee on Data Engineering continue to regularly publish its newsletter, the Data Engineering Bulletin, and to sponsor the ICDE (International Conference on Data Engineering) and the associated workshop RIDE (Research Issues in Databases). In this context I will also continue to work with SIGMOD to make the ICDE proceedings and other relevant publications available on CD-ROM and the SIGMOD archive.

Erich Neuhold
Darmstadt University of Technology

ELECTION BALLOT

ELECTION BALLOT

TECHNICAL COMMITTEE ON
DATA ENGINEERING

The Technical Committee on Data Engineering (TCDE) is holding an election for Chair. The term of the current chair, Erich Neuhold, expires at the end of 2003. Please email, mail or fax in your vote.

BALLOT FOR ELECTION OF CHAIR
Term: (January, 2004 - December, 2005)

Please vote for one candidate.

☐ Erich Neuhold

☐ _____
(write in)

Your Signature:_____

Your Name:_____

IEEE CS Membership No.:_____
(Note: You must provide your member number. Only TCDE members who are Computer Society members are eligible to vote.)

Please email, mail or fax the ballot to arrive by January 25, 2004 to:

s.wagner@computer.org
Fax: +1-202-728-0884

IEEE Computer Society
Attn: Stacy Wagner
1730 Massachusetts Avenue, NW
Washington, DC 20036-1992

RETURN BY January 25, 2004

TC ON DATA ENGINEERING
IEEE Computer Society

Letter from the Special Issue Editor

The Semantic Web is a collaborative effort by the W3C to standardize the representation of Web content and to enable its automated processing. After years of research and standards activity, there is growing interest in finding real-world applications for Semantic Web technologies, e.g., RDF, RDF Schema, OWL. Tim-Berners Lee emphasized this in his keynote address at the 2003 WWW conference. Products involving Semantic Web technologies are emerging from start-ups such as Semagix and Network Inference. But Semantic Web applications have not yet made a large impact on the mainstream.

Real-world applications are subject to real-world requirements. The opportunity is at hand for the database community to address challenges faced by Semantic Web applications. Indeed, the database community is becoming more visible in the Semantic Web world. The 2002 Amicalola Report invited senior researchers from within and outside of database and information systems community to discuss DB-IS Research for Semantic Web and Enterprises. VLDB 2003 saw the first workshop on the Semantic Web and Databases and the International Semantic Web Conference had the first workshop on Practical and Scalable Semantic Systems.

The theme of the March 2002 Data Engineering Bulletin was "Organizing and Discovering the Semantic Web." It focused primarily on search technologies, e.g., computing Web page importance, classification, etc. This issue has a follow-up theme – "Making the Semantic Web Real." We begin with a commentary by Frank Manola who provides his perspective on the Semantic Web, comparing it to a heterogeneous distributed database system. He then makes some acute observations about the practical viability of the Semantic Web and discusses relevant database research challenges.

The remaining articles are organized into two sections, the first addressing core technology challenges in realizing the Semantic Web and the second describing Semantic Web applications and application frameworks. The article by Vassilis Christophides et al. presents their Semantic Web data integration middleware that uses Datalog-like rules to map XML and relational data to RDF where it can then be processed with Semantic Web tools and query languages. The article by Laks Lakshmanan and Fereidoon Sadri also proposes mapping data sources to a semantic model. They describe query processing and optimization techniques that use semantics of the data sources such as foreign keys and functional dependencies.

The article by Paulo Pinheiro da Silva et al. addresses information *provenance* and presents an infrastructure for managing provenance at varying levels of detail. Provenance, and the related notion of context, is an important issue for the Semantic Web where information is retrieved and derived from multiple sources. The article by Wilkinson et al. discusses challenges in storing and retrieving large RDF data sets and the implementation of RDF persistence in Jena, a Semantic Web developers' toolkit.

The first application article is by Amit Sheth and Cartic Ramakrishnan. It argues that "a little bit of semantics" can significantly improve applications, that complex reasoning is not always required. It then presents the Semagix Freedom architecture, an application framework that embodies this notion. The article by Nathalie Morena et al. presents another application framework. It has tools for developing storage schema, querying and annotating content, including the ability to annotate content from the *Deep Web*, i.e. generated content.

The article by Joseph Kopena and William Regli describes the use of Semantic Web technologies for complex design repositories. This enables components to be described and reused based on functional characteristics rather than just the physical characteristics captured by current design repositories. The issue concludes with Christoph Bussler's article on the relevance of Semantic Web technology to Enterprise Application Integration and the extension to semantic web services.

We hope you find these articles interesting and that they suggest ideas for future research. And, we want to thank the authors for their excellent contributions.

Umeshwar Dayal, Harumi Kuno, Kevin Wilkinson
Hewlett-Packard Laboratories
Palo Alto, California, USA

A Database Perspective on the Semantic Web: A Brief Commentary

Frank Manola
fmanola@acm.org

1 The Semantic Web as a Database

The Semantic Web [15] is an exciting evolution of the current World Wide Web, and there is much activity surrounding it. Much of what has been published about it has emphasized the potential of what might be accomplished using improved semantic descriptions of Web resources, based on ontologies and logic-based processing. However, the emphasis on logic, agent-based processing, and so on may obscure for the database community the fact that much of the essence of the Semantic Web comes from its similarity to a database. To back this up, consider the following statements by Tim Berners-Lee, the "inventor" of the World Wide Web, about his vision of the 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." [15]

> "The Semantic Web is a web of data, in some ways like a global database." and "Leaving aside the artificial intelligence problem of training machines to behave like people, the Semantic Web approach instead develops languages for expressing information in a machine processable form." [3]

> "The semantic web data model is very directly connected with the model of relational databases ... Indeed, one of the main driving forces for the Semantic web has always been the expression, on the Web, of the vast amount of relational database information in a way that can be processed by machines." [4]

These statements indicate that a significant focus of the Semantic Web involves simply making Web data more machine-processable, *and more like a database*. Of course, if programs are to process this data correctly, they must process it in a manner consistent with its semantics. However, *the Semantic Web vision doesn't really care how the programs acquire this capability.* For example, this capability may exist because the data uses a vocabulary (and associated semantics) that has been agreed on within a user community, and the programs are written to use that vocabulary, just as with most databases now.

This isn't to say that having more semantics explicitly described in metadata (such as ontologies), using richer languages capable of associating more "meaning" with the data, won't be of tremendous importance. Obviously such explicitly-specified semantic information will be increasingly important in the Web environment,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

given its scale, number of data sources, and the autonomy and heterogeneity of these sources. For example, such information will be crucial in such areas as integrating and making sense of information from heterogeneous sources, optimizations of various kinds, advanced types of concurrency control, and so on. However, lots of very effective Semantic Web applications (and certainly lots of near-term ones) will not necessarily involve extensive use of machine-processable semantic information (and only a relatively small amount of the semantics of terms can actually be captured in a formal ontology [10], although even a small amount can be extremely useful in practice). As someone said on the xml-dev email group, "Semantic Web my behind! I just want to order a pizza, not have mozzarella explained to me!" It will be a tremendous job to migrate the current Web to the point where all of the data is machine-processable, and where all of the current tools we have available in database systems are available, even without machine-processable semantics. Hence, it will be important not to neglect technologies in areas other than those directly involving machine-processable semantic information, and to integrate those technologies more thoroughly into the Web environment.

So, at its core, the Semantic Web is about making the Web more like a distributed database system, making it more like a *heterogeneous* distributed database system, giving it a richer conceptual model to better describe that data, and adding additional deductive capabilities on top to make better use of all that data. These are all lines of development that have been, and still are, currently active in the database community.

Of course, moving structured database concepts into the Web environment involves some differences as well as similarities. For example, like the current Web, all Semantic Web data would be considered as part of *one big database*, unlike current database systems, in which related data is created as physically (and logically) separate "databases."

Because all this development happens in *one big database* (the Web), rather than in lots of separate ones, different individuals and user communities will use parts of this big database for different purposes, and hence different parts of this big database will use different (or overlapping terms), and will often contain inconsistent (and sometimes untrustworthy) data (just the way the Web does now, using less-structured data). Of course, the hope is that gradually the Semantic Web will migrate toward increasingly-large portions of that Web database using more consistent vocabularies and data. And technologies pioneered in work on heterogeneous database integration will be used to enable further data integration. But even without that, the Semantic Web is still a database, and people can do useful things with such data, in many cases by applying database-like operations to large collections of it, including data from different sources (and this without a single global ontology, as some commentators claim is necessary.)

In fact, people and organizations are getting real use out of some of the Semantic Web technologies now, as illustrated by some of the applications described in the RDF Primer [9].

Moreover, the structured data making up the Semantic Web will be *added* to the existing Web, not placed in an entirely separate space. This creates all kinds of possibilities for making use of both the easily-human-interpretable data on the current Web, and the easily-machine-processable data added to it on the Semantic Web, in novel combinations. This together with the fact that, since this big database is the Web, it can (security and privacy concerns permitting) be both accessed by and added to by anyone.

To deal with some of the differences involved in using database ideas in the Web environment, and in mixing database-like data with the current Web, the Semantic Web employs variants on certain database ideas. For example, from a data modeling perspective, the Semantic Web is based on the Resource Description Framework (RDF) [14], which is effectively a form of highly-normalized relational model. In this model, URIs (Uniform Resource Identifiers [2]) identify the things people want to talk about (termed "resources"). URIs can be thought of as a superset of the URLs (Uniform Resource Locators) used in web browsers. As with URLs, different persons or organizations can independently create URIs, and use them to identify things. However, unlike URLs, URIs are not limited to identifying things that have network locations, or use other computer access mechanisms. In fact, a URI can be created to refer to anything anyone wants to talk about, including

- network-accessible things, such as an electronic document, an image, a service (e.g., "today's weather

report for Los Angeles"), or a collection of other resources.

- things that are not network-accessible, such as human beings, corporations, and bound books in a library.
- abstract concepts that don't physically exist, like the concept of a "creator" property.

Given URIs to identify resources, people can then write property/value pairs describing those resources in the form of triples of **subject** (resource), **property**, and **object** (resource or literal value). This is the basic idea behind not only RDF, but also languages layered on it such as the ontology languages DAML+OIL [7] and OWL [12].

Because URIs are unique throughout the Web, one person's use of a term can be distinguished from another person's use of the same term, through the use of distinct URIs e.g.,

- *http://mysite.example.com/vocabulary/creator* **vs** *http://yoursite.example.com/vocabulary/creator*.

In addition, one person can add additional descriptive information about a resource described by another person, by using the appropriate URI as its key, and (effectively) joins can be used to aggregate this information.

The basics of this type of model have had a long history in the database community, in the form of binary relations with surrogates (the URIs) as keys, since RDF triples can also be thought of as rows in binary relations having the form **property (subject, object)**. The pros and cons of this type of model have been long-debated in the database community, although not in the Web context (see, e.g., some of the comments on "semantic" models in textbooks such as [8], and the associated references). In particular, RDF resembles Codd's RM/T model [6] in some respects, particularly since in RDF, schema/ontology information is represented in the same way as instance data, and hence mixed data and metadata queries are easily possible. RDF is also very much like the graph models used in recent work on "model management". However, the RDF / Semantic Web variant involves an important extension, in that it is grounded in the Web: the subjects, objects, *and properties* in RDF statements (in effect, the table and column names, as well as key values) are URIs, and hence have global identity. Moreover, in many designs these URIs can be dereferenced (you can use them as URLs and point a browser at them) to retrieve Web data that describes the resources. The result is an interesting mixture of relational and object-oriented ideas (since the URIs are also a form of object identifier), and of data and metadata. Moreover, compared with XML, RDF really does play the role of the "relational model of the Web", since, for example, it requires that relationships not be recorded implicitly in data structures (such as XML nested elements), but instead must be made explicit.

The variations on relational themes represented by RDF help deal with a number of differences between the Web environment and a typical database environment. For example, in RDF each triple (row in a binary table) represents a single fact, unlike a conventional relational table, in which each row represents a set of facts. In a conventional database (and in certain Web applications too), it is expected that a relatively small, fixed set of attributes will be used in describing a particular entity. However, on the Web, even though a particular schema might intend to constrain the properties that apply to a particular resource, any number of properties, from multiple sources, might be unpredictably applied to describe a particular resource (often by different people or organizations), and the set of properties might be extended at any time.

Moreover, RDF schemas are *descriptive*, not *prescriptive*. An application may choose to validate instance data against a schema, but this is not mandatory. For example, instance data may be perfectly valid either without some schema-defined properties, or with additional properties not defined in a given schema [9].

Hence, one of the things RDF caters to is the potential for highly irregular or unpredictable amounts of data about a given resource. Where structural regularity is possible (and this is likely in many applications), the RDF from a given source could be stored in a relational (or XML) database (and relational data can easily be interpreted as RDF).

And, of course, the use of URIs as names enables RDF to deal with the problem of distinguishing terms and other resources that might be referred to by many people and organizations in a single, large database.

2 Research Themes

Given the above description of the Semantic Web as a database, it should be clear that there is considerable room for applying numerous database research themes in this new context.

The papers in this issue illustrate a number of these themes, themes that are also illustrated in numerous other papers on the Semantic Web. These include both the application of Semantic Web languages to classic database research areas such as heterogeneous data integration, consideration of relatively new issues that the Semantic Web context makes particularly important, such as that of data provenance, and the use of variations on "semantic" technology in general.

In fact, considering the Semantic Web as a giant database illustrates that the whole range of database research themes, suitably adapted to the Semantic Web differences, can be usefully explored in this new context. This includes such issues as:

- From a data model perspective, to what extent does the Semantic Web truly justify a new data model (this echoes a corresponding concern within the logic/knowledge representation community at being restricted to RDF's binary relations)? Do earlier arguments about RDF-like data models still apply, and with what force? Are there alternatives to handling global naming and its ramifications, and the need to be able to mix and match terms from multiple vocabularies?
- What is the best way to integrate RDF and relational-style processing with XML? RDF has an XML syntax, but this syntax has received numerous bad reviews. Is there a better syntactic approach?
- The current Web is asymmetric with respect to reading and writing (creating and modifying Web data generally involves different tools than accessing it). What is the best way of reducing this asymmetry? In particular, although some updating protocols exist (like WebDAV), how can we get better transactional behavior for Web data, and what forms of transactions are most suitable?
- Databases have historically emphasized a distinction between logical and physical data structures. In contrast, in the Semantic Web this distinction is currently not as well developed. RDF is typically stored as a graph, rather than using more efficient representations. Can we develop more efficient processing for RDF, and still retain its ability to deal with irregular and varying data structures?

Regarding capturing "semantics" (or conceptual design), people typically have "ontologies" in mind when they design databases. We've been training them to think like this for years in studying database design. However, we haven't had a clean way (or much reason) to capture all that information in the richer models that knowledge representations provide (without using a separate tool often not linked to the database). Now we have better reasons to push this harder. The information systems community needs to adopt ontologies, make them a more explicit part of their data management strategies, and then investigate how to use them in database processing. All the prior work by the database community in richer conceptual languages and deductive databases is relevant here. At the same time, we need to make sure to point to the need to support "industrial scale" management of large amounts of definitions and content that continually change (leveraging, e.g., prior work on versioning and schema modification), and the whole design and implementation process.

Prior work by the database community on integration of heterogeneous data is clearly relevant, and needs to be pushed and extended. A major effort is required to make the process of doing this integration easier. It can't all be automated (e.g., writing "articulation axioms"), or the requirements known ahead of time, so we need to investigate how to make it easier for people to help, both at design time (as before) and *at run time* (we've got instant messaging between people; how about between an agent or query engine and a person, along the lines of "I don't understand exactly what 'sale price' consists of; can you clarify?"). Such an intervention might be quite reasonable in the context of some types of Web applications.

Work on integration also needs to consider how to use the Web (and its technologies) as a potential organizing basis and environment or infrastructure for information sharing (i.e., use of the Web as a *tool* for solving data

management problems, not just as the *object* of data management technologies). A key observation here seems to me that part of solving the data integration problem is providing a data integration environment that enables all the pieces to be put together (and, for example, enables people to have access to and potentially reuse metadata and schema designs). The Web is a widely-available shared environment which can make this happen (the development of shared repositories for XML DTDs and schemas is an example of this idea in action).

Finally, a lot more work needs to be done on the various notions of "contexts" [11, 13, 1]. This involves more than just recording and using (e.g., for mediation) metadata about the definitions and assumptions used by different data collections. It also involves being able to integrate information described using languages of different descriptive capabilities, as well as being able to integrate different "reasoning engines" that operate on that information, on the same Web, and have them interoperate. The Web allows people with different reasoning capabilities to interoperate (up to a point!), and this needs to be a goal for our systems as well. The "tradeoff between expressiveness and computability" is frequently cited in discussing the use of deductive machinery in the Semantic Web. We need to be prepared for the fact that people will make different decisions about this tradeoff (e.g., making use of more powerful deductive machinery on relatively limited *subsets* of the Semantic Web, and making use of more efficient machinery on problems of larger scope), and try to cope with all of them.

3 Some Observations

While there is an awful lot of "hype" about the Semantic Web, there is also an awful lot of what might be termed "negative hype" as well; that is, discussion that is equally "hypish", but negative. This is based largely on the use of the word "semantic" in Semantic Web, and the negative visions people sometimes associate with it. (See, e.g., [5] for additional discussion on this subject.) I personally think the Semantic Web has lots of promise (I've been involved with the W3C Semantic Web Activity for the past 3 years, and I don't feel like a stoker on the Titanic!). Of course, I'm less hopeful for some of the more grandiose visions that "Semantic Web" sometimes calls to mind, but we have to wait and see. Part of the problem is people who seem to make the unstated assumption that we have to reason the following way:

1. For the Semantic Web to be really useful and successful, it has to achieve all the grandiose visions anyone has ever included in a "futures" paper about it.
2. The Semantic Web will never achieve those grandiose visions.
3. Therefore the Semantic Web will never be really useful or successful.

This strikes me as being a dubious argument, precisely because, whether or not #2 is true, #1 isn't. But some people seem to want to avoid thinking about what a "success criterion" would be for the Semantic Web (or how they would know a Semantic Web if they saw one, for that matter).

Just because someone can't do impossible thing X with all this data doesn't mean someone else can't do useful thing Y with it. And perhaps X isn't impossible after all. Or, if it's impossible on the Web as a whole, it isn't impossible (and is even useful) on some subset of the Web. Most people, after all, aren't interested in the entire Web. They're interested in some particular part of it, containing the data that's important to them. If they can work out how to structure it, and develop a common vocabulary (or even an ontology), that's terrific, and we should be encouraging them. And they'll be mainly using techniques explored first in the database community.

If the Web simply managed to provide an environment in which people could hook up existing databases, work out agreements on the terms, and define (say) synonym relationships between some of the attributes, and this became the basis of lots of useful data interchange, would that be "unsuccessful" because it only involved, say 10% of the data on the Web (what percentage of the world's data is in databases of any kind at the moment, he asked rhetorically)? 10% (or even some lesser number) sure sounds like a success to me!

Finally, I think it's important to emphasize that this is *the Web* we're talking about: the largest distributed system in existence, and the largest distributed database in existence. It's accessed, *and contributed to*, by

thousands and thousands of individuals and companies, and it implicitly contains all database data (as well as all the services) that can be accessed through it. This represents a grand challenge for information systems technology if I ever saw one! Even without advanced semantics processing, the Semantic Web is an evolution of this existing Web that will provide a database in which programs can do useful things whose scope is hard to imagine. Even though this evolution will happen to different extents, at different rates, in different parts of the Web, over time, it not only will happen, it is happening right now. So, in my view, when people sometimes refer to the Semantic Web as "a vision", they seriously understate the case: "an emerging reality" would be more accurate.

References

[1] T. Berners-Lee. Conceptual graphs and the semantic web, 2001. http://www.w3.org/DesignIssues/CG.html.

[2] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. http://www.isi.edu/in-notes/rfc2396.txt, Aug. 1998.

[3] T. Bernes-Lee. *Semantic Web road map*, Sept. 1998. http://www.w3.org/DesignIssues/Semantic.html.

[4] T. Berners-Lee. *What the Semantic Web can represent*, Sept. 1998. [also appears as "What the Semantic Web is not"] http://www.w3.org/DesignIssues/RDFnot.html.

[5] M. H. Butler. Is the semantic web hype? http://www-uk.hpl.hp.com/people/marbut/isTheSemanticWebHype.pdf.

[6] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Systems*, 4(4), Dec. 1979.

[7] D. Connolly, F. van Harmelen, I. Horrocks, D. M. P. Patel-Schneider, and L. Stein. DAML+OIL (March 2001) Reference Description. W3C, Dec. 2001. http://www.w3.org/TR/daml+oil-reference.

[8] J. Date. *An Introduction to Database Systems*. Addison Wesley Longman, Inc., 2000.

[9] F. Manola and E. Miller (editors). RDF Primer, Oct. 2003. http://www.w3.org/TR/rdf-primer/.

[10] N. Guarino and A. Persidis. Evaluation framework for content standards, 2003. http://ontoweb.aifb.uni-karlsruhe.de/Members/ruben/Deliverable%203.5.

[11] R. V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford University, 1991. http://www-formal.stanford.edu/guha/index.html.

[12] M. Dean, G. Schreiber (Editors); F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein (Authors). OWL Web Ontology Language Reference. World Wide Web Consortium, Aug. 2003. http://www.w3.org/TR/owl-ref/.

[13] J. McCarthy and S. Buvac. Formalizing context (expanded notes). http://www-formal.stanford.edu/jmc/mccarthy-buvac-98/index.html.

[14] W3C (World Wide Web Consortium) RDF Core Working Group page. http://www.w3.org/2001/sw/RDFCore/.

[15] World Wide Web Consortium (W3C). *W3C (World Wide Web Consortium) Semantic Web Activity page*. http://www.w3.org/2001/sw/.

The ICS-FORTH Semantic Web Integration Middleware (SWIM)*

Vassilis Christophides, Gregory Karvounarakis,
Aimilia Magkanaraki, Dimitris Plexousakis
Inst. of Computer Science, FORTH
Vassilika Vouton, P.O.Box 1385,
GR 711 10, Heraklion, Greece
{christop, gregkar, aimilia, dp }@ics.forth.gr

Val Tannen
Dept. of Computer and Information Science
Univ. of Pennsylvania, 200 South 33rd St.
Philadelphia, PA 19104-6389, USA
val@cis.upenn.edu

1 Introduction

A cornerstone issue in the realization of the Semantic Web (SW) vision is the achievement of semantic interoperability among legacy data sources spread worldwide. In order to capture information semantics in a machine processable way, various ontology-based formalisms have been recently proposed (e.g., RDF/S [20, 5], DAML+OIL [27], OWL [10]). However, the vast majority of existing legacy data is not yet in RDF/S or any other SW language [23, 25]. As a matter of fact, most of the data is physically stored in relational database (RDB) systems and actually published on the Web or corporate intranets as *virtual* XML.

SW applications, however, require to view data as *virtual* RDF, valid instance of a domain or application specific RDF/S schema, and to be able to manipulate them with high-level query languages, such as RQL [18] or RVL [24]. Therefore, we need middleware systems that can either republish XML as RDF, or publish RDB data directly as RDF, or - even better - be capable of doing both. Sometimes the practical solution will be to rely just on the virtual XML schema and XML query interface of an existing XML publishing system. At other times, the SW publishing middleware will be built as an alternative to the XML publishing system, taking advantage of direct access to the underlying RDB management system (RDBMS). It is also possible that the SW middleware will have to integrate data in some RDBMS with data in native XML storage.

We need to deal flexibly with all these situations in a uniform framework. A decade of experience with information integration architectures based on mediators [8, 29, 26, 21] suggests that it is highly beneficial to (semi)automatically generate such systems from succinct formal specifications, rather than programming their semantics into low-level code. This greatly enhances the maintainability and reliability of the systems in an environment of often revised and shifting requirements. This paper presents the fundamental ideas for devising a comprehensive framework that allows user communities to

1. specify XML → RDF and RDB → RDF mappings;
2. verify that these mappings conform to the semantics of the employed SW ontologies;
3. compose RQL queries with these mappings and produce XML or RDB queries;
4. specify further levels of abstraction as RDF → RDF views;
5. compose RQL queries with such views;
6. perform query optimizations.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was partially supported by the EU project SeLeNe (IST-2001-39045).

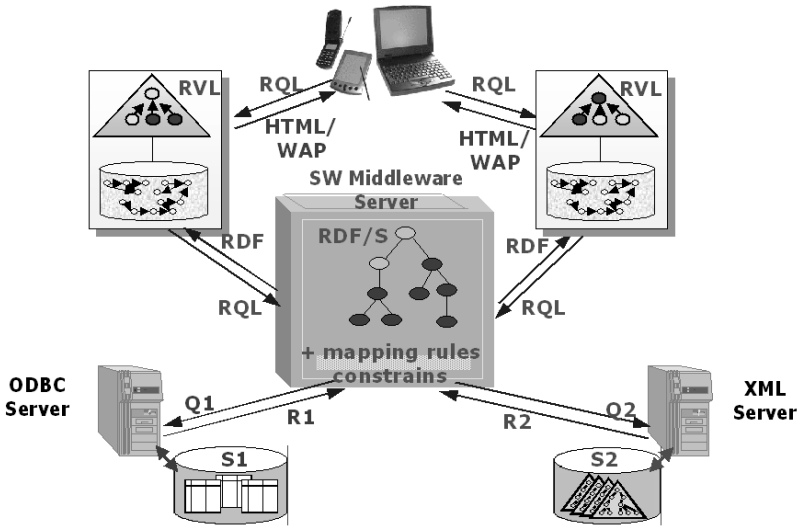


Figure 1: SWIM Architecture

The last requirement is extremely important in such systems. Queries written by humans will rarely have blatant redundancies but queries resulting from automated manipulation/generation are often very ”dumb”. Minimization techniques, sometimes taking advantage of data semantics provided by ontologies expressed in a SW language, can transform such queries into more efficient ones.

Figure 1 sketches the architecture of a SW integration middleware system that we are building, called SWIM. The lower part of the figure depicts data sources, that could be XML repositories or RDBMS. On top of these sources, we have a domain or application ontology for a particular community, expressed, for instance, in RDF/S. Mapping rules can then be used for the integration, i.e., to translate back and forth from RDF/S to the source data models. As a result, through a SWIM server we can view the underlying sources as virtual RDF repositories and use RQL to query these sources as RDF data or even define personalized views on top using RVL. In this context, the main challenge is to choose an expressive, but still tractable logical framework in which the above functionality (1-6) can be effectively supported by appropriate SW (reasoning) services.

This paper only presents our preliminary design for the SWIM framework. We expect to report on many of the technical challenges and engineering decisions in future publications. The remainder of the paper is organized as follows. Section 2 presents a motivating example for cultural data available in RDB or XML sources which can be integrated through an appropriate RDF/S schema. Section 3 presents the internal logical framework of SWIM and its use in the translation and composition of RQL queries. Section 4 touches upon the issue of query optimization by minimization using dependencies while Section 5 presents our conclusions and an outlook for further research.

2 Motivating Example and SWIM Mapping Rules

Let us assume an XML repository with cultural data, a sample of which appears in the left part of Figure 2. This data could be queried using an XML query language, such as XQuery [7]. But now, suppose we add a SWIM server on top of this XML data. For this purpose we design - or import from some community standardization body - an RDF/S cultural schema, as the one depicted in the top part of Figure 2. Now we can formulate queries using an RDF query language by employing only few abstract classes and properties from our mediation RDF/S schema. For example, the following RQL query returns the names of the artists (sculptors or painters) whose work is exhibited in the “Reina Sofia” museum:

```
SELECT Z
FROM   {X}creates.exhibited.denom{Y}, {X}name{Z}
WHERE  Y = "Reina Sofia"
```

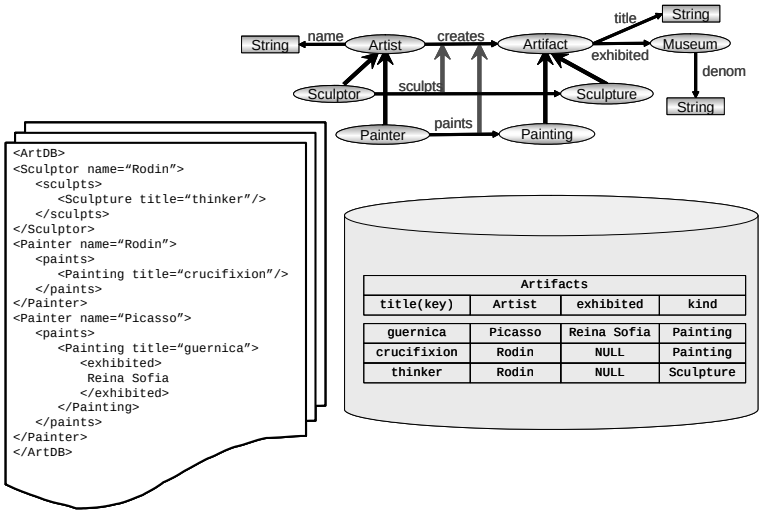


Figure 2: Example of XML/RDF sources and Mediation RDF/S schema

We can observe that the RDF/S layer is completely virtual. The actual data can only be queried using an XML language. Hence, the RQL query we saw needs to be reformulated by the middleware into an XML query. This reformulation should be guided by a formal description of the relationship between the XML and the RDF data, for example a *mapping* from XML to RDF. The question that normally arises is: *how do we express formally such mappings?*

The rich theory developed in the relational case has identified classes of queries and mappings (views) that can be manipulated formally such that various problems like query containment, composing queries with views and rewriting queries with views are algorithmically solvable [1, 15]. These problems can also be solved in the presence of certain classes of relational constraints [1, 11, 14]. We shall try to rely as much as possible on a well-known and robust formalism: conjunctive queries and views, and embedded implicational dependencies [1]. The results about queries and views are easily extended with union, therefore dealing with the positive existential first-order queries also known as non-recursive Datalog. The dependencies can easily be extended with disjunction [12].

To define XML→RDF mappings we will use an analog to the relational queries just mentioned. We use the same logical shape as that of Datalog rules, but instead of relational atoms, we use XPath atoms in the bodies (this is similar to the XBind queries of [14]). For example, the XPath atom `.//Painting (X,Y)` is satisfied by any valuation that maps `X` and `Y` to element nodes in the XML document, such that `Y` has tag `Painting` and is a descendant of `X`. The heads of the rules define RDF instances in the style of the `VIEW` clause employed by the RDF/S view definition language RVL [24]. So, as part of the mapping we can use rules, such as:

```
Painter(X) :-  (//Painter) (X)     Sculptor(X) :-  (//Sculptor) (X)
```

to define the (direct) extent (i.e., the set of direct instances) of the classes `Painter` and `Sculptor` in the virtual RDF layer. Property extents can be also defined in the same style:

```
paints(X,Y) :-  (//Painter) (X),  (.//Painting) (X,Y)
```

Note that this mapping is not always straightforward, since there usually exist schematic and semantic discrepancies between the source and the middleware schema. For example, class inheritance is not expressed in the XML document. Moreover, properties (let alone property inheritance) `creates`, `paints` and `sculpts` are not used explicitly in the XML document. We expect SWIM to be able to take the RQL query and the XML→RDF mapping given above and produce an XML query (e.g., an XQuery). We will discuss in Section 3 how this reformulation can be done.

In addition of being available in XML, the cultural data may be available through an RDBMS, for instance in a table as illustrated in the right part of Figure 2. As for XML, there is an RDB→RDF mapping which is also expressed in a mixed language, where instead of XPath atoms we can use standard Datalog atoms:

```
Painter(X) :- Artifacts(_, X, _, "Painting")
paints(X,Y):- Artifacts(Y, X, _, "Painting")
name(X,Y)  :- Artifact(_, X, _, "Painting"), Y=X
name(X,Y)  :- Artifact(_, X, _, "Sculpture"), Y=X
```

As in the case of XML, there may also be discrepancies in the RDB→RDF mapping. For instance, in our example, the classification of an Artist to `Painter` or `Sculptor` is determined by the value of the attribute `kind`, i.e., schema information is "encoded" inside data values. Again, the SW middleware should be able to automatically reformulate the RQL query, using this mapping, into a relational query, presumably SQL [17].

3 Query Mediation in SWIM

We need an internal logical framework that captures RDF/S semantics, as well as queries, so that we can "virtually populate" given RDF/S schemas. It should also capture - to any needed extent - the XML and RDB semantics. As we showed in the previous section, Datalog-like rules are very convenient for expressing mappings, even across data models, such as XML→ RDF. Based on the experience of [13, 11] of performing XML query reformulation via translation in a first-order, relational framework, we propose to follow the same approach for RDF, in order to translate both queries and mappings into this framework.

3.1 SWIM Internal Logical Framework

The SWIM internal logic framework employs first-order relations together with some first-order constraints to model RDF/S. It is convenient to use a signature with three sorts: Resource, Property, Class[1]. The relations used have the following meaning:

- C_EXT(c, x) iff the resource x is in the *proper extent* (i.e., it is a direct instance) of class c. In RDF, class extents can overlap, due to multiple classification of resources.
- C_SUB(c, d) iff c is a (not necessarily direct) subclass of d.
- PROP(c, p, d) iff class c is the domain and class d is the range of property p.
- P_EXT(x, p, y) iff (x, y) is in the *proper extent* (i.e., it is a direct instance) of property p. In our model instances of properties are represented as ordered pairs of the resources they connect.
- P_SUB(p, q) iff p is a (not necessarily direct) subproperty of q.

The relations must satisfy some *built-in* RDF/S constraints which are considered by RQL. In particular, the domain and range of a property must be unique, while the subclass and subproperty relations must be reflexive, transitive and satisfy the following *subproperty/subclass compatibility* constraint:

$$\forall\, a, p, b, c, q, d\ \mathsf{P\_SUB}(q, p)\ \wedge\ \mathsf{PROP}(a, p, b)\ \wedge\ \mathsf{PROP}(c, q, d) \longrightarrow\ \mathsf{C\_SUB}(c, a)\ \wedge\ \mathsf{C\_SUB}(d, b)$$

This means that if q is a subproperty of p, the domain and range of q are subclasses of the domain and range of p, respectively. Finally, we have the *property-class extent compatibility* constraint, i.e., any instance of a property p connects a pair of instances of some subclasses of the domain and range of p, respectively:

$$\forall\, a, p, b, x, y\ \mathsf{PROP}(a, p, b)\ \wedge\ \mathsf{P\_EXT}(x, p, y) \longrightarrow\ \exists\, c, d\ \mathsf{C\_SUB}(c, a)\ \wedge\ \mathsf{C\_SUB}(d, b)\ \wedge\ \mathsf{C\_EXT}(c, x)\ \wedge\ \mathsf{C\_EXT}(d, y)$$

Let Δ_{RDF} be the set of dependencies (constraints) used to axiomatize the internal RDF/S model.

[1] For simplicity reasons, we ignore metaclasses and metaproperties in this discussion but they can be handled easily in the same way.

Theorem 1: It is decidable whether $\Delta_{\mathrm{RDF}} \models d$ and whether $\Delta_{\mathrm{RDF}} \models Q_1 \sqsubseteq Q_2$, where d is an embedded implicational dependency, Q_1, Q_2 are conjunctive queries and $\sqsubseteq$ is query containment.

It is straightforward to translate the information of an RDF/S schema to the SWIM internal framework as a set of relational facts (in Datalog parlance—an extensional database), involving the relations C_SUB, PROP, P_SUB as well as the names of classes and properties in the schema as constants. Some of the facts obtained from the schema in Figure 2:

C_SUB(Painting, Artifact) PROP(Artist, name, String) P_SUB(sculpts, creates)

Note that this set of facts will include all C_SUB and P_SUB reflexivity instances and will be "closed" under transitivity and under subproperty/subclass compatibility.

3.2 Translation of RQL Queries

RQL is a powerful language for querying smoothly both RDF/S schemas and their instances. An RQL *conjunctive* query has the form $ans(\bar{X}) :- C_1, \ldots, C_n$ where C_i's are either RQL class or property patterns (as they appear in the RQL FROM clause) or equalities involving variables and/or constants and $\bar{X}$ is a tuple of variables or constants (range restrictions [1] are also required). Many RQL queries are in fact conjunctive queries, e.g., the query given in Section 2 can be written:

```
ans(Z):- {X}creates{V}, {V}exhibited{W}, {W}denom{Y},
         {X}name{Z}, Y="Reina Sofia"
```

Conjunctive RQL queries can then be translated into relational conjunctive queries in the SWIM internal logical framework. Indeed, according to the declarative semantics in [18], RQL patterns have the same meaning as conjunctions of relational atoms. For example:

RQL Pattern	*Internal SWIM Translation*
$\{X;\$C\}@P\{Y;\$D\}$	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_EXT(c,x), C_EXT(d,y)
$\{X\}@P\{Y\}$	P_SUB(q,p), P_EXT(x,q,y)

In the above RQL patterns, X, Y are resource variables, $\$C, \$D$ are class variables (and can be replaced with constant class names), and $@P$ is a property variable (that also can be replaced by a constant property name). Using these patterns, the RQL conjunctive query above translates internally to the following Datalog rule:

$$\begin{aligned} ans(z) \; :- \; & \text{P\_SUB}(q_1, \texttt{creates}),\ \text{P\_EXT}(x, q_1, v), \\ & \text{P\_SUB}(q_2, \texttt{exhibited}),\ \text{P\_EXT}(v, q_2, w), \\ & \text{P\_SUB}(q_3, \texttt{denom}),\ \text{P\_EXT}(w, q_3, \text{"}\texttt{Reina Sofia}\text{"}), \\ & \text{P\_SUB}(q_4, \texttt{name}),\ \text{P\_EXT}(x, q_4, z) \end{aligned}$$

3.3 Composing Queries with Mappings

Starting with the internal translation of the query, we perform an interesting *partial evaluation* using the RDF schema information, i.e., we evaluate first the schema-part of the query, namely the P_SUB expressions. This is related to partial evaluation of Datalog programs [4]. Because some atoms (e.g., P_SUB$(q_1, \texttt{creates})$) match more than one fact in the schema, what was a single conjunctive query now becomes a (non-recursive) Datalog program. Here is one of the rules in our example (the other two feature `sculpts` and `creates`):

$$\begin{aligned} ans(z) :- \; & \text{P\_EXT}(x, \texttt{paints}, v),\ \text{P\_EXT}(v, \texttt{exhibited}, w), \\ & \text{P\_EXT}(w, \texttt{denom}, \text{"}\texttt{Reina Sofia}\text{"}),\ \text{P\_EXT}(x, \texttt{name}, z) \end{aligned}$$

The next step is to translate into the SWIM internal framework the heads of the rules that define the mappings. For example, a rule (expressed in RVL syntax) defining the extent of the class `Painter` has the head

`Painter(X)`. We translate this into C_EXT(`Painter`, x). In the same style we can translate the rule defining the extent of the property `paints(X,Y)` into P_EXT(x, `paints`, y). Thus, the mapping becomes a (non-recursive) Datalog-like program with XPath atoms for the XML→RDF case and a plain non-recursive Datalog program for the RDB→RDF case. The composition of the query and the mapping is now simply the composition of two Datalog programs.

To finish the reformulation, we must still eliminate the intermediate predicates C_EXT, P_EXT because they are not part of the data sources. This is done with standard matching/substitution but it may increase (square, in fact) the number of rules. In the examples we have looked at so far, however, the resulting union of conjunctive queries can be minimized significantly because many of the rules are unsatisfiable and hence can be discarded.

4 RQL Query Reformulation and Optimization

Continuing the example from Section 3.3, we compose the query with the mapping for the RDB→RDF case. After eliminating the intermediate predicates C_EXT and P_EXT we obtain a Datalog program with eight rules. Six of these rules, however, are unsatisfiable because their bodies equate distinct constants. Moreover, standard conjunctive query minimization [1] applies to the remaining two rules. The final reformulated query, after optimizations, for the RDB→RDF case is the following union of conjunctive query (a non-recursive Datalog program with two rules):

```
ans(z) :- Artifacts(x, z, "Reina Sofia", "Painting")
ans(z) :- Artifacts(x, z, "Reina Sofia", "Sculpture")
```

Similar transformations are performed in the case of the XML→RDF mapping. We also encounter six unsatisfiable rules: for example in a rule containing both (`//Sculpture`) (y) and (`.//Painting`) (x, y) there is no valuation for y since an XML element cannot have two different tags (i.e., `Sculpture` and `Painting`). The reformulated query for the XML→RDF case is given below:

```
ans(z) :- (//Painter)(x),  (./@name)(x, z),
          (//Painter)(x),  (./Painting)(x, y),
          (//Painting)(y), (./@exhibited)(y, "Reina Sofia")

ans(z) :- (//Sculptor)(x),  (./@name)(x, z),
          (//Sculptor)(x),  (./Sculpture)(x, y),
          (//Sculpture)(y), (./@exhibited)(y, "Reina Sofia")
```

However, the problem of deciding satisfiability of rules with XPath atoms is a difficult one. We expect that the techniques developed in [14] will help with this problem and more generally with the minimization of such queries.

The optimizations we have seen so far do not take into account the specifics of the RDF/S semantics considered by RQL. However, once we have encoded this semantics into the relational dependencies Δ_{RDF} (see Section 3.1) we can use Δ_{RDF} in minimizing queries. For example, by translating into the internal model and by using minimization under dependencies done with the Chase&Backchase algorithm [11] it is possible to show that the conjunctive RQL queries of the form

```
ans(X,@P,Y) :- {X;$C}@P{Y;$D}, rest(X,@P,Y)
```

minimize to (the internal translation of):

```
ans(X,@P,Y) :- {X}@P{Y}, rest(X,@P,Y)
```

thus eliminating several redundant scans over the class variables $\$C$ and $\$D$ (`rest(X,@P,Y)` stands for a boolean predicate whose variables are X, $@P$ and Y only). It should be stressed that if we just translate these queries into SWIM internal conjunctive queries, the results are not equivalent in the absence of Δ_{RDF}. The examples we saw in this section serve as a guide for design decisions regarding what kind of optimization facilities need to be incorporated into SWIM.

5 Conclusions and Future Work

In this paper we presented the principles underlying the design of SWIM (Semantic Web Integration Middleware) and described the components that achieve semantic integration by mapping XML and relational data to RDF. The unifying framework proposed relies on the use of Datalog-like rules for expressing the mappings and reformulating RQL queries. Furthermore, this framework permits the optimization of RQL queries as well as their composition with the specified mappings in order to produce XML or relational database queries. Previous projects sharing similar motivations are described in [2, 3], [28] and [16]. Our approach is closest to that of [2, 3], while using a more expressive language for the specification of mappings and a different ontology query language. The papers [22, 6] present formal specifications of mappings from less structured schemas such as XML and relational to more structured schemas of the same level of complexity as RDF. Languages similar to our Datalog with XPath atoms are also used, for example, in [9, 19]. Finally, compared to the Datalog framework for RDF/S-based query mediation of [28], SWIM ensures the compositionality of queries with views and mappings, as well as, supports advanced optimization and verification services.

Several issues require further investigation. Specifically, we have dealt so far with the case of conjunctive RQL queries and conjunctive RVL view definitions. In both these cases we obtain a translation into non-recursive Datalog programs to which we can apply well-known optimization techniques and for which the problem of determining the consistency of the mappings is decidable. We intend to study the conditions under which similar results can be obtained for a broader class of RQL queries and RVL view definitions. Another issue is the exploitation of knowledge about the source schemas and data in order to perform further optimizations during the reformulation process. SWIM's internal model can also accommodate constraints such as the ones expressible in OWL [10]. It will be interesting to study the optimization potential that stems from the use of such constraints (e.g., uniqueness or disjointness constraints) in query reformulation / minimization.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In *Proc. of the International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 2002.

[3] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Querying XML Sources Using an Ontology-Based Mediator. In *Proceedings of International Conf. on Cooperative Information Systems (CoopIS)*, pages 429–448, Irvine, California, USA, Nov. 2002.

[4] K. Benkerimi and J. Lloyd. A Partial Evaluation Procedure for Logic Programs. In *Proceedings of the North American Conference on Logic Programs*, Austin, TX, USA, 1990. MIT Press.

[5] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, Mar. 27, 2000.

[6] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, pages 270–284, Yokohama, Japan, Nov. 2001.

[7] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: An XML Query Language. W3C Working Draft, May 2003. See http://www.w3.org/TR/xquery/.

[8] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 177–188, Seattle, WA, USA, June 1998.

[9] S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 271–280, Roma, Italy, Sept. 2001.

[10] M. Dean, D. Connolly, F. V. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference Version 1.0, W3C Working Draft. Technical report, W3C, Dec. 12, 2002.

[11] A. Deutsch, L. Popa, and V. Tannen. Physical Data Independence, Constraints, and Optimization with Universal Plans. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 459–470, Edinburgh, Scotland, UK, Sept. 1999.

[12] A. Deutsch and V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Proc. of International Workshop on Database Programming Languages*, pages 21–39, Frascati, Italy, 2001.

[13] A. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. In *Proc. of International Conf. on Very Large Databases (VLDB)*, Berlin, Germany, Sept. 2003.

[14] A. Deutsch and V. Tannen. Reformulation of XML Queries and Constraints. In *Proc. of International Conf. on Database Theory (ICDT)*, pages 255–241, Siena, Italy, Jan. 2003.

[15] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[16] A. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proc. of International World Wide Web Conf.*, pages 556–567, Budapest, Hungary, 2003.

[17] ISO/IEC 9075: Information technology – Database Languages – SQL, 1999.

[18] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the Semantic Web with RQL. *Computer Networks*, 42(5):617–640, Aug. 2003.

[19] L. Lakshmanan and F. Sadri. XML Interoperability. In *Proc. of the International Workshop on the Web and Databases (WebDB)*, San Diego, California, USA, June 2003.

[20] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, Feb. 1999. Available at http://www.w3.org/TR/REC-rdf-syntax.

[21] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of International Conf. on Very Large Databases (VLDB)*, pages 251–262, Bombay, India, Sept. 1996.

[22] B. Ludäscher, A. Gupta, and M. Martone. Model-Based Mediation with Domain Maps. In *Proc. of IEEE International Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, Apr. 2001.

[23] A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In *Proc. of the International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 2002.

[24] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the Semantic Web Through RVL Lenses. In *Proc. of the Int'l Semantic Web Conf. (ISWC)*, Sanibel Island, Florida, Oct. 2003.

[25] L. Mignet, D. Barbosa, and P. Veltri. The XML web: a first study. In *Proc. of International World Wide Web Conf.*, pages 500–510, Budapest, Hungary, May 2003.

[26] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proc. of IEEE International Conf. on Data Engineering (ICDE)*, pages 251–260, Taipei, Taiwan, Mar. 1995.

[27] F. van Harmelen, P. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL ontology markup language. http://www.daml.org/2001/03/reference.html, Mar. 2001.

[28] W. Nejdl and B. Wolf and C. Qu and S. Decker and M. Sintek and A. Naeve and M. Nilsson and M. Palmér and T. Risch. EDUTELLA: A p2p networking infrastructure based on rdf. In *Proc. of International World Wide Web Conf.*, Honolulu, Hawaii, USA, May 2002.

[29] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

Information Integration and the Semantic Web

Laks V. S. Lakshmanan*
University of British Columbia
laks@cs.ubc.ca

Fereidoon Sadri†
University of NC, Greensboro
sadri@uncg.edu

1 Introduction

Information integration and interoperability among information sources are related problems that have received significant attention since early days of computer information processing. Initially, for a few decades, the focus was on integration/interoperability for a relatively small number of sources. This is the setting encountered in traditional business and service applications, for example when two companies merge or several services interoperate (which requires the integration of their information systems). Much of the work in this context of federated or multi-databases focused on integrating schemas by defining a global schema in an expressive data model and defining mappings from local schemas to the global one [19]. More recently, in the context of integration of data sources on the internet, the so-called *global-as-view* (GAV) and *local-as-view* (LAV) paradigms have emerged out of projects such as TSIMMIS [20] and Information Manifold (IM) [12].

Recently, the advent of XML as a standard for online data interchange holds much promise toward promoting interoperability and data integration. The focus has also shifted to one of providing integration and interoperability among a large number of independent and autonomous information sources. But XML, being a syntactic model, in itself cannot make interoperability happen automatically. Two main challenges to overcome are: (i) data sources may model XML data in heterogeneous ways, e.g., using different nestings or groupings or interchanging elements and attributes, and (ii) sources may employ different terminology, a classic problem even in multi-database interoperability. While earlier approaches to integration can be extended to handle XML, they suffer from the significant overhead of having to design a commonly agreed upon global schema. Can we interoperate without this overhead?

Indeed, there are a few recent proposals that do overcome the need for a global schema – Halevy et al. [9] and Miller et al. [15]. In a nutshell, both of these approaches rely on source to source mappings. One problem here is that requiring such mappings for all pairs is too tedious and cumbersome since the mappings, even if generated semi-automatically, still require significant manual intervention to create. In order to mitigate this, one can merely insist, as [9] does, that the graph of pairs with available mappings be connected. A second problem is that when a source s_i is mapped to source s_j, if s_j does not have some of the concepts present in s_i, then they will be lost. E.g., s_i may include the `ISBN` for all its books while s_j may not.

Independently of all this, recently there has been much excitement around the *Semantic Web* [18] initiative, coming as it does with its own host of technologies such as resource description framework (RDF) [16] and ontology description languages such as DAML+OIL and OWL [5, 14]. The promise of the Semantic Web is to expose the semantics of the information content of web resources (including text, audio, video, etc.) thus taking the web to a higher semantic level, enabling easy exchange of data and applications.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Supported by NSERC.
†Supported by NSF grant IIS-0083312.

Our work on interoperability and information integration has been motivated by asking how we can take advantage of the Semantic Web initiative. Our thesis is that each source('s administrator) interested in participating in data and application sharing should "enrich" itself with adequate *semantic declarations*, providing a semantic view of the information contents. These declarations would essentially expose the semantic concepts present in the source using common, but possibly application or even community specific terminology, in the spirit of the Semantic Web initiative and frameworks such as RDF. Further infrastructure such as application wide ontologies could exist and help relate terminologies used across different communities within an application domain or even beyond. Queries across data sources so enriched can be composed over the vocabulary consisting of community wide RDF vocabulary or application wide ontology. They can be agnostic to the specific modeling constructs or terminologies employed in the local sources (i.e., below their semantic views). Query processing and optimization is the responsibility of the interoperability system, and is carried out in consultation with ontology servers that provide the common terminologies [11]. We should emphasize that we are not assuming a single standard ontology, not even for a given application. Rather, we expect that, within any one application domain, a limited number of ontologies will become widely used by the interested communities.

Below we provide a summary of of our ongoing research on the X-DARES-U (XML DAta waREhouse with Semantic enrichment at UBC/UNCG) project [10]. We also discuss inherent challenges – both technical and social in making interoperability and information integration possible on the internet scale. The need for a killer application has been realized for some time in the Semantic Web community. We argue that interoperability and integration offer such an application.

2 X-DARES-U Overview

The X-DARES-U approach is based on the existence of an infrastructure constituted by the following components: A semantic view provided by data sources, data mappings to represent source data in its semantic view, existence of ontologies and inter-ontology mappings, tools for inferring integrity constraints over mappings, and query optimization techniques. In this section, we briefly overview each of these.

2.1 Semantic Declarations

Local information sources can be XML documents, relational databases, spreadsheet documents, or servers of other data formats. A local source can join an interoperability effort as long as it can provide semantic declarations and mappings as discussed below. *We should emphasize that these declarations are local.* No overhead in providing a global schema is needed in our approach. This is in contrast to traditional approaches to data integration where significant overhead is incurred in developing a global schema for the federation.

The semantic content of an information source is declared in the form of *concepts* and *properties* represented in the source. This is in the spirit of knowledge representation languages and ontologies that use concepts (or classes) and properties (or relationships) for semantic representation. Each source chooses an appropriate semantic model based on terminologies that are either chosen from a common ontology, or it can create and publish its own terminology. A mapping of the source's native data format to the semantic model is then specified. If necessary, the source also specifies a mapping from its semantic model to a better known ontology. These declarations can be provided in a number of ways, including Semantic Web tools such as RDF Schema [17]. We will use the term *semantic content model* (*semantic model*, for short) of a local source to refer to the semantic declarations, in the form of concept/property constructs, as discussed above.

Example 1 (Semantic Model): Consider a federation of catalog sales stores maintaining their merchandise information in various systems (XML, relational, etc...) and formats. The DTD of one of these stores is shown below (for additional DTDs see the complete example in [11]).

```
<!ELEMENT store (warehouse*)>
<!ELEMENT warehouse (city, state, item*)>
<!ELEMENT item (id, name, description)>
<!ATTLIST warehouse id ID #REQUIRED>
```

A possible semantic model for these sources consists of the concepts `item`, `warehouse`, `city`, `state`, and properties `item-itemId`, `item-name`, `item-description`, `item-warehouse`, `warehouse-warehouseId`, `warehouse-city`, `warehouse-state`. Motivated by RDF, the first parameter of each property is a URI. The second parameter can be a URI or a string. Note that we need to provide the mapping between URIs and database-style identifiers, such as `itemId`. The properties `item-itemId` and `warehouse-warehouseId` do precisely this.

Once the concepts and properties for the declaration of the semantic content of a source are determined, the local DBA or user should provide the mappings from the local data into the selected semantic content model. This, in effect, is providing a *semantic view* of the information content of the source in its semantic model. We envision semi-automated tools that can assist DBA or user in this task [11]. The mappings can be represented succinctly as rules using a combination of XPath and Datalog as in [11]. Providing the mappings in a more practical representation, such as an XSLT code that generates an RDF document (the view) as output, is also possible. The mapping specification tool can be configured to produce a number of different presentations.

Example 2 (Mappings): The mapping rule for the property `item-warehouse` is shown below. The functions f_I and f_W are *URI-generating* functions that produce URIs from database-style IDs. Please refer to [11] for details and additional mappings.

`item-warehouse(`f_I`($I),`f_W`($W))` $\leftarrow$ `source1/store/warehouse $X, $X/item/id $I, $X/@id $W`

2.2 Query Formulation

Consider a federation of information sources enhanced with semantic declarations and mappings as discussed in the previous section. A *global* query is formulated in terms of concepts and properties. The interoperability system is responsible for the optimization and execution of the query. It may require services from ontology brokers and directory servers to reconcile ontological and taxonomic differences in local semantic models as well as local data. For brevity, we suppress this detail and focus on the case of single ontology in the sequel.

As mentioned above, a query is formulated in terms of concepts and properties, which may be at any level of an ontology. Let p be a property mentioned in the query (such as `item-warehouse` in our example). A source i may have this property in its semantic model. If so, it contains a *fragment* of p, which we will denote by p_i. The *global content* of p is the collection of all of its local fragments (p_i). What exactly is the meaning of a *collection* of fragments? We assume, in general, that local sources have *overlapping* data. Hence, *collection* should provide a mechanism to handle the overlap. If data among sources are *consistent*, then the collection is simply a (possibly duplicate eliminating) union. More complicated mechanisms are needed when sources contain inconsistent and/or uncertain data. Here we assume consistency within each source as well as among sources, and use (possibly duplicate eliminating) union as the semantics of collection.

2.3 Architecture, Query Processing and Optimization

A coordinator-based architecture for semantic enrichment and query processing is suggested in Figure 1. Other architectures such as peer-to-peer are also possible. The global query is submitted to the *coordinator*. The coordinator is in charge of resolving ontological differences, and supervising query optimization and execution. Query processing itself can be carried out in many different ways utilizing a combination of sources and the coordinator. We briefly discuss some of these options below.

The simplest approach to query processing in this architecture is for the coordinator to (1) resolve ontological differences with the aid of ontology servers, (2) request each source to materialize their fragments of properties that appear in the query, (3) collect the fragments from sources, (4) perform taxonomic resolution on the data and materialize global properties, and (5) execute the query. Although conceptually simple, this approach can be inefficient in practice.

An alternative approach is for the coordinator to decompose the query and submit the subqueries to the sources for processing. Then collect these results, and possibly execute further queries on these intermediate

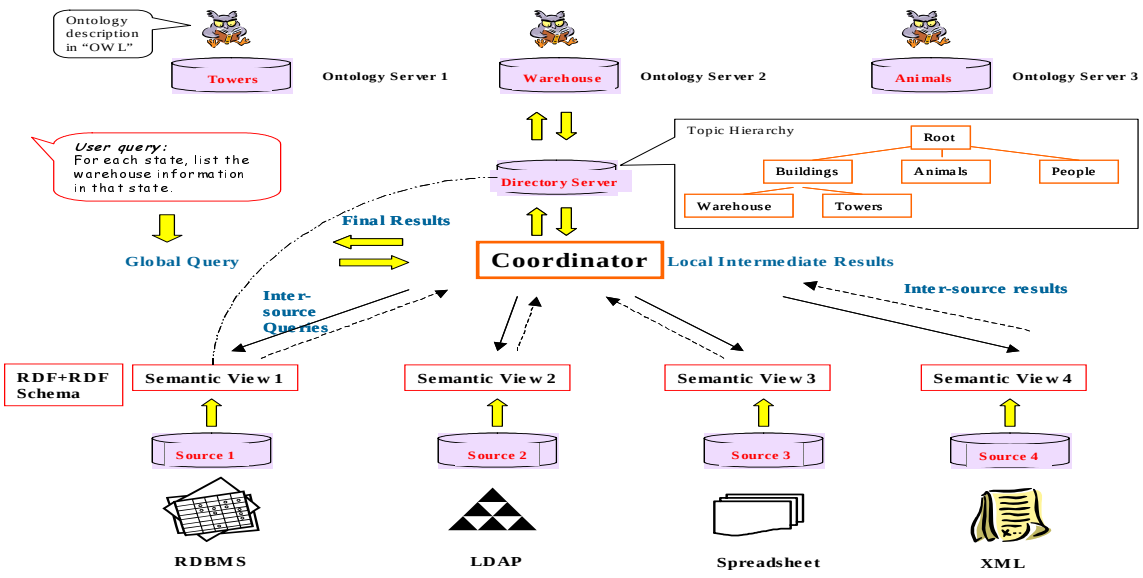


Figure 1: An architecture for semantic markup and query processing.

results to generate the final answer. Interesting optimization opportunities are possible in this alternative plan. We will use a simple example to discuss this approach. More details can be found in [11].

Consider a simple global query that involves the join of two properties p and q, $p \bowtie q$. Suppose there are n sources, $s_1, \ldots, s_n$. The expansion of the query results in the following subqueries: $p_1 \bowtie q_1$, $\ldots$, $p_1 \bowtie q_n$, $\ldots p_n \bowtie q_1$, $\ldots$, $p_n \bowtie q_n$. Among these subqueries, there are n *local* subqueries $p_1 \bowtie q_1$, $\ldots$, $p_n \bowtie q_n$. These subqueries refer only to fragments in a single source and can be processed solely by that source. The rest need fragments from different sources and are called *inter-source* sub-queries. In general, if the query involves k properties and the number of sources is n, there can be up to n local sub-queries and $O(n^k)$ inter-source sub-queries. An important optimization opportunity here is that if certain integrity constraints (key and foreign key constraints) hold then some inter-source sub-queries need not be processed at all. E.g., suppose the functional dependency $q : Y \rightarrow Z$ holds for property q, and according to source s_i, the foreign key constraint (RIC) $p_i[Y] \subseteq q_i[Y]$ holds, where Y is the join argument in $p \bowtie q$. Then, we can show that all inter-source queries involving p_i are redundant. Indeed, we have characterized cases where inter-source sub-queries can be avoided, and have also developed algorithms to derive foreign-key integrity constraints such as above. Local sub-queries can be reformulated in terms of the local source data structure (*e.g.* XML document), and executed (partly or in full) locally. Interested readers are referred to [11] for details. We have also developed optimization techniques for the execution of inter-source sub-queries for cases that require inter-source processing. Note that previous work on distributed query optimization can be leveraged, although the presence of ontologies, availability of semantic knowledge in the form of integrity constraints, and the differences in the data storage formats, mean that the cost model for the optimization should be revisited. More work is needed in the future to address these issues thoroughly. While distributed query optimization is not new, the twists brought about as a result of the Semantic Web are indeed new and challenging.

3 Under the covers: What is needed?

In this section we discuss the question "what is needed to make X-DARES-U approach, or more generally, wide scale interoperability and integration successful?" We propose the following criteria for measuring success: (1) It should be relatively easy for a new source to join a federation of interoperable sources, (2) Query formulation should be simple, (3) Query processing should have an acceptable performance, and (4) Last, but not least, the

system should be scalable to large number of sources.

Tools and techniques from several domains are needed for the ultimate success of this project. Some of these are already available, some are at different stages of development, and some still need to be developed. We provide a summary below.

3.1 Semantic Web tools and techniques

The Semantic Web initiative has taken the first important steps in realizing languages and tools needed for interoperability. Resource Description Language (RDF) [16], RDF Vocabulary Description Language (RDF Schema) [17], the DARPA Agent Markup Language (DAML) [5], and OWL Web Ontology Language [14] are examples of languages that are already developed or are under development. They facilitate semantic markups and ontology management. Many companies and research institutes are actively developing tools and systems for these languages (see, for example, [7]).

While ontology description languages are quite expressive, in our opinion, more powerful tools are needed for ontology management, especially for describing relationships among concepts and properties of two or more different independently-designed ontologies (please also see related discussions on ontology languages and ontology maintenance in Section 3.2). There is a fine balance between expressive power of languages and complexity (efficiency) of operations in these languages. The key is to maximize the expressive power while maintaining an acceptable performance. There is also an acute need for efficient *ontology brokers* to reconcile ontological and taxonomic differences among sources in order to make interoperability possible.

3.2 Theoretical and conceptual questions and solutions

Knowledge representation and ontology description and maintenance have been active research areas for decades. The Semantic Web initiative has contributed new languages, such as DAML and OWL, which pay special attention to *efficiency* as well as *expressive power*. Description Logics (DL) provide the mathematical basis for ontology languages. In fact, OWL itself has been inspired and influenced by DLs [1]. When multiple ontologies are used by information sources, there is a need to determine and declare inter-ontology relationships. An expressive language is needed for the declaration of these relationships, and (semi)automatic techniques are needed for their discovery. These topics, sometimes referred to as *ontology integration*, *ontology matching*, and *ontology alignment* are active areas of research [4]. AI research is rich with respect to matching and similarity-based algorithms, and some of these techniques have been applied recently to schema and ontology matching [8, 13]. However, several fundamental questions have yet to be answered. For instance, how are we to choose an appropriate semantic model and mapping for a source? What (technical and other) incentives can we offer a source administrator other than the promise that they can join a "federation"?

Optimization and processing of (global) queries in a multi-source environment poses many challenges, some of which were discussed in Section 2.3. Of particular interest is the case where multiple ontologies are employed. For example, consider a (global) query Q that uses an ontology $\mathcal{O}$, and assume a source i uses a different ontology $\mathcal{O}_i$. The algorithm that generates the local sub-query Q_i at source i needs to perform a *maximal sound rewrite* of Q in terms of the ontology $\mathcal{O}_i$, in the sense that answers to the rewritten query are provably answers to the original query and there is no rewrite which produces a proper superset of such valid answers. The algorithm for and the complexity of this task critically depend on the expressive power of the language used to declare inter-ontology relationships. We have explored this question for a simple language that allows concept/property equivalence as well as sub-concept and sub-property declarations in [11]. Further investigation is needed for more expressive languages.

An orthogonal issue in query optimization is the avoidance or minimization of redundant processing. While some initial work on identifying and inferring types of integrity constraints useful for this form of query optimization has been done in [6, 11], more work is needed on this question.

When overlapping information across sources is consistent, taking their union (with or without duplicate elimination) is just one possible way of combining them. Other operators (e.g., intersection) may be meaningful

under some circumstances. When there *is* inconsistency across data sources, how should we resolve them? How can we answer queries reliably and efficiently in this case? Some proposed approaches in database research are based on accociating *certainty* (or *reliability*) factors with data, either directly or indirectly by associating these factors with information sources. In the Semantic Web approach, this can be accomplished elegantly by *reification*. Storage and processing efficiency for reified statements becomes of utmost importance for this type of application that makes very heavy use of reification. The implementors of the Semantic Web framework Jena have addressed this issue in their recent (Jena2) release [21].

There are thus many deep semantic, algorithmic, and complexity questions that provide fertile ground for further research.

3.3 Database tools and techniques

Efficiency is one of the most important goals of an interoperability system that incorporates large number of sources. We need to investigate a multitude of techniques from database research (such as indexing, caching, and optimization using materialized views), as well as develop new techniques related to directory organization and maintenance, ontology broker and server systems, and web-based peer-to-peer systems to harness the complexity of scale. Our work in [11] investigates interesting *semantic* optimization, *i.e.* optimization made possible by data integrity constraints such as key and foreign key constraints.

The *cost-based* query optimization technique has been applied quite successfully in (centralized and distributed) database systems. In this approach, a small number of query execution plans are generated for a given query, and the cost of each plan is estimated. The plan with the lowest estimated cost is then selected for processing the query. We believe cost-based optimization techniques developed for distributed databases should prove useful but will need to be gracefully adapted to deal with the following challenges: (i) The search space is more explosive than for distributed databases which have a small number of sources; (ii) Statistics are harder to collect in a wide distributed context such as ours; (iii) Server loads and network traffic may change dynamically and rapidly. As discussed in Section 2.3, there are many important differences in this application compared to classical distributed query processing that arise from (i) the presence of ontologies, (ii) the availability of semantic knowledge that can be leveraged for substantial query optimization, and (iii) the differences in the data storage formats. We believe large scale (XML) data integration and interoperability will be a killer application for the Semantic Web. For this application to be successful, the aforementioned challenges must be addressed.

3.4 Standards

Any information integration and interoperability system is ultimately dependent on the establishment of standards. It is unrealistic to expect that a single, universally accepted ontology will be established and used by every information source. Even if we restrict ourselves to a single application domain (e.g., purchase order processing), we cannot expect a single ontology that is accepted by everybody. On the other hand, it is unlikely that every source involved in an application will use a different ontology. Instead, community wide ontologies offer a nice hierarchical level in between. Our thesis is that for each application domain a relatively small number of (community wide) ontologies will be established and used by a large percentage of information sources in that domain. Efforts towards this goal are already underway by various groups of parties engaged in specific applications [3]. More efforts for integrating multiple ontologies related to a single application domain are needed and need to be done in an incremental fashion.

4 Conclusions

Interoperability and information integration is a long standing unresolved problem with tremendous application pull. The advent of XML has created a framework akin to low level plumbing in which this problem can be profitably studied. Our vision is wide scale interoperability and integration. The recent Semantic Web initiative with its host of technologies brings several relevant and useful tools and techniques to the table which we can exploit in the interoperability context. However, this raises a variety of challenges and questions at various

levels – from fundamental theoretical to systems, to technological to social. In our X-DARES-U project, we are investigating several of these technical challenges. There is a well recognized need for a killer application for the Semantic Web. We argue that wide scale interoperability and integration is a great killer application and invite researchers, developers, and industries to take on that challenge. Tim Berners-Lee [2] advocates publishing and processing everything in RDF using ontologies. We thus believe the timing is just perfect for wide scale interoperability to be tackled, riding this ontology wave!

References

[1] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer, 2003. To appear.

[2] T. Berners-Lee: Semantic Web: Where to direct our energy. Invited talk. International Semantic Web Conference ISWC'03. `http://iswc2003.semanticweb.org/invitedtalks.html`.

[3] T. Berners-Lee and E. Miller. The semantic web lifts off. *ERCIM News*, 51:9–11, 2002.

[4] D. Calvanese, G. D. Giacomo, and M. Lenzerini. A framework for ontology integration. In *Proceedings of Semantic Web Working Symposium*, 2001.

[5] The DARPA agent markup language (DAML). `http://www.daml.org/`.

[6] S. Davidson, W. Fan, C. Hara, and J. Qin. Propagating XML constraints to relations. In *Proceedings of IEEE International Conference on Data Engineering*, 2003.

[7] M. Denny. Ontology building: A survey of editing tools. `http://www.xml.com/pub/a/2002/11/06/ontologies.html`, 2002.

[8] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the International WWW Conference*, pages 662–673, 2002.

[9] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhaven, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Proceedings of Conference on Innovative Data Systems Research*, 2003.

[10] L. V. S. Lakshmanan. Is the Semantic Web a Database? Keynote Talk, *Int. Database Engineering and Applications Symposium*, July 2003, Hong Kong.

[11] L. V. S. Lakshmanan and F. Sadri. Interoperability on XML data. In *Proceedings of the International Semantic Web Conference*, 2003.

[12] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of International Conference on Very Large Databases*, pages 251–262, 1996.

[13] N. F. Noy and M. A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[14] OWL web ontology language. `http://www.w3c.org/TR/owl-features`.

[15] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *Proceedings of International Conference on Very Large Databases*, 2002.

[16] Resource Description Framework (RDF). `http://www.w3c.org/RDF/`.

[17] RDF vocabulary description language (RDF Schema). `http://www.w3c.org/TR/RDF-schema/`.

[18] Semantic Web. `http://www.w3c.org/2001/sw/`.

[19] Amit Sheth and James Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. ACM Computing Surveys, 22(3):183–236, September 1990.

[20] The TSIMMIS project home page. `http://www-db.stanford.edu/tsimmis/tsimmis.html`.

[21] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *Proceedings of VLDB Workshop on Semantic Web and Databases*, pages 131–150, 2003.

Knowledge Provenance Infrastructure

Paulo Pinheiro da Silva Deborah L. McGuinness Rob McCool
Knowledge Systems Laboratory,
Stanford University, Stanford CA 94305
{pp,dlm,robm}@ksl.stanford.edu

Abstract

The web lacks support for explaining information provenance. When web applications return answers, many users do not know what information sources were used, when they were updated, how reliable the source was, or what information was looked up versus derived. Support for information provenance is expected to be a harder problem in the Semantic Web where more answers result from some manipulation of information (instead of simple retrieval of information). Manipulation includes, among other things, retrieving, matching, aggregating, filtering, and deriving information possibly from multiple sources. This article defines a broad notion of information provenance called knowledge provenance that includes proof-like information on how a question answering system arrived at its answer(s). The article also describes an approach for a knowledge provenance infrastructure supporting the extraction, maintenance and usage of knowledge provenance related to answers of web applications and services.

1 Introduction

People who have become effective at using information obtained from the web have become proficient at investigating and evaluating sources of information. If a person believes that, for example, the CNN television station or the New York Times newspaper are reliable sources, then she may be willing to believe the information those organizations publish on the web. If the reputation of the information source is unknown, the person may want more information about the source that may reveal biases, agendas, affiliations, etc. before believing the information. If the information about the source is unavailable or the information source itself is unknown, the person may have a reason to disbelieve the data.

When the user of information is an agent, the task of source investigation and evaluation can not rely on common-sense knowledge such as the reputations of CNN or the New York Times. The agent must have access to the source of information *and* it must have some information about the source in order to be able to evaluate its credibility as a publisher of web information.

If users are going to trust answers obtained from the Semantic Web, then users (humans and agents) need access to knowledge provenance. In order for the Semantic Web to provide knowledge provenance, it needs underlying standards and tools for storing, maintaining and using knowledge provenance. Moreover, this infrastructure must be comprehensive enough to allow the tracking of knowledge provenance from answers to their sources. Therefore, such infrastructure should deal with at least the following kinds of systems:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- Information extraction tools used for building knowledge sources such as ontologies, databases, knowledge bases, taxonomies and thesauri. These tools should be able to register meta information about the sources of assertions in knowledge sources.
- Search engines used for producing answers from web documents. These tools should be able to register and present meta information about retrieved documents.
- Inference engines used for deriving answers from knowledge sources. These tools should be able to register meta information about the knowledge sources used in the process of deriving answers. Also, they should be able to dump their proof traces in a sharable, portable format that can be used to explain how answers are derived.
- Web applications and services using answers derived by inference engines and documents retrieved by search engines. These systems should be able to present provenance information in response to user requests. These systems need to understand knowledge provenance registered by other systems.

This article describes a knowledge provenance infrastructure that facilitates the integration of web applications requiring provenance information. Systems can use the infrastructure for keeping knowledge provenance during the process of extracting knowledge and building knowledge sources. Also, knowledge provenance is considered at a level of granularity appropriate for assertions within knowledge sources. Our previous work on knowledge provenance [8, 9] describes a more conservative approach where knowledge provenance is aimed at the level of granularity appropriate for knowledge sources as a whole.

Moreover, the infrastructure supports Semantic Web functionalities beyond the identification of sources of answers. Our infrastructure provides a specification of a portable proof that can be used to capture information manipulation descriptions. This information is then used by the Inference Web browser to display interactive proof displays for debugging and abstractions of the proofs into more understandable explanations for end users. This information may also be used by truth maintenance systems as well as hybrid reasoning environments.

2 Knowledge Provenance

Knowledge provenance includes *source meta-information*, which is a description of the origin of a piece of knowledge, and *knowledge process information*, which is a description of the reasoning process used to generate the answer. We have used the phrase knowledge provenance instead of data provenance intentionally. Data provenance [1, 3] may be viewed as the analog to knowledge provenance aimed at the database community. That community's definition typically includes both a description of the origin of the information and the process by which it arrived in the database. Knowledge provenance is essentially the same except that it includes proof-like information about the process by which knowledge arrives in the knowledge base. This process may include extensive reasoning used to generate deductive closure information. In this sense, knowledge provenance broadens the notion of data derivation that can be performed before data is inserted into a database or after data is retrieved from a database. Nevertheless, data provenance and knowledge provenance have the same concerns and motivations.

The use of reasoning is not a requirement for using a knowledge provenance infrastructure. For instance, many components of the Inference Web [8] such as the IWBase (a registry containing information about objects useful for proofs and explanations) and portable proofs (a proof interlingua) are used in the ARDA Aquaint project[1], which has a main thrust of question answering using information retrieval techniques. Also, the infrastructure can be used to provide simple source justification for answers that are simply retrieved or for answers

[1] http://www.ic-arda.org/InfoExploit/aquaint/

that have been obtained using complex reasoning and, more typically, it can be used when the answers are derived using a combination of both. A typical scenario includes using knowledge sources where information is available in a format appropriate for machine processing e.g., RDF [6], DAML+OIL [2], OWL [10], etc. If a knowledge base was built using a particular source, for example CNN, then Inference Web would store CNN as the original source of the knowledge. Additional information may be stored about knowledge sources such as the source's authoritativeness, URL, contributors, date of input and update, etc. If some of the information in a knowledge base is from another source, for example the AP news wire, then Inference Web may be used to store that certain assertions came from another source. This information may be attributed at the knowledge base level or at the assertion level.

3 The Stanford Knowledge Provenance Infrastructure

The Stanford Knowledge Provenance Infrastructure (KPI) is an integration approach for TAP [4] with Inference Web's IWBase [9] and Inference Web's portable proofs [12]. TAP is a tool for extracting information and building knowledge sources. It can store provenance information at the level of assertions. IWBase provides infrastructure for provenance originally aimed at the granularity of knowledge bases. In KPI, IWBase will support provenance at the assertion level. Portable proofs support explanations of inferred information along with provenance-based explanations of retrieved answers. These systems are in use for funded projects and are supporting different levels of knowledge provenance. Currently, TAP is not integrated with IWBase and portable proofs. This paper describes the integration route underway and shows how putting the two systems together provides much more than TAP or the Inference Web can do alone. The integrated system provides a scalable solution to knowledge provenance that is aimed at the needs of knowledge bases generated as the result of automated programs (e.g., wrapper and extraction software, e.g., Fetch [5]) as well as knowledge bases generated by humans using tools such as OWL ontology editors (e.g., Protégé [11]).

3.1 IWBase: Infrastructure for Meta-Information Annotation

IWBase [9] (formerly known as the IW Registry) is an interconnection of distributed repositories of meta-information relevant to proofs and explanations, including knowledge provenance information. Every entry in these repositories is an instance of an IWBase concept. For example, *Knowledge Source* is an IWBase concept that is useful for entries such as ontologies, knowledge bases, thesauri, etc. The knowledge source entry for an ontology describes stores of assertions about the ontology such as its original creator(s), date of creation, data of last update, version, URL (for browsing), description in English, etc. IWBase's provenance information is expanding on an as-needed basis driven by application demands.

Every entry has an URI and is stored both as a file written in DAML+OIL/OWL and as a set of tuples in a database. IWBase files are mainly used by portable proofs (see Section 3.3) to annotate their content. IWBase databases are mainly used for evolving meta-information and for supporting web services querying the IWBase.

IWBase can keep provenance at the level of documents. For example, consider the case where the data came from "Joe's Tom Hanks Fan Information Collection", and Joe does not make information available about the source of each piece of data. In KPI, IWBase will also be able to keep provenance at the assertion level. For instance, suppose we have an entry about "Tom Hanks", who is an "Actor". This entry may come from a RDF document where one triple says that there is an entity with a `rdfs:label` of "Tom Hanks", and another triple says that his `rdf:type` is Actor. Some IWBase users, however, simply do not want to keep provenance information at the triple level. In this case, IWBase can store the fact that all of the elements in that particular file or knowledge base, i.e., triples, came from Joe. Thus, when users ask where the particular elements come from, an application using the IWBase can dynamically attach the provenance to the elements and return it. The inclusion of provenance is possible whether provenance is kept at the level of documents, assertions or both

documents and assertions.

3.2 TAP: Infrastructure for Knowledge Source Construction

TAP is a system for knitting together data fragments from disparate XML/SOAP based web services, and from disparate HTML based web sites, into a single schematically unified global knowledge base. The TAP system consists of a number of data servers which communicate with each other and with applications using an XML/SOAP based protocol we call *GetData*. In this case, sites wanting to provide data to the Semantic Web about a particular subject must first agree on the schema for describing that subject. New sites may be added to the system without any modifications to the applications that use the data.

The TAP system has two types of knowledge source tracking. The first method is implicit: the TAP server knows it is interacting, for example, with CNN, therefore it knows that any data it receives in this session is from CNN. The second method is explicit: if the data being retrieved was somehow manipulated, then the TAP object identifiers for each entity being discussed can be examined to discover the page from which they originated. A small match code appended to the page URL also indicates the approximate region of the page from which the entity was found. In either case, the TAP system includes information about each site, and a way to query which site a particular source page comes from. Knowledge provenance information extracted by TAP can be added into the IWBase in an automatic way. Thus, users can see the knowledge provenance later when asking for the sources of answers derived from TAP generated knowledge sources.

TAP provides a bootstrapping system which uses HTML parsers to transform web sites intended for humans into Semantic Web sites intended for agents. To date, TAP has scanned and aggregated data from over 110,000 URLs spread across 35 sites into a knowledge base consisting of over 860,000 logical sentences about over 500,000 individuals. Individual types include corporations, nations, politicians, locations, celebrities, movies, music albums, weapons systems, and many others.

3.3 Portable Proofs: Infrastructure for Answer Derivation Representation

The portable proof specification[2] is a DAML+OIL (migrating to OWL) representation of proofs produced by reasoners during the process of deriving answers. There, a *node* in a deduction tree is labeled by one formula and one inference rule used to conclude the labeling formula. Labeling formulas are formula occurrences. Conceptually one can think of a node in a deduction tree as a representation of one step in a deductive information manipulation process. It is the result of a single rule application applied to some previous information deriving a single formula. A *node set* is a set of one or more nodes where all the nodes are labeled by the same formula. Conceptually one can think of a node set as a set of applications of inference rules used to derive the identical formula in a single step. Node sets capture information concerning *all* of the ways one or more question answering systems came to believe a single statement. A node captures *one* way one or more question answering systems came to believe a single statement. Node sets are a critical building block of the Inference Web since they are the key to proof combination and multiple explanations.

Node sets are used to support knowledge provenance since they are used to track how conclusions are derived from antecedents. The premises of an inference step are the formulas labeling node sets associated with the inference step as antecedents. An answer is derived by the last inference step in a proof.

Knowledge source information may be stored at the level of an entire knowledge base or at an individual assertion level. Either way, Inference Web can take any particular answer and trace back through the inference steps used, looking at their antecedents and determining all of the sources used to arrive at an answer. This process allows Inference Web to provide a summary collection of all sources used to obtain an answer and also allows it to provide the sources used for any particular statement. The identification of all sources used is an important strategy to determine whether or not we should trust the data. For example, we may not trust

[2] http://www.ksl.stanford.edu/software/IW/spec/iw.daml.

information coming from "Joe's Tom Hanks Collection" but we may trust information coming from "The Rita Wilson Fan Club". Thus, we will probably be more likely to believe the data if it is associated with both sources rather than just the "Joe's Tom Hanks Collection". Moreover, suppose that we know that "Joe's Tom Hanks Collection" is known for publishing unreliable information. Then we may be inclined to disbelieve the data even if it is also associated with "The Rita Wilson Fan Club".

Portable proofs may also be used to tackle some problems related to knowledge provenance redundancy. In the simple case, if everything in one knowledge base came from one source, a single statement may be used to capture the source of every statement in the knowledge base. If the knowledge base is created as a view or aggregation of the content available from multiple sources, IWBase can be used to store source information at the statement granularity or it can store that the information in this knowledge base used multiple sources and not distinguish which assertions came from which source.

4 Knowledge Provenance Usage

The infrastructure provides support for provenance information whenever it is possible to identify some document or document element to which we can associate provenance information. Also, it provides a systematic way for generating documents that are relevant for the "semantic part" of the web. Three approaches are considered for an application to use provenance information: it incorporates source meta-information into documents; it incorporates knowledge process information into documents; and it interacts with a data server which is performing multi-site aggregation of data and provenance information.

4.1 Incorporating Source Meta-Information

Applications using our infrastructure do not need to store and manipulate data and its corresponding provenance information in any particular format: provenance information is kept separately in the IWBase, and then reassembled upon request. In fact, our approach has been either to avoid transporting provenance data where we can (and use services to access it later), or to simply accept the cost of storing and maintaining provenance information as a necessary one in order to support trustworthiness of data.

When provenance information is needed, it can be added on a per-file basis. Thus, an application can use a KPI service to retrieve provenance information and it can apply its preferred way of incorporating the information including reification, appending new XML elements, or using quads [7]. For example, for RDF, DAML, and OWL files, the application can use the same approach that we use with TAP documents where TAP can ask IWBase for the URI of provenance information of a given piece of information, e.g., a RDF triple, and apply reification.

The identification of a specific piece of information within a document may be a problem for some document formats such as XML. However, we expect that new standards for XML such as XPath will provide a solution for this problem for XML files.

4.2 Incorporating Knowledge Process Information

When an application computes an answer, the Inference Web infrastructure allows it to dump a portable proof format of the computation process. It also allows an interaction mode that would ask the application for a regeneration of the answer with the portable proof support on demand if that interaction style makes more sense. In either case, the agent (or user) through use of Inference Web can peruse the portable proof to find ground facts supporting the derived answer. If the application dumps limited granularity in the proof such as if it used told information (from a particular source) or used told information from a source and then applied complicated reasoning, the end user could at least have access to the sources used and if the application manipulated the information. We encourage granularity in the portable proof dumps that support demand-based explanations

giving access to the deduction path but we do not require it. This allows us to interact with question answering systems that can not or do not want to provide details of their information manipulation path but still can provide access to the source of the original information.

It is important to note that a portable proof is a forest of proof trees rather than a single tree. This structure is required so that Inference Web can support infrastructures where multiple question answering systems contribute pieces of an answer and also can support hybrid reasoning environments where query managers may break up questions into components that different agents will answer. This is also used to support situations where the same answer can be obtained from multiple paths. This forest feature is one potential reason why Inference Web and the knowledge provenance work may be well suited (and potentially better suited than a data provenance approach) to supporting explanation of the Semantic Web.

4.3 Querying Provenance Information

Each node of the IWBase is a repository of DAML/OWL files mirrored in a relational database. This means that documents can refer to IWBase entries as typical DAML/OWL documents without needing to know about details of database management systems. It also means that queries are expected to be performed in an effective way over the database. In fact, the metamodel for storing provenance meta-information (see the class diagrams in http://www.ksl.stanford.edu/software/iw/spec) is a typical database schema using conventional indexing techniques. Thus, queries over the structured database are expected to have a better performance than over a RDF file storing all the triples for provenance information.

TAP can generate the RDF triples from any particular site on demand and pass the provenance information to IWBase. The set of source URLs and sites contributed to any aggregated data block can then be recorded on IWBase. Any receiving application can then query the IWBase for the source(s) of any triple.

5 Conclusions

The Semantic Web will need infrastructure for knowledge provenance if users are going to trust answers produced by Semantic Web applications and services. In this article we described an infrastructure that can provide comprehensive answer-to-source knowledge provenance for the Semantic Web. This solution integrates the Inference Web infrastructure for explaining answers from web applications and the TAP system for extraction and semantic search.

We also described how provenance information supported by the infrastructure could be used on demand in association with web documents. The Wine Agent[3], the DAML Query Service[4], and the OWL Query Service[5] are Semantic Web agents supported by the Inference Web that present knowledge provenance at the granularity of knowledge sources. These agents are based on the Stanford's JTP hybrid reasoner that produces portable proofs. Also, in the context of the CALO project[6], we are creating a new agent that provides answers along with knowledge provenance information supported by KPI to handle a distributed, hybrid question answering system using a number of reasoning systems.

We are currently extending SRI's SNARK theorem prover[7] to produce portable proofs and integrating with ISI's query planner as well as pursuing discussions with designers of other reasoning systems including W3C's CWM[8] and UT's KM[9]. We also presented a solution that provides provenance information at a granularity aimed

[3] http://www.ksl.stanford.edu/people/dlm/webont/wineAgent/
[4] http://www.ksl.stanford.edu/projects/owl-ql/
[5] http://www.ksl.stanford.edu/projects/dql/
[6] http://www.ai.sri.com/project/CALO
[7] http://www.ai.sri.com/~stickel/snark.html
[8] http://www.w3.org/2000/10/swap/doc/cwm.html
[9] http://www.cs.utexas.edu/users/mfkb/km.html

at facts. Our work provides insight into how to obtain, manipulate, and use meta information using the Inference Web and TAP tools to improve trust on the Semantic Web.

Acknowledgment The authors would like to thank Kevin Wilkinson for his many valuable comments during the preparation of this article.

References

[1] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *Proceedings of 8th International Conference on Database Theory*, pages 316–330, January 2001.

[2] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. DAML+OIL (March 2001) Reference Description. Technical Report Note 18, World Wide Web Committee (W3C), December 2001.

[3] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Trans. on Database Systems*, 25(2):179–227, June 2000.

[4] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 700–709, Budapest, Hungary, May 2003. ACM Press.

[5] Fetch Technologies Inc. http://www.fetch.com/products.asp?sub=prod-agentplatform.

[6] Ora Lassila and Ralph Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 22 February 1999.

[7] Robert MacGregor and In-Young Ko. Representing Contextualized Data using Semantic Web Tools. In R. Volz, S. Decker, and I. Cruz, editors, *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, FL, USA, 2003.

[8] Deborah L. McGuinness and Paulo Pinheiro da Silva. Infrastructure for Web Explanations. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, LNCS-2870, pages 113–129, Sanibel, FL, USA, October 2003. Springer.

[9] Deborah L. McGuinness and Paulo Pinheiro da Silva. Registry-Based Support for Information Integration. In *Proceedings of IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pages 117–122, Acapulco, Mexico, August 2003.

[10] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), August 2003. Candidate Recommendation.

[11] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[12] Paulo Pinheiro da Silva and Deborah L. McGuinness. Combinable Proof Fragments for the Web. Technical Report KSL-03-04, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, January 2003.

Supporting Scalable, Persistent Semantic Web Applications

Kevin Wilkinson[1], Craig Sayers[1], Harumi Kuno[1], Dave Reynolds[1], Luping Ding[2]
HP Labs
{*firstName.lastName*}@hp.com[1], lisading@wpi.edu[2]

Abstract

To realize the vision of the Semantic Web, efficient storage and retrieval of large RDF data sets is required. A common technique for persisting RDF data (graphs) is to use a single relational database table, a triple store. But, we believe a single triple store cannot scale for large-scale applications. This paper describes storing and querying persistent RDF graphs in Jena, a Semantic Web programmers' toolkit. Jena augments the triple store with property tables that cluster multiple property values in a single table row. We also describe two tools to assist in designing application-specific RDF storage schema. The first is a synthetic data generator that generates RDF graphs consistent with an underlying ontology. The second mines an RDF graph or an RDF query log for frequently occurring patterns. These patterns can be applied to schema design or caching strategies to improve performance. We also briefly describe Jena inferencing and a new approach to context in RDF which we call snippets.

1 Introduction

This paper addresses the problems of storing and retrieving Resource Description Framework (RDF) graphs, the data model used for the Semantic Web. It describes RDF persistence in Jena, a widely-used, open-source Semantic Web programmers' toolkit [9, 18]. This paper assumes no familiarity with the Semantic Web and this introduction provides an overview of RDF [15]. The next section describes how Jena implements RDF persistence. The following sections outline Jena query processing and our support for developing application-specific RDF storage schema. A final section discusses the higher-level issues of inferencing and context.

The underlying data model of the Semantic Web, RDF, is a labeled, directed graph. This graph describes properties of and relationships between Web resources. An RDF graph is encoded as a set of triples where each triple encodes one arc of the graph. A triple, also referred to as a statement, has the form <*S*,*P*,*O*>, where S, the subject, and O, the object, are nodes of the graph and P, the predicate, labels an arc from S to O. The subject and predicate are always URIs while the object may be either a literal value or a URI. RDF has the notion of an anonymous resource, also called a blank node or bnode. It denotes an existential variable which is used to indicate a resource whose identity (URI) is unknown or not of interest. A bnode can be used to model n-ary relationships and to model containment (is-part-of). For example, to model a compund name, a bnode can be used to represent an (anonymous) name resource with two arcs to literal nodes for the first name and last name.

Just as LISP builds on a simple list abstraction to create higher-level constructs, RDF builds on predicates and the statement abstraction to model higher-level constructs. RDF includes predefined predicates such as *type*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

to state that the subject is in the class referenced by the object. Other predefined predicates model bags, lists, etc. Of particular interest is the notion of *reification*. It gives identity to a statement and so enables one to make statements about a statement. It uses the predefined predicates *subject, predicate, object, type*. For example, to reify the statement <*sky,is,blue*> so that we can refer to it from other statements, we first need a URI or bnode to identify the statement, say *X*. We then encode the reified statement with four separate statements as follows: <*X,subject,sky*>, <*X,predicate,is*>, <*X,object,blue*>, <*X,type,Statement*>. Given this, to express "Alice said the sky is blue", we add the statement <*Alice,said,X*>. Reification enables an RDF graph to include seemingly contradictory assertions, e.g., by reifiying <*sky,is,green*>.

RDFS (RDF Schema [7]) builds on RDF by adding more predefined predicates to increase the modeling power of the graph. RDFS includes a *subclass* predicate to support class hierarchies and a *sub-property* predicate. OWL (Ontology Web Language [19]) adds another modeling layer with predicates for ontologies and reasoning, e.g., to describe transitive properties and functional predicates. It is largely derived from DAML+OIL [10] and is based on Description Logics. However, note that OWL, RDFS and RDF graphs all have the same representation, i.e., a set of triples (statements).

There are a number of toolkits that implement the RDF model. Although there is no standard API such as ODBC, most RDF systems support methods to add a statement to a graph, remove a statement from a graph and to query graphs. In addition, they usually support some graph operations such as graph creation, graph merging, graph serialization, etc. However, the RDF specifications have no explicit notion of graph. So issues like graph identity, querying across graphs, distributing graphs are handled differently (or not at all) across the toolkits.

There is currently no standard query language for RDF. We describe here the Jena query capabilities which are typical of many RDF toolkits. Jena supports two forms of querying: triple match and RDQL [16]. A *triple match* takes a match template of the form *(s,p,o)*, where each term is either a constant or a don't-care. It returns all statements that match the template. For complex queries Jena supports RDQL, a declarative RDF query language that has been widely adopted. An RDQL query may be expressed as a conjunction of triple match templates in which a match term may also be a variable. The variables enable joins across match templates. Additional constraint clauses can be used to limit the value range of variable bindings using arithmetic and string processing functions. The query returns all bindings of variables that satisfy the triple matches.

2 Persistent RDF Storage in Jena

A common approach to making RDF graphs persistent is to use a relational database and store the graph in a three column table, one column each for the subject, predicate and object of a statement. Many RDF storage systems use this *triple store* approach [4, 8, 14]. Typically, a normalized schema is used in the triple store approach which adds two additional tables. A literals table is used to store all literal values and a resources table is used to store all URI and bnodes. The columns of the triple store then reference values in the literals and resources tables. The normalized triple store approach is very efficient in space since a literal or resource is stored only once. However, retrieving a statement requires a three way join.

There are challenging problems in using relational databases for storing RDF. For example, the character lengths of the literals and URIs in RDF graphs may vary widely and may even be blobs. The object of an RDF statement is not strongly typed and may either be a literal or a URI. An additional (boolean) column is sometimes used to indicate which it is. An RDF literal may optionally have an XSD datatype [5] or even a user-defined type. Consequently, there is no database column type that is suitable for all literal values. Typically, implementations simply store the value as a character string. Some systems optionally store the literal redundantly in a separate column in a native format, e.g., integer. This is usually only done for a subset of the XSD types.

Jena Denormalized Schema. The initial Jena implementation used a normalized triple store [20]. It also supported drivers for a number of different database engines, including MySQL, PostgreSQL, Oracle and Berkeley

DB (BDB). The BDB driver actually used a different, denormalized, schema and it was observed that the BDB implementation was significantly faster than the relational engines. Consequently, for the second generation Jena, we implemented a denormalized triple store for the relational engines [22]. In this schema, *short* literal values and *short* resource URIs are stored directly in the triple store table. Only when the length of a literal or URI exceeds a threshold is it stored separately in the literals or resources table. To tag its location, a literal value is stored with a prefix to indicate if it is in-line or in the literals table. Similarly, for URIs.

The denormalized approach uses more space than the normalized approach. However it saves time in that short values in statements can be retrieved without a join. And, the length threshold for short values is configurable so that applications may control the degree of denormalization, trading off space for time. Preliminary performance results for simple retrievals show the denormalized approach to be twice as fast as the normalized approach but approximately twice the space [22].

Property Tables. In the absence knowledge about the graph being stored, there are few options for storage optimization and a triple store approach is reasonable. But, as mentioned above, RDF encodes higher-level constructs such as lists. And, real-world data often have repeating patterns or structures. A simple triple store cannot leverage knowledge of patterns. For scalability and high-performance, we believe the triple store must be augmented by other storage strategies better tuned for the graph being stored or the applications using it.

To leverage patterns in an RDF graph, in Jena we have started to experiment with *property tables* to augment the triple store. A property table clusters multiple property values for the same subject in one table row. It uses one column for the subject and one column for each predicate value. In general, a property table stores all statements for its set of predicates. For example, a Person property table might store all statements about names, addresses and phones. There would be no statements with those predicates in the triple store.

A property table saves space over a triple store in that a property table does not explicitly store the predicate URIs. Instead the column (name) identifies the predicates. Also, for all statements in the row of a property table, the subject is only stored once. A property table can also save time over a triple store. When there is access locality to its statements, the entire row of a property table can be cached to avoid repeated calls to the database engine. Note that, unlike the triple store, the columns of a property table can be null. So the sparsity of property values is a factor in choosing which predicates to cluster in a property table. In general, the choice of the property tables is application-specific. Later, we describe tools to help users choose property tables.

In the current implementation, Jena supports one type of property table. It is used to store *reified* statements. Reification is an important special case because some RDF applications reify every statement. It would be inefficient to store each reified statement as four separate triples. For reification, Jena uses a property table of five columns, one for the statement identifier and one each for the subject, predicate, object and type values. Some preliminary performance results show that simple retrievals from the reification property table to be four times faster than retrieving the same statements from a triple store [22]. In some cases, the speed-up is as high as ten times faster which we believe is due to caching effects since the property table makes more effective use of the cache (e.g., the predicate URI is not repeatedly stored).

In the future, Jena will support arbitrary property tables for applications. One issue is that property tables complicate query processing, as described in the next section. However, once general property tables are supported, it should be an easy next step to directly connect Jena to arbitrary (legacy) relational databases to extract that data as RDF statements, e.g., using techniques similar to those in [6].

3 RDF Query Processing in Jena

There are two forms of Jena querying: triple match and RDQL querying. A triple match returns all statements that match a template of the form *(s,p,o)* where each element is either a constant or a don't-care. An RDQL query [16] can be expressed as a conjunction of triple match templates, where each term is either a constant, a

don't-care or a variable. Variables enable joins across the triple matches. A query returns all valid bindings of its variables over statements in the graph.

The addition of property tables significantly complicates query processing. Consider iterators. Unlike a triple store table where each row corresponds to a single RDF statement, an iterator over a property table must expand a row into multiple statements, one for each predicate value. However, the major complexity occurs when a template includes an unspecified property, i.e., where the predicate is a don't-care or a variable that will be bound when the query is processed. For example, a common template is to return all known property values for a specified subject, e.g., *("Alice",-,-)*.

For a triple match where the predicate is unknown, in addition to the scanning the triple store, all property tables must be scanned and the results concatenated. In principle, it is possible to generate a single SQL union query for this match but such queries can quickly become unwieldy so this optimization has not been pursued.

In Jena, an RDQL query can be processed in a naive fashion as a nested-loops pipeline of triple match templates. The variable bindings of one triple match are used to as input to the next triple match which then binds variables for subsequent triple matches until all matches are processed. However, it would be more efficient if joins could be pushed into the database engine for evaluation. This is a difficult problem in the presence of property tables and unspecified predicates. However, in the case that all triple match templates reference only the triple store, it is possible to generate a single SQL query to resolve all match templates. Similarly, if all match templates reference only property tables, it is also possible to generate a single SQL query.

However, when the match templates span both the triple store and property tables or the predicates are unspecified (variable or don't-care), then it is difficult to generate a single SQL query. The current approach in Jena is to partition the match templates for a query into groups, where each group contains matches that can be completely evaluated by the same table and that are connected by join variables. There is one additional group containing templates that span tables. SQL queries are then generated for the single table groups and the final group is processed using the (naive) nested-loops approach. More efficient algorithms remain future work.

4 Developing Application-Specific Schema for RDF

By application-specific schema, we mean an RDF storage schema that augments the triple store with additional tables tailored for the application. Application-specific schema for storing RDF are supported in other RDF stores [4, 8, 14]. However, those systems use the RDF Schema class hierarchies to derive the application-specific database schema. In Jena, we want to support arbitrary property tables independent of the class hierarchies.

To faciliate developing an application-specific schema, we have created two tools [11]. The first is a data mining tool that discovers patterns within RDF graphs. The second is a synthetic data generator that generates realistic RDF graphs according to user-supplied specifications. These synthetic data sets can be used to populate and benchmark candidate application-specific schema. This section briefly describes these tools (see also [11]).

The RDF mining tool uses *Market-Basket Analysis* [2] techniques to discover patterns in RDF graphs and RDF query logs. If an application has an initial or sample data set (an RDF graph), our mining tool can be applied to this graph to discover property co-occurrence patterns to suggest possible properties to store together. For example, if the tool found that the predicates *Name, Address, Phone* were a frequent pattern for subjects, they would be a candidate property table.

The mining tool finds *frequent item sets* [3] in a collection of transactions of items purchased together (a market basket). Applying this to a set of RDF statements, we use the statement subject as the transaction identifier and its item set is the set of all predicates that appear with that subject in some statement. The mining algorithm then finds the longest frequently occurring item sets (for a common subject) that exceed a support threshold, i.e., a percentage of all statements.

In addition to statement patterns, the set of queries applied to a graph may also exhibit patterns and these patterns should be considered when devising a storage schema. Our mining tool can find several types of patterns

in a log of RDQL queries. By varying the definition of transaction identifier and item set, our mining tool can uncover both frequently occurring queries (e.g., common join queries) or frequently queried predicates for a common subject. By using a sequence mining algorithm, it can also find commonly occurring query sequences.

Our data generator tool quickly produces large volumes of synthetic RDF data. The user-provided specifications allow a high degree of control over the characteristics of the generated data. Unlike many other synthetic data generators, it is capable of modeling relationships among class instances. Currently, the tool generates some number of class instances where each class instance comprises values for predicates of that class. Class definitions may come from an existing ontology or from a user specification. The number of occurrences of a predicate (its cardinality) in an instance may be fixed or varied. The values may be literals or resources and may randomly generated or selected from a fixed population.

Distribution functions may be specified for both the predicate cardinalities and the values themselves. The distributions supported include constant, uniform, Normal, Poisson and Zipf. The tool can also generate composite values (e.g. a name comprised of first and last names) by using bnodes and the RDF container structures *Bag, Seq* and *Alt*. Resource values may reference other generated class instances or external resources. Properties of a class may also be self-referencing to model hierarchical relationships such as taxonomies, ancestors, sub-parts. More complicated chains of references such as the Markov chains in [1] remain future work.

5 Support for Inferencing and Contexts

Inferencing. In addition to a basic graph API, Jena supports an ontology API that can generate entailments, i.e., statements that are inferred by the class and property relationships but that are not explicitly present in a base graph. For example, given the two statements <*employee,subclass,person*>, <*"Alice",type,employee*>, then, in a list of all persons, the RDFS entailments would include the statement <*"Alice",type,person*> even though it is not explicitly stated.

In Jena, inferencing is implemented through inference graphs. An inference graph implements the standard graph interface but also supports additional methods for consistency checking, access to deduction traces and query with posits. Inference graphs are layered on top of a base graph implementation, i.e., in-memory graphs or persistent graphs. The inference graph API can be used to access a range of different inference engines. Jena ships with a pair of generic rule engines - a forward chaining engine based on the standard RETE algorithm and backward chaining engine which uses SLD style backtracking with tabling. These engines can run in a hybrid mode (forward-backward chaining). Jena includes built-in rule sets which enable these rule engines to provide RDFS and a useful subset of OWL inferences. The architecture is extensible to allow external rule engines to be used and there has been some early experimental work on connecting to the Racer[12] description logic engine.

The current rule engines will work over persistent graphs but all inferencing is done in memory, outside the database engine. There is clearly scope for further optimization that might leverage support in the database engine for transitive closure or recursion. However, at the time of writing, there has been no specific investigation into direct deductive database implementations in Jena. Some RDF systems support inferencing directly in the storage engine by precomputing and storing some or all of the entailments [8, 13]. The hybrid rule engine approach of Jena lends itself to this - the partial closure generated by the forward rules could be persisted in the database with the backward rules being used to rewrite queries dynamically.

Contexts. Some Semantic Web applications require the ability to represent the context of a statement. For example, one might want to represent that a set of statements all refer to a particular place or occurred at a certain time or were uttered by a given speaker. We expect that a collection of statements made about a particular subject and in a particular context will be a commonly-occuring pattern. We have invented a name for that case, we call it a *Snippet* [21]. Each snippet consists of a set of statements $\{<S,P_i,O_i>\}$, $0<i<N$, made about a subject S in a context C. A snippet may be stored in a standard form as a bag of reified statements as described in [21].

Many RDF stores optimize the storage of reified statements internally by storing the four triples as a single *quad*, i.e., a statement with a fourth element. Some stores also use a quad representation for context, with a contextual node as the fourth element. While quads are efficient, they are non-standard and so may limit interoperability. For snippets, we created a compact representation with the form:

<*X, type, Snippet*> <*X, about, S*> <*X, context, C*> <*X,* P_1*,* O_1> ... <*X,* P_N*,* O_N>

Others have termed a similar construct a *Snapshot* [17]. Since this compact representation uses triples, it may be stored, transported and queried in the compact form using any existing RDF triple store. The application and reasoning layers need to know that a compact snippet is an abreviation but the underlying storage, transport, and querying layers do not.

Snippets neatly encapsulate a unit of knowledge about some (subject) resource within some context. Consequently, it is a natural unit of distribution and sharing. Snippets may be directly accessed by their identifier (*X* above) or associately accessed by their subject and context (*S* and *C* above). In addition, we believe a common case is that a unit of knowledge is immutable, or perhaps versioned. For this case, we have developed techniques to efficiently generate snippet identifiers using a content-based hash [21].

The use of content-based identifiers is particularly convenient for distributed environments. The identifier is short-hand for the content so it is easy to check that two snippets have the same content. Snippets can easily be cached and replicated with no risk of data inconsistency. Finally, we believe that snippets can be stored efficiently using a property table, allowing a set of N statements to reside in a single table row. Of course, this assumes the set of properties in the snippet is relatively static but this may be a common case.

6 Conclusions

An advantage of RDF is that it provides a flexible and extensible model for representing associations. Because the model does not have strong type requirements, this flexible representation raises challenges as to how to store and query RDF data efficiently. By combining the standard triple store with arbitrary property tables, we gain the advantage of being able to leverage a structured schema while keeping the flexibility of a generic triple store. We also described snippets, a new construct for the efficient expression and storage of the context associated with statements. These techniques provides efficient support for both reified and non-reified data.

References

[1] A. Aboulnaga, J. Naughton, and C. Zhang. Generating synthetic complex structured XML data. In *Proc. 4th Int. Workshop on the Web and Databases (WebDB'2001)*, May 2001.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*, Washington, D.C., 1993.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *11th Intl Conf on Data Engineering*, Taipei, Taiwan, 1995.

[4] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In *2nd Intl Workshop on the Semantic Web*, May 2001.

[5] P. Biron and A. Malhotra. XML schema part 2: Datatypes. http://www.w3.org/TR/xmlschema-2, May 2001.

[6] C. Bizer. D2R map - a database to RDF mapping language. In *12th Intl World Wide Web Conf*, Budapest, May 2003.

[7] D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF schema. http://www.w3.org/TR/rdf-schema, October 2003.

[8] J. Broekstra and A. Kampman. Sesame: A generic architecture for storing and querying RDF and RDF schema, October 2001. http://sesame.aidministrator.nl/publications/del10.pdf.

[9] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. The Jena Semantic Web Platform: Architecture and design. Technical report, Hewlett Packard Laboratories, 2003.

[10] D. Connolly, F. van Harmelen, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. A. Stein. Daml+oil (march 2001) reference description. http://www.w3.org/TR/daml+oil-reference, December 2001.

[11] L. Ding, K. Wilkinson, C. Sayers, and H. Kuno. Application-specific schema design for storing large RDF datasets. In *First Intl Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, Florida, October 2003.

[12] V. Haarsley and R. Moller. Description of the racer system and its applications. In *Intl Workshop on Description Logics*, Stanford, California, August 2001.

[13] S. Harris and N. Gibbins. 3store: Efficient bulf rdf storage. In *First Intl Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, Florida, October 2003.

[14] KAON - the Karlsruhe Ontology and Semantic Web Tool Suite. http://kaon.semanticweb.org/.

[15] G. Klyne and J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts, November 2002.

[16] A. R. L. Miller, A. Seaborne. Three implementations of SquishQL, a simple RDF query language. Technical Report HPL-2002-110, Hewlett Packard Laboratories, Palo Alto, California, 2002.

[17] B. MacGregor and I.-Y. Ko. Representing contextualized data using semantic web tools. In *First Intl Workshop on Practical and Scalable Semantic Systems*, Sanibel Island, Florida, October 2003.

[18] B. McBride. Jena: Implementing the RDF model and syntax specification. In S. Decker et al., editors, *Second International Workshop on the Semantic Web*, Hong Kong, May 2001.

[19] D. McGuinness and F. van Harmelen. Owl web ontology language overview. http://www.w3.org/TR/owl-features, August 2003.

[20] D. Reynolds. Jena Relational Database interface - performance notes (in Jena 1.6.1), 2003. http://www.hpl.hp.com/semweb/download.htm/.

[21] C. Sayers and K. Wilkinson. A pragmatic approach to storing and distributing rdf context using snippets. Technical Report HPL-2003-231, Hewlett Packard Laboratories, Palo Alto, California, 2003.

[22] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *First Intl Workshop on Semantic Web and Databases (SWDB'03, with VLDB03)*, Berlin, September 2003.

Semantic (Web) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis

Amit Sheth
Semagix and LSDIS Lab,
University of Georgia,
Athens, GA
amit@cs.uga.edu

Cartic Ramakrishnan
LSDIS Lab,
University of Georgia
Athens, GA
cartic@cs.uga.edu

Abstract

Semantics is seen as the key ingredient in the next phase of the Web infrastructure as well as the next generation of information systems applications. In this context, we review some reservations expressed about the viability of the Semantic Web. We respond to these by identifying a Semantic Technology that supports the key capabilities also needed to realize the Semantic Web vision, namely representing, acquiring and utilizing knowledge. Given that scalability is a key challenge, we briefly review our observations from developing three classes of real world applications and corresponding technology components: search/browsing, integration and analytics. We distinguish these proven technologies from some parts of the Semantic Web approach and offer subjective remarks, which we hope will foster further debate.

1 Introduction

Semantics is arguably the single most important ingredient in propelling the Web to its next phase, and is closely supported by Web services and Web processes that provide standards based interoperability of applications. Semantics is considered to be the best framework to deal with the heterogeneity, massive scale and dynamic nature of resources on the Web. Issues pertaining to semantics have been addressed in other fields like linguistics, knowledge representation and AI. The promise of semantics and challenges in developing semantic techniques are not new to researchers in the database and information system field either. For instance, semantics has been studied or applied in the context of data modeling, query and transaction processing etc. Recently, a group of both database and non-database researchers came together at the *Amicalola State Park* for an intensive look at the relationship between database research and the Semantic Web. During this collaboration, they identified three pages worth of opportunities to further database research while addressing the challenges in realizing the Semantic Web [16]. A follow on workshop also presented opportunity to present research at the intersection of database and the Semantic Web [`http://swdb.semanticweb.org`]. Nevertheless, many researchers in the database community continue to express significant reservations toward the Semantic Web. Table 1 shows some examples of criticisms or skeptical remarks about Semantic Web technology (taken from actual NSF proposal reviews and conference panel remarks).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

"As a constituent technology, ontology work of this sort is defensible. As the basis for programmatic research and implementation, it is a speculative and immature technology of uncertain promise."
"Users will be able to use programs that can understand semantics of the data to help them answer complex questions .... This sort of hyperbole is characteristic of much of the genre of semantic web conjectures, papers, and proposals thus far. It is reminiscent of the AI hype of a decade ago and practical systems based on these ideas are no more in evidence now than they were then."
"Such research is fashionable at the moment, due in part to support from defense agencies, in part because the Web offers the first distributed environment that makes even the dream seem tractable."
"It (proposed research in Semantic Web) pre-supposes the availability of semantic information extracted from the base documents–an unsolved problem of many years, ..."
"Google has shown that huge improvements in search technology can be made without understanding semantics. Perhaps after a certain point, semantics are needed for further improvements, but a better argument is needed."

Table 1: Some Reservation among DB researchers about the Semantic Web

These reservations and skepticisms likely stem from a variety of reasons. First, this may be a product of the lofty goals of the Semantic Web as depicted in [3]. Specifically, database researchers may have reservations stemming from the overwhelming role of description logics in the W3C's Semantic Web Activity and related standards. The vision of the Semantic Web proposed in several articles may seem, to many readers, like a proposed solution to the long standing AI problems. Lastly, some skepticism stems from legitimate concern about the scalability of the three requisite core capabilities of the Semantic Web: (a) ontology creation and maintenance of large ontologies, (b) semantic annotation, and (c) inference mechanisms or other computing approaches involving large, realistic ontologies, metadata and heterogeneous data sets. Despite these reservations, some of them well justified, we believe semantic technology is beginning to mature and will play a significant role in the development of future information systems. We believe that database research will greatly benefit by playing critical roles in the development of both Semantic Technology and the Semantic Web. In addition, we also feel that the database community is very well equipped to play their part in realizing this vision. We aim here to:

- Identify some prevalent myths about the Semantic Web
- Identify instances of Semantic (Web) Technology in action and how the database community can make invaluable contributions to the same.

For the purpose of this article, as well as for tagging real and existing versus more futuristic and speculative alternatives, we distinguish between Semantic Technology and Semantic Web technology. By Semantic Technology [11] (a term that predates the "Semantic Web"), we imply application of techniques that support and exploit semantics of information (as opposed to syntax and structure/schematic issues [Sheth 99][1]) to enhance existing information systems. In contrast, the Semantic Web technology (more specifically its vision) is best defined as *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."* [3]. Currently in more practical terms, Semantic Web technology also implies the use of standards such as RDF/RDFS, and for some, OWL. It is however important to note that while description logic is a center piece for many Semantic Web researchers, it is not a necessary component for *all* Semantic Web applications. For the Semantic Technology as the term is used here, complex query processing, involving both metadata and ontology takes center stage, and is where database technology continues to play a critical role. For our purpose, semi-formal ontologies are those that do not claim formal semantics and/or are populated with partial or incomplete knowledge. For example, in such an ontology, all schema level constraints may not be observed in the knowledgebase that is instantiated *wrt* the ontology schema. This becomes especially relevant when ontology is populated by many persons or by extracting and integrating knowledge from multiple sources.

[1]For a commercial use of term "Semantic Technology" see [11].

2 Examples of Semantic (Web) Technology Applications and Observations

We summarize a few applications developed using a commercial technology to provide insights into what Semantic (Web) Technology can do today. Based on the increasing complexity and the deeper role of semantics, we divide the applications into three types[2]:

Semantic search and contextual browsing: In *Taalee* (now *Semagix*) Semantic Search Engine [18], the ontology consisted of general interest areas with five major categories and over 16 subcategories such as News, Entertainment and Sports. Blended Semantic Browsing and Querying (BSBQ) provided domain specific search (search based on these domain specific attributes) and contextual browsing. The application involved crawling/extracting audio, video and text content from well over 200 sources (e.g. CNN website). This application was commercially deployed for a Web-audio company called *Voquette*. An interesting related application not developed by Semagix is reported in [7].

Semantic integration: In *Equity Analyst's Workbench* [15] audio, video and text content from tens of sites and NewML feeds aggregated from 90+ international sources (such as news agencies of various countries) were constantly classified into a small taxonomy and domain specific metadata was automatically extracted. The equity market ontology used by this application consists of over one million facts (entity and relationship instances). An illustrative example of a complex semantic query involving metadata and ontology this application supported is: Show analyst reports (from many sources in various formats) that are competitors of Intel Corporation. In an application involving *Repertoire Management* for a multinational Entertainment conglomerate, its ontology with relatively simple schema is used to extract over 14.5 million instances. The application provided integrated access to heterogeneous content in the companys extensive media holding while addressing semantic heterogeneity (e.g., in naming of artist, tracks, etc.)

Analytics and Knowledge Discovery: In *Passenger Threat Assessment* application for homeland security [1] and *Semagix's Antimoney Laundering* solution [14], the knowledge base is populated from many public, licensed and proprietary knowledge sources using the CIRAS ontology. The resulting knowledge base has over one million instances. Periodic or continuous metadata extraction from tens of heterogeneous sources (150 files formats, HTML, XML, dynamic Web sites, databases, etc.) is also performed. When the appropriate computing infrastructure is used, the system is scalable to hundreds of sources, or about a million documents per day per server. A somewhat related non-Semagix business intelligence [9] application has demonstrated scalability by extracting metadata (albeit somewhat limited types of metadata with a significantly smaller ontology) from a billion pages [12].

Our experience in building the above real-world applications has led us to some empirical observations:

- Applications validate the importance of ontology in the current semantic approaches. An ontology represent a part of the domain or the real-world for which it represents and captures a shared knowledge around which the semantic application revolves. It is the "ontological commitment" reflecting agreement among the experts defining the ontology and its uses, that is the basis for "semantic normalization" necessary for semantic integration.
- Ontology population is critical. Among the ontologies developed by Semagix or using its technology, median size of ontology is over 1 million facts. This level of capture of knowledge makes the system very powerful. Since it is obvious that this is the sort of scale Semantic Web applications are going to be dealing with, means of populating ontologies with instance data need to be automated.
- Two of the most fundamental "semantic" techniques are named entity recognition and semantic ambiguity resolution (also closely tied to data quality problem). Without good solutions to these none of the applications listed above will be of any practical use. Both require highly multidisciplinary approaches, borrowing for NLP techniques, statistical and IR techniques and possibly machine learning techniques.

[2] At least the applications underlined are known to have been developed by commercial technology/product or deployed.

- Semi-formal ontologies that may be based on limited expressive power are most practical and useful. Formal or semi-formal ontologies represented in very expressive languages (compared to moderately expressive ones) have in practice, yielded little value in real-world applications. One reason for this may be difficulty capturing knowledge that uses the more expressive constructs of a representation language. This difficulty is especially apparent when trying to populate an ontology that uses a very expressive language to model a domain. Hence the additional effort in modeling these constructs for a particular domain is often not justifiable in terms of the gain in performance. Also there is a widely-accepted trade-off between expressive power of such representation languages and computational complexity associated with inference mechanisms for such languages. Practical applications often end up using languages that lie closer to less expressive languages in the "expressiveness vs. computational complexity continuum". This resonates with so-called Hendler's hypothesis ("little semantics goes a long way").
- Large scale metadata extraction and semantic annotation is possible. Storage and manipulation of metadata for millions to hundreds of millions of content items challenges us to apply and improve the performance and scalability of known database techniques in the presence of new, complex, structures.
- Support for heterogeneous data is key - it is too hard to deploy separate products within a single enterprise to deal with structured and unstructured data/content management. New applications involve extensive types of heterogeneity in format, media and access/delivery mechanisms (e.g., news feed in NewsML news, Web posted article in HTML or served up dynamically through database query and XSLT transformation, analyst report in PDF or WORD, subscription service with API-based access to Lexis/Nexis, etc). Database researchers have long studied this issue of integrating heterogeneous data.
- Semantic query processing with the ability to query both ontology and metadata to retrieve heterogeneous content is highly valuable. Consider the query "Give me all articles on the competitors of Intel", where ontology gives information on competitors, supports semantics (with the understanding that "Palm" is a company and that "Palm" and "Palm, Inc." are the same in this case), and metadata identifies the company an article refers to, regardless of format of the article. Analytical applications could require sub-second response time for tens of concurrent complex queries over large metadata base and ontology, and can benefit from further database research. High performance and highly scalable query processing that deal with more complex representations compared to database schemas and with more explicit role of relationships, is important. Database researcher can also contribute to the strategies of dealing with large RDF stores.
- A vast majority of the Semantic (Web) applications that have been developed or envisioned rely on three crucial capabilities namely ontology creation, semantic annotation and querying/inferencing. Enterprise scale application share many requirements in these three respects with pan Web applications. All these capabilities must scale to millions of documents and concepts (rather than hundreds to thousands).

3 Discussion

Ontologies come in bewildering variety; Figure 1 represents just three of the dimensions of variation. To keep a focus on real world applications and for the sake of brevity, we restrict the scope to task specific and domain specific ontologies. As observed recently by Gruber [6], currently the ontologies that are semi-formal have demonstrated very high practical value. We believe that ontology development effort for semi-formal ontologies can be significantly smaller compared to that required for developing formal ontologies or ontologies with more expressive representations. This is especially evident when considering ontology instance population efforts. Semi-formal ontologies have provided good examples of both value and utility. E.g., GO (`http://www.geneontology.org/`), is arguably a nomenclature from the perspective of representation and lacks formal and richer representation necessary to qualify as a formal ontology (discussed in more detail later).

Research in database and information systems have and will continue to play a critical role with respect to the scalability. We review the crucial scalability aspect of the three capabilities next.

3.1 Availability of large and "useful" ontologies

Although an ontology schema may resemble at a representational level a database schema, and instances may reflect database tuples, the fundamental difference is that ontology is supposed to capture some aspect of real-world or domain semantics, as well as represent ontological commitment forming the basis of semantic normalization. Methods for creating ontologies can be grouped into the following types:

- Social processes where a group of users go through a process of suggestions and iteratively revise versions of ontologies to capture domain semantics.
- Automatically (or semi-automatically) extract the ontology schema (i.e., ontology learning) from content of various kinds. Despite a recent spate of interest, this approach relates to the knowledge acquisition bottleneck faced in the AI research of eighties, and we have little practical experience to rely upon [5].
- Automatic or semi-automatic (with human curation) population of the knowledge base with respect to human design ontology schema. We can report on practical experience with a scalable approach of this type since several ontologies with over million instances have been create with a total of 4 to 8 weeks of effort (e.g., knowledge extraction in *SemagixFreedom* [15]).

3.2 Semantic Metadata Extraction or Semantic Annotation of massive content

Annotating heterogeneous content with semantics provided by relevant ontology (or ontologies) is a key challenge for the Semantic Web. Recently there have been commercial results providing detailed semantic annotations of heterogeneous content (structured, semi-structured, and unstructured with different formats) [15, 8], as well as research reporting annotation of over a billion Web pages [12]. As observed in the efforts on automatic semantic annotation, two resources necessary for realizing the semantic web are: (a) large scale availability of domain specific ontologies; and (b) scalable techniques to annotate content with high quality metadata descriptions based on the terms, concepts or relationships provided by these ontologies. We believe main area of challenge here is to support increasing number of practical and scalable techniques for semantic disambiguation.

3.3 Inference Mechanisms that Scale

Inference mechanisms that can deal with the massive number of assertions that would be encountered by Semantic Web applications are required. The claimed power behind many of the proposed applications of Semantic Web technology is the ability to infer knowledge that is not explicitly expressed. Needless to say, this feature has attracted attention from the AI community since they have been dealing with issues relating to inference mechanisms for quite some time. Inference mechanisms are applicable only in the context of formal ontologies. One of the most common knowledge representation languages has been Description Logic [10] on which DAML, one of the earliest Semantic Web languages is based. It was in fact one of the less expressive members of the DL family. The reason for limiting the expressive power of such a knowledge representation formalism was very clear when the decidability and complexity issues were considered. Although several optimized methods of inference were introduced later [2], inference mechanisms were still overshadowed by the performance advantage of traditional database systems. This has lead to reluctance among many database researchers to accept the Semantic Web vision as viable. Description Logics may form a part of some Semantic Web solutions of the future. We are however convinced it is not the only knowledge representation formalism that will go on and make the Semantic Web vision a reality. It is possible to do some sort of rudimentary inference using RDFS (using *subClassOf* and *subPropertyOf*). However, using RDF/RDFS does not force one to use *only* inference mechanisms of some sort in applications. Since RDFS has a graph model associated with it there is the possibility to use other techniques to answer complex queries [1].

3.4 Why semi-formal, less expressive ontologies?

Ontologies serve several purposes, including: having an agreement between humans, having a common representation for knowledge, having machines (software) get common interpretation of something that humans have agreed to, and forming the basis for defining metadata or semantic annotation. Tom Gruber, who many would credit with bringing the term to vogue in contemporary knowledge representation and information systems research, identifies three types of ontologies-informal, semiformal and formal [6]. He stresses the value of semi-formal ontology in meeting several challenges; especially that of information integration. Some researchers in the Semantic Web community would argue for only formal ontologies (and discount the value of semi-formal ontologies). We do not doubt that formal ontologies have a potential role in Semantic Web research. However, database researchers should particularly realize the value of and exploit semi-formal ontologies. Figure 1 stresses that there is a very large body of work that can and needs to be done using semi-formal ontologies.

GO ontology, which is more a nomenclature and taxonomy, than a formal ontology, is highly successful and extensively used. Although GO is technically a nomenclature rather than an ontology [13][3] it has been successfully used to annotate large volumes of data and consequently support interoperability and integration from heterogeneous data sets. This shows that highly expressive formal ontologies are not required for all Semantic Web applications. It also shows that real world applications often can be developed with very little semantics (Hendler's hypothesis: "little semantics goes a long way"). There is a very good reason as to why semi-formal ontologies are both more abundant and more useful than formal ontologies. The answer lies in the ease with which semi-formal ontologies can be built to a scale that is useful in real-world applications.

Our objective in touting the value of semi-formal ontologies is to prevent research in the Semantic web field from leading straight into the very problems that AI found itself in. We hope to do this by reducing the prevalent emphasis on formal ontologies and pure deductive inference mechanisms. The reader should note however that we do not completely discount the value of the same. We liken the current research direction in the Semantic Web community to, attempting to construct a new building using the flawed building blocks that lead to the downfall of previous building attempts. Our reasoning behind this is that most motivating examples described in this field pay little or no attention to the fundamental (read hard) problems of entity/relationship identification and ambiguity resolution. Database researchers working on schema integration are only too familiar with the problems relating to ambiguity resolution. According to [17], most scenarios described for potential applications of the Semantic Web trivialize these fundamentally hard problems while emphasizing the trivial problems. Our views coincide with those expressed in [13, 4]. In [4] the Semantic Web community is urged to not waste their efforts on "fixing the plumbing" (referring to infrastructure issues) and to focus their efforts on the more fundamental problems.

4 Semagix Freedom: An example of state of the art Semantic Technology

Let us briefly describe a state of the art commercial technology and product that is built upon the key perspectives we presented above. *Semagix Freedom* exploits task and domain ontologies that are populated with relevant facts in all key functions: automatic classification of content, ontology-driven metadata extraction, and support for complex query processing involving metadata and ontology for all three types of semantic applications identified in Section 2. It provides tools that enable automation in every step in the content chain - specifically ontology design, content aggregation, knowledge aggregation and creation, metadata extraction, content tagging and querying of content and knowledge. Scalability, supported by a high degree of automation and high performance based on main memory based query processing has been of critical importance in building this commercial technology and product. Figure 2 below shows the architecture of *Semagix Freedom*.

[3] "For data annotation, in principle not a full fledged ontology as described above is required but only a controlled vocabulary since the main purpose is to provide constant and unique reference points." [13]

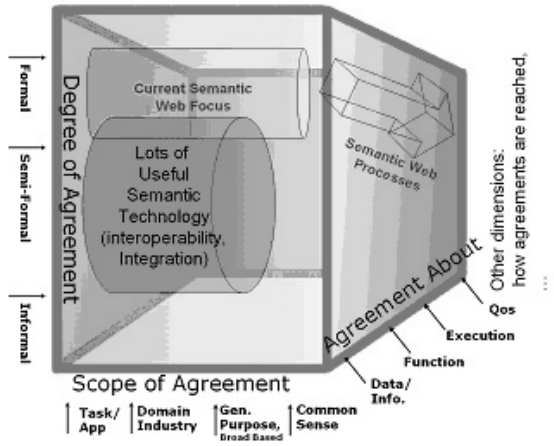


Figure 1: Dimensions along which ontologies vary

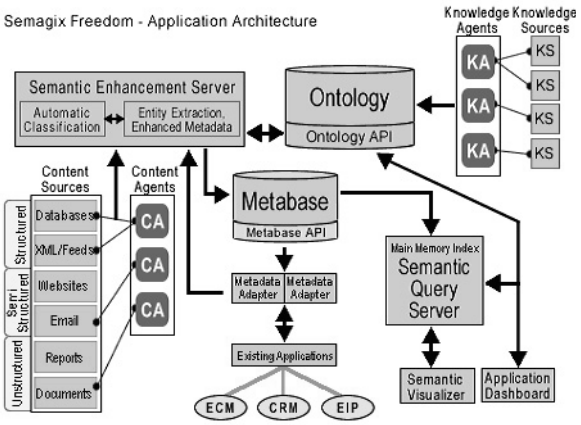


Figure 2: Semagix Freedom Architecture

Freedom provides a modeling tool to design the ontology schema based on application requirements. The domain specific information architecture is dynamically updated to reflect changes in the environment, and is easy to configure and maintain. The Freedom ontology is populated with knowledge–any factual, real-world information about a domain in the form of entities, relationships, attributes and constraints. The ontology is automatically maintained by Knowledge Agents Figure 2, top right). These are software agents created without programming that traverse trusted knowledge sources that may be heterogeneous, but either semi-structured or structured (i..e, concept extraction from plain text to populate ontology is currently not supported but may be supported in future). Knowledge Agents exploit structure to extract useful entities and relationships for populating the ontology automatically. Once created, they can be scheduled to automatically keept the ontology up-to-date with respect to changes in the knowledge sources. Semantic ambiguity resolution (between entity instances) is one of the most important capabilities associated with this activity, as well as with the metadata extraction. Ontology can be exported in RDF/RDFS barring some constraints that cannot be presented in RDF/RDFS.

Freedom also aggregates structured, semi-structured and unstructured content from any source and format. Two forms of content processing are supported: automatic classification and automatic metadata extraction. Automatic classification utilizes a classifier committee based on statistical, learning, and knowledgebase classifiers. Metadata extraction involves named entity identification and semantic disambiguation to extract syntactic and contextually relevant semantic metadata (Figure 2, left). Custom meta-tags, driven by business requirements, can be defined at a schema level. Much like Knowledge Agents, Content Agents are software agents created without programming using an extensive toolkit. Incoming content is further “enhanced” by passing it through the Semantic Enhancement Server [8]. The Semantic Enhancement Server can identify relevant document features such as currencies, dates, etc., perform entity disambiguation, tag the metadata with relevant knowledge (i.e, the instances within the ontology) and produce a semantically annotated content (that references relevant nodes in the ontology) or a tagged output of metadata. Automatic classification aid metadata extraction and enhancement by providing context needed to apply the relevant portion of a large ontology.

The Metabase stores both semantic and syntactic metadata related to content. It stores content in a relational database as well as a main-memory checkpoint. At any point in time, a snapshot of the Metabase (index) resides in main memory (RAM), so that retrieval of assets is accelerated using the Semantic Query Server. This index is both incremental (to keep up with new metadata acquisition) and distributed (i.e., layered over multiple processors, to scale with number of contents and size of the Metabase). The Semantic Query Server is a main memory-based front-end query server. The Semantic Enhancement and Query Servers enable semantic applications (or agents) to query Metabase and ontology using http and Java-based APIs, returning results in XML with published DTDs. This ability, with the context provided by ontology and ambiguity resolution, form

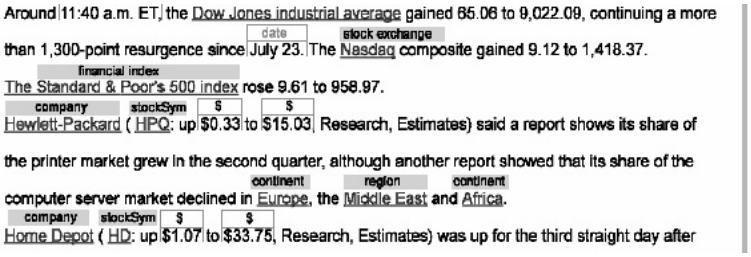


Figure 3: An example of Automatic Semantic Metadata Extraction/Annotation

the basis for contextual, complex, and high performance query processing, providing highly relevant content to the semantic applications. We end the review of Freedom by summarizing some of its characteristics:

- Typical size of an ontology schema for a domain or task ontology: 10's of (entity) classes, 10's of relationships, few hundred property types
- Average size of ontology population (number of instances): over a million of named entities
- Number of instances that can be extracted and stored in a day (before human curation, if needed): up to a million per server per day
- Number of text documents that can be processed for automatic metadata extraction per server per day: hundreds of thousands to a million
- Performance for search engine type keyword queries: well over 10 million queries per hour with approx. 10ms per query for 64 concurrent users
- Query processing requirement observed in an analytical application: approx. 20 complex queries (involving both Ontology and Metabase) to display a page with analysis, taking a total of 1/3 second for computation (roughly equivalent to 50+ queries over an RDB with response time over 50 seconds).

5 Conclusion

Formal ontologies in description logic based representation; supported by deductive inference mechanisms may not be the primary (and certainly not the only) means of addressing major challenges in realizing the Semantic Web vision. The database community should realize that the Semantic Web vision is not one of solving the AI problem, or OWL with subsumption based inference mechanisms. Instead, it can make critical contribution to the Semantic Web by drawing upon its past work and further research on topics such as supporting processing of heterogeneous data/content, semantic ambiguity resolution, complex query processing involving metadata and knowledge represented in semi-formal ontology, and ability to scale with large amount of structured and semi-structured information. In supporting this view point, we provided an overview of one instance of the commercial technology that has been used to develop a broad variety of real world semantic applications. We also provided high level information on scalability requirements observed in supporting these applications. Alternative strategies to realize the vision of the Semantic Web will need to show they will need to scale and perform at least as well as what todays commercial technologies (such as the one briefly discussed in this article) do, and probably well beyond that.

Acknowledgements: Comments to a draft from Dean Allemang, Kemafor Anyanwu, Vipul Kashyap, Harumi Kuno and Kevin Wilkinson are gratefully acknowledged. We also received editing assistance and comments from members of the LSDIS lab including Chris Halaschek, Christopher Thomas, Meenakshi Nagarajan and Kunal Verma.

References

[1] A. Sheth et al. Semantic association identification and knowledge discovery for national security applications. *Journal of Database Management*, 2004 (to appear).

[2] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69:5-40, 2001.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, May 2001.

[4] M. Brodie. The Long and Winding Road To Industrial Strength Semantic Web Services, 2003. Keynote Talk, *I*SWC 2003, http://iswc2003.semanticweb.org/brodie.pdf.

[5] Gómez-Pérez and Manzano-Macho. A survey of ontology learning methods and techniques, 2003. http://ontoweb.aifb.uni-karlsruhe.de/Members/ruben/Deliverable%201.5.

[6] T. Gruber. It is what it does: The pragmatics of ontology. Invited talk at *Sharing the Knowledge-International CIDOC CRM Symposium*, Mar. 2003. http://tomgruber.org/writing/cidoc-ontology.htm.

[7] R. Guha, R. McCool, and E. Miller. Semantic search. In *The Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[8] B. Hammond, A. Sheth, and K. Kochut. Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content. In *Real World Semantic Web Applications*, pages 29-49. IOS Press, Dec. 2002.

[9] IBM. WebFountain. http://www-1.ibm.com/mediumbusiness/venture_ development/emerging/wf.html.

[10] D. Nardi and R. J. Brachman. An introduction to description logics. *The Description Logic Handbook*, pages 5-44, 2002.

[11] I. Polikoff and D. Allemang. Semantic technology. *TopQuadrant Technology Briefing v1.1*, Sept. 2003. http://www.topquadrant.com/documents/TQ03_ Semantic_ Technology_ Briefing.PDF.

[12] S. Dill et al. SemTag and SemSeeker: Bootstrapping the Semantic Web via automated semantic annotation. In *Proceedings of the 12th International WWW Conference (WWW 2003)*, Budapest, Hungary, May 2003.

[13] S. Schulze-Kremer. Ontologies for molecular biology and bioinformatics. *In Silico Biology*, 2(17), 2002.

[14] Semagix, Inc. Anti-money laundering. http://www.semagix.com/solutions_ ciras.html.

[15] A. Sheth, C. Bertram, D. Avant, B. Hammond, K. Kochut, and Y. Warke. Semantic content management for enterprises and the web. *IEEE Internet Computing*, pages 80-87, July/August 2002.

[16] A. Sheth and R. Meersman. Amicalola Report: Database and Information Systems Research Challenges and Opportunities in Semantic Web and Enterprises. *ACM SIGMOD Record*, 31(4):98-106, Dec. 2002. http://lsdis.cs.uga.edu/SemNSF/.

[17] C. Shirky. The Semantic Web, Syllogism, and Worldview. *Networks, Economics, and Culture*, Nov. 2003.

[18] J. Townley. The streaming search engine that reads your mind. *Streaming Media World*, Aug. 2000. http://smw.internet.com/gen/reviews/searchassociation/index.html.

Putting the Semantic Web to Work with DB Technology*

N.Moreno, I. Navas, J.F. Aldana
Higher Technical School of Computer Science Engineering
Computer Languages and Computing Science
Bulevard Louis Pasteur n^o 35. 29071. Málaga, Spain
{vergara, ismael, jfam}@lcc.uma.es

Abstract

We present an architectural design that provides several services in an integrated environment, such as query/inference capabilities over both data and knowledge, dynamic annotation of web pages, maintaining consistency for repositories and annotated pages, crawling services and automatic matching between schemata. This architecture offers a practical and feasible step towards the design, maintenance, integration and development of semantic web applications within concrete domains, making use of and extending standard database technology. For example, in the field of bioinfomatics, non-expert users need tools to assist them to store, query and place the relevant results of their research efforts. Our methodology provides such tools to leverage information gathered from both the "Deep" and "Surface" Web.

1 Introduction

Semantic Web technologies provide a natural and flexible solution for integrating and combining two levels of abstraction, the data level and the knowledge level, which are related by means of metadata. Information is annotated with semantic content/metadata that conform to a domain ontology. For this purpose, mark-up frameworks [1] have been developed to focus on the "*Surface Web*," those static web pages that can be easily found and indexed by traditional web crawlers. However, a large part of the data is stored in on-line databases that comprise the "*Deep Web*" (e.g., dynamic pages built up from web forms). In consequence, current web search engine technology can only be used to query the small fraction of all the data available on the Internet, that is in the "*Surface Web*." The rest can be queried using form-based interfaces that automatically convert user queries into queries over the databases. Although easy to use, these form-based interfaces reduce expressibility of the queries, only allowing conjunctive queries with selection predicates. The user queries are specified by filling in the form fields, imposing strict constraints (mostly equality) on the attribute values stored in the database.

Traditional database techniques do not support effective and efficient query processing in the Semantic Web context because they lack inference capabilities. Methodologies from the field of artificial intelligence, on the other hand, support inference, but can only handle relatively simple models when persisting and manipulating

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work has been supported by the Spanish MCyT Grant (TIC2002-04186-C04-04)

large quantities of data. The latter find answers about the model which had not been explicitly predefined and that involve fewer, but much more complex, data. In this sense, ontologies contribute to the Semantic Web's goal, making explicit a higher level of abstraction and knowledge than semantic data models, and providing new ways for applying efficient reasoning techniques that are not supported by database management systems.

This paper focuses on the use of ontologies to assist database modeling and query processing. For the latter, we explicitly store the relationships that exist between a semantic data model and a related domain ontology. These mappings enhance the semantic knowledge about data and their structure. The preservation of the relationships allows us to use them during query processing or semantic optimization. This paper is organized as follows: section 2 gives an overview of the system in which our proposal is framed; section 3 details those aspects from the Storage Service relevant in making it easy to understand how the query service works; section 4 describes the query service; and finally we conclude with some remarks and a description of future work.

2 System Overview

The emergence of the Semantic Web has lead to the development of distributed systems which share knowledge by making use of conceptualizations of specific domains, generally defined by means of ontologies. Multiple conceptualizations may describe similar domains using different terminologies, or they may overlap. These issues should be, but are not usually, taken into account when a new system is designed, which leads to the increase of development time and effort. In order to tackle these problems, we propose that the design of the application schema should start with the domain ontology (O_D). This allows us to formally define and store the mappings between the O_D and the application schema. The relationships between the data model of a concrete application and the corresponding domain ontology semantically enrich the information in the application domain, capturing it in a more expressive data model and improving several other processes. For instance, this enables enhanced query processing capabilities and semantic query optimisation.

Furthermore, we think that Web applications should be designed taking into account interoperability and integration issues. In this sense, the proposed design methodology based on the existence of a O_D produces the definition and storage of solid mappings, which could be used both to facilitate interoperability between applications and to build an integration scheme. We have therefore developed a prototype that offers functionalities implemented through integrated tools, such as Protégé, ERWin and RACER. This approach is an integrated solution for the realization of the Semantic Web, which consists of four services: a storage service, a crawling service, a semantic page annotator, and a query service. Web developers can first use the storage service to design and implement the stores for their web site, then (by making use of the annotation service) semantically annotate their web contents. They can then take advantage of the crawling service to extract semantics from other web sites, whether or not those other sites were developed with our prototype system. Finally, non-human users (e.g., intelligent agents) can exploit annotations as well as the query service.

Storage Service: The main functionality of this service is to support the computer aided design of conceptual schemata and to translate them into physical implementations (for the most popular RDBMS such as Oracle, Sybase, SQL Server, etc.), leveraging the expressiveness and reasoning techniques derived from using the ontology as a conceptual model [2]. This is the core of the system, as the *Execution Plan Generator*, the *Crawler* or the *Annotation Generator*, among other components (see Figure 1), require information and knowledge derived from it (a set of mappings that relates data with stored knowledge).

Annotation Generator Service: We support the automatic annotation of both static and dynamic web documents. We assume the use of XML technology, and introduce an annotation function in the document's stylesheet to generate them automatically. For dynamic documents two steps are followed: the first one is to annotate the queries enveloped in the dynamic document, and the second to annotate the results of these queries. For the former we have described an ontology that defines the general structure of a dynamic document. Its annotation consists of automatically generating a public instance of that ontology. The second step, the

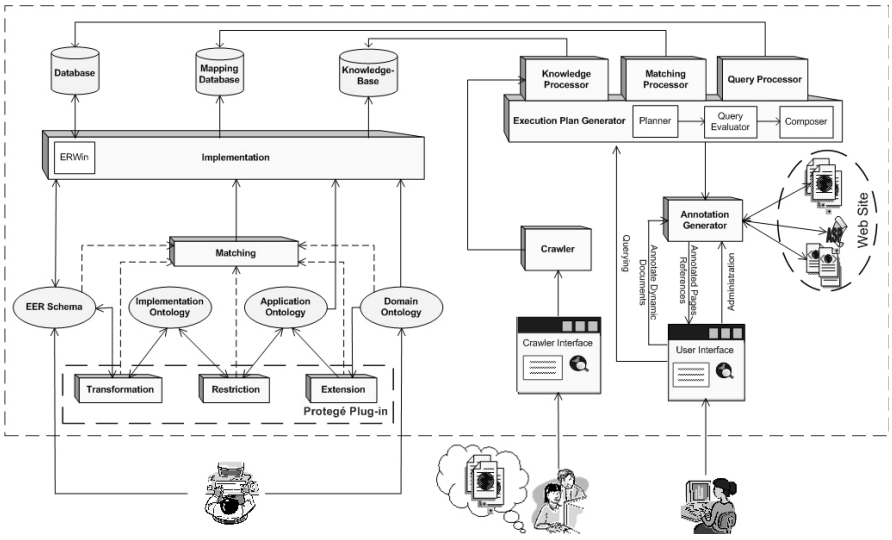


Figure 1: Internal System Architecture

annotation of results of these queries, is addressed applying the same methodology as for static documents.

Crawling Service: This service is an extended semantic crawler that combines [3] and [4] crawling techniques with the semantic annotation of the server-side programs described above to populate the knowledge-base (KB). Static contents are treated like traditional and semantic web crawlers. However, when the crawling process finds a dynamic document, it searches the document with the annotations for queries enveloped in it. The crawler uses this information to prepare an "attack plan" for the dynamic document. As the crawler discovers annotations, they are stored in the KB along with a reference to the web pages in which they were found.

Querying the System: Traditional database systems efficiently resolve a specific kind of query, namely associative queries. Their query capabilities are limited to physical database schemata. In contrast, search engines do not have this restriction, but they cannot derive a concrete answer returning links like query results. Our prototype simulates both behaviors, extending the query capabilities of traditional database systems. Furthermore, it also provides information related to the query results (e.g., links to other web pages). That is, it combines querying and reasoning into a single process.

3 Conceptual Database Design for Semantic Web Applications

The main functionality offered by the Storage Service is to support the computer aided design of conceptual schemata, and to translate them into physical implementations. This service (see Figure 1) requires that an expert user provide the system with a domain ontology following the OWL Lite nomenclature. This feature permits efficient reasoning with the RACER tool [5], which allow users to associate logical formulas to instances. Besides, by using its elementary predefined functions for navigating inside the schema, several questions can be solved and applied during the query processing. For example, we can learn which is the most specific concept one instance belongs to or what properties are related to a relationship (transitive, symmetric, inverse).

To obtain the database conceptual schema we can follow two paths: (1) designing an (E)ER schema from scratch, or (2) using a domain ontology as a foundation and extending it by means of a refinement process In the first case, with the ERWin graphical interface we can design the conceptual schema, which is exported to an intermediate representation in XML. The Matching Component matches the conceptual schema with the domain ontology [6, 7], generating a set of rewriting rules about the relationships between both models.

However, the design based on a domain ontology entails several advantages. To assist this process we

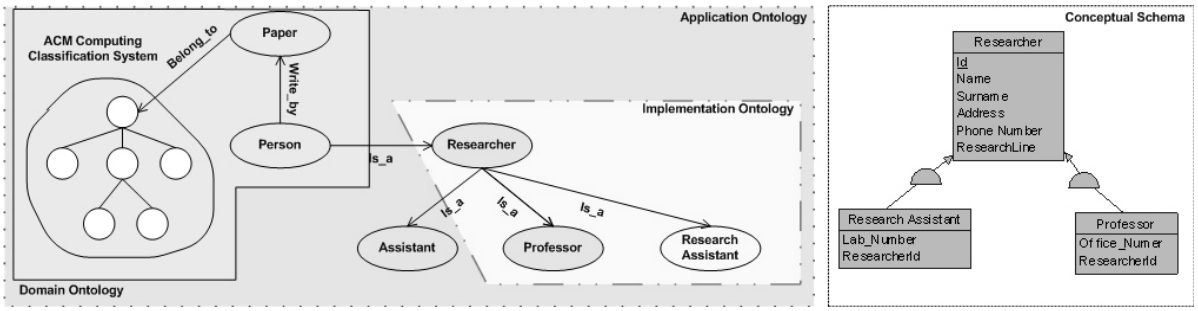


Figure 2: Ontology Creation Process. For simplicity and clarity, slots, rules, constraints, etc. do not appear

have developed a plug-in for the Protégé [8] tool that loads an OWL Lite domain ontology (O_D) and allows the designer to extend it with new relevant concepts, relationships and integrity constraints to the application domain. The result of this extension is an application ontology (O_A) focused on those elements needed for the specific web application under development. Nevertheless, not all the elements represented in O_A are required for a physical implementation (all could be relevant but perhaps persistent instances of all of these concepts are not necessary). Instead, only a selected subset of O_A will be queried by our database applications: O_I makes reference to the ontology obtained in this step, and it verifies $O_I \subseteq O_A$. The selection/renaming of the subset O_I from O_A produces a set of correspondences between these ontologies that is generated by the Restriction Component in order to be used by the Matching Component. Finally, we automatically obtain an enriched ER schema from the implementation ontology [2, 9]. In the process of deriving a final application schema from the domain ontology, we obtain a conceptual model enriched with the expressiveness and the inference capabilities of an ontology. These capabilities can be used and exploited during query processing.

Figure 2 shows an example illustrating the conceptual database schema design for a web application that stores information about authors and their publications. The example extends the *ACM Computing Classification System* domain ontology with classes that provide support for management instances of *Researcher*, *Assistant* and *Professor*, resulting in the O_A. Selecting only those parts that need physical storage from the O_A, we limit the scope of the O_I to the *Researcher* class and its sub-classes, renaming the *Assistant* class as *Research Assistant*. Finally, a translation from the O_I to an (E)ER schema is performed, generating a *Conceptual Schema*.

4 Querying the Surface/Deep Web

The Execution Plan Generator enables users to query annotated Web documents. This component (see Figure 1) analyzes conceptual queries, written in terms of the $O_D \cup O_A$, and can return document links, data and knowledge. From a conceptual point of view and following our design methodology, the data stored in the database can be seen as instances of the knowledge-base due to the fact that O_I (the ontology with which data committed) is contained in $(O_D \cup O_A)$. This particular feature allows us to evaluate queries like an inference process. Users formulate conjunctive queries in terms of $O_D \cup O_A$, which are more expressive than those offered by traditional web forms. Unlike traditional database systems, this extension adds support for reasoning to the system's initial query capabilities. Thus, when the Execute Plan Generator receives a query, the Planner analyzes dependencies between variables to generate the corresponding dependency graph, and locates where each sub-goal can be solved. This last task is carried out applying structural reasoning (e.g., class/sub-class subsumption) against the knowledge-base schema ($O_D \cup O_A$) and the database schema (S_I).

Making use of the information derived from the dependency graph and the location for evaluating each sub-objective, the Planner builds a query graph that is part of the evaluation plan. The latter includes a query graph with a list of sub-goals, which are the leaf nodes, and how they must be combined. Each leaf node includes

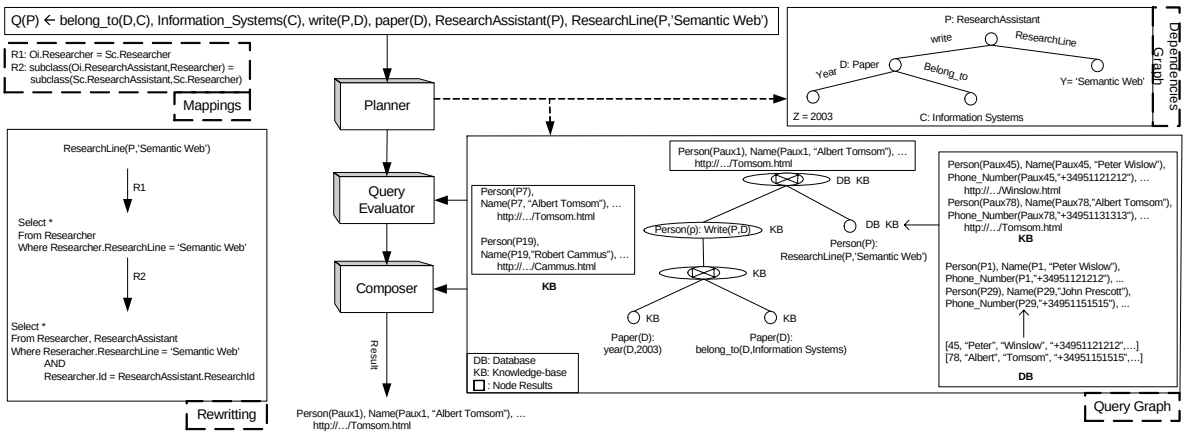


Figure 3: Example for querying the system

information about possible locations where it can be solved. This information is propagated to ancestor nodes of the graph in order to be used in generating the plan. For example, when the Planner needs to build a join node, its behavior depends on the provenance of the descendant nodes. If the nodes retrieve information from the same source, the intermediate node is generated based on a natural composition over unique key identifiers. Otherwise, the join is based on the similarity between data results. The graph structure defines for itself several alternatives to be evaluated, generating different execution plans. The Planner realizes a cost analysis of these plans and determines which has the lowest cost value based on heuristics, information about connection speeds, etc. The others are stored according to their cost value, in case execution errors occur.

Once the Planner ends, it sends the evaluation plans to the Query Evaluator component. Its main functionality is to rewrite and evaluate each sub-goal in terms of the corresponding sources in which it will be evaluated. For the rewriting phase this component takes advantage of the stored mappings looking for a sequence of correspondence rules that relates the initial sub-goal with an element of the data sources, in other words a class, a slot, a table, a relationship, etc. As users wait for responses in terms of $O_D \cup O_A$, the Query Evaluator must try to generate the inverse sequence too. If it is not possible to rewrite a sub-goal i.e. the direct or inverse mapping sequence is not found, then the execution is aborted and an empty result is returned. Using the inverse rules mentioned, partial results are translated to the initial nomenclature.

The Composer receives the execution plans and the results of evaluations from the Query Evaluator component. The reader should note that a leaf node can be resolved in several resources, and in this case it is necessary to combine all results providing a unique solution. Although this approach reduces efficiency as regards response time, we have opted for a more complete solution. Combining results entails having to face traditional issues of heterogeneous database integration, such as replication or inconsistency.

Figure 3 shows an example of query processing in which we solve the query: *Find Research Assistants whose research line is focused on Semantic Web and have a paper in the category Information Systems*. The query is shown in predicate logic terms, due to space limitations. Once the query reaches the Planner, it builds a dependency graph that represents the evaluation precedence of each variable in the query. Starting from this graph the planner generates the query graph, which in this context involves a parallel evaluation of leaf nodes and the composition of intermediate nodes when results are available. Note that each node is annotated with its source location. This annotation allows the Query Evaluator to decide how to rewrite each sub-goal in order to solve it. We have included in this illustration an example of a rewriting process based on stored mapping rules for the evaluation of the database. The annotations of the intermediate nodes provide the Composer with knowledge to decide which type of join must be performed. That is, if a join node only receives information from the knowledge-base, the join is done using the instance identifiers. However, when the information is provided

from the database and the knowledge-base, the Composer combines the results making use of semantic distance among the instances. For example, it can use in the root join a similarity measurement based on the name of the people retrieved from the descendant nodes. In the illustration, the data provided from the knowledge-base and database will be combined deleting the redundancies. In this case, the instances about "Peter Wislow" and "Peter Winslow" are considered redundant and the data from the database are considered the valid ones, but the link to his personal web page is maintained on the combined instance. On the other hand, the root node applies a join based on the content of descendant node instances, due to data provided from the database and the knowledge-base. Note that the result contains information as well as links to web documents.

5 Conclusions and Future Work

In this paper we have presented a system and design methodology for Semantic Web applications, which allow us to translate query processing on databases to an instance retrieval process on inference engines. We enable such applications to combine traditional database systems' ability to manage large amounts of data with the expressiveness and the semantic enrichment of a conceptual model, bringing new query/inference capabilities, such as structural inference. Ontology users (or agents) will first query the ontology schema to identify what relevant information exists, and then proceed to query the data to extract the desired information from the underlying databases. Besides, we provide a way to automatically annotate both static and dynamic documents. Thus, we achieve the annotation of the Surface Web as well as the Deep Web, allowing to improve web crawling. In addition, the domain ontology used as the starting point in database schema design can act as the mediation schema for the integration of those databases that commit with it. This feature makes it possible to easily translate queries from one data source to another. Thus, we plan to apply this methodology to mediation systems.

References

[1] Semantic Web - Annotation and Authoring. http://annotation.semanticweb.org/tools

[2] A. Borgida, M. Lenzerini, R. Rosati. Description Logics for Data Bases. The Description Logic Handbook: Theory, Implementation and Applications. 2003.

[3] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In VLDB 2001: 27th International Conference on Very Large Data Bases, September 2001.

[4] D. Zeinalipour-Yazti, M. Dikaiakos. Design and implementation of a distributed crawler and filtering processor. In Proc. of NGITS 2002.

[5] V. Haarslev, R. Möller. RACER system description. In Proc. Of the Int. Joint Conf. On Automated Reasoning (IJCAR 2001), volume 2083 of LNAI, pages 701-705. Springer, 2001.

[6] G. Wang, J. Goguen, Y. Nam, K. Lin. Interactive Schema Matching With Semantic Functions. In Semantic Integration Workshop, Sanibel Island FL, October 2003

[7] E. Rahm, P. A. Bernstein. A survey of approaches to automatic schema matching. In The VLDB Journal: Volume 10 Issue , (2001), pages 334-350, 2001.

[8] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, M. A. Musen. Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems 16(2):60-71, 2001.

[9] U. Sattler, D. Calvanese, R. Molitor. Relationship with other Formalisms. The Description Logic Handbook: Theory, Implementation and Applications. 2003.

Functional Modeling of Engineering Designs for the Semantic Web

Joseph B. Kopena and William C. Regli*
Geometric and Intelligent Computing Laboratory
Department of Computer Science, College of Engineering
Drexel University
http://gicl.cs.drexel.edu/

Abstract

A goal of this paper is to briefly describe work on a novel application of the Semantic Web to design repositories—collections of diverse engineering knowledge with sophisticated reasoning services. However, a more general goal of this paper is to demonstrate the extra utility enabled in database-styled applications by the incorporation of even simple semantics. It also discusses the development of practical, useful representations and applications for the Semantic Web.

1 Introduction

Engineers spend large portions of their time searching through corporate legacy data and catalogs searching for existing solutions which can be modified to solve new problems or to be assembled into a new device [14]. Corporations maintain huge collections of current and past product designs, each associated with a wide variety of media and information, including detailed geometry, tolerances, mechanical assembly models, finite element meshes, simulation output, and natural language documentation. In current practice such information is often underutilized due to the absence of adequate methods and tools to organize, manage, and work with such bodies of design knowledge. Browsing and navigating such collections are based on manually-constructed categorizations which are error prone, difficult to maintain, and often based on an insufficiently dense hierarchy. Search functionality is limited to inadequate keyword scanning or matching on overly simplistic attributes.

Design repositories [11, 13] are an evolution of traditional design databases. They aim to overcome current limitations through the application of knowledge representation techniques. Function, behavior, rationale, and other aspects of the designs are captured and reasoned on to enable search, categorization, and other tasks in support of the engineer, similar to case-based reasoning [5]. Figure 1 depicts this process.

Although many standards and assembly representations exist for capturing engineering artifact data, few are adequate for use in a design repository. Most, such as the prominent STEP AP203, capture only detailed geometric information, ignoring the wide range of information available and in particular not capturing abstract concepts such as function. In addition, most define only a shared syntax with human-interpretable semantics, unsuitable for automated reasoning beyond a syntactic level.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Also of the Department of Mechanical Engineering; Email: regli@drexel.edu

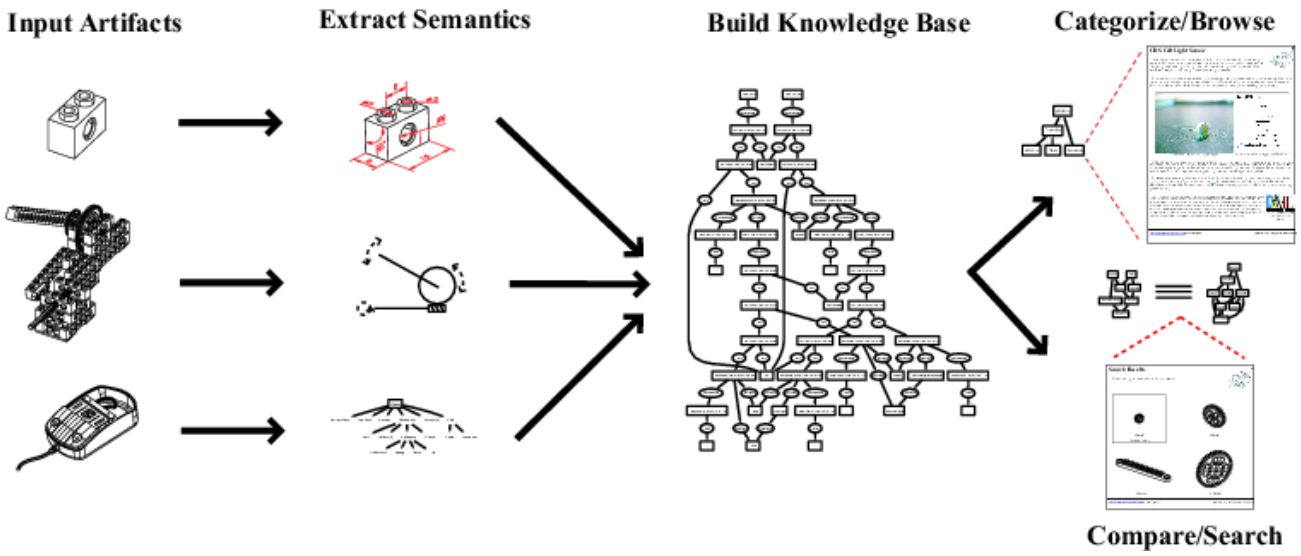


Figure 1: The design repository process.

In this work, description logics and the Semantic Web are used to develop a representation and set of reasoning mechanisms which can support a design repository. An ontology of electromechanical devices based on function and flow is used in representing, classifying, and comparing such devices. Description logic reasoning mechanisms are used to provide design repository inferences not possible with traditional database implementations. New methods of repository interaction, architecture, and applications are enabled by Semantic Web technologies, including uses outside engineering design.

This paper is organized as follows: Section 2 describes the representation developed in this work. Reasoning tasks required of a repository are demonstrated using this representation in Section 3. The role of the Semantic Web in this work is discussed in Section 4. Finally, Section 5 presents some closing remarks.

2 A Representation Based on Function

In order to tackle the many abstract concepts and underlying structures involved, effective reasoning on assemblies mandates knowledge-based mechanisms. Assemblies are primarily defined by their intended *function*—goals achieved and tasks performed—and the decomposition of that function into subfunctions. These in turn define form, structure, and behavior. It is therefore the representation and reasoning of function to which this work has been initially scoped. Reasoning at this abstract, design rationale-oriented level is best accomplished through the application of knowledge representation and reasoning techniques. Doing so under a framework with clear semantics, desirable for automated reasoning, entails formal logic.

Applying knowledge representation to repository reasoning requires an *ontology* for assemblies, a formal description of a conceptualization of this domain. It has been assumed in this work, in the absence of evidence to the contrary, that a great deal of repository-oriented reasoning can be accomplished within first-order logic. However, an ontology in this domain with strong semantics defined in first order logic faces several issues:

- A quick study of the problem and previous efforts makes it clear that it would be difficult to construct. This observation is discussed further within the context of the Semantic Web in Section 5.
- It might be unnecessary, given the problems at hand. The focus on this work is not on solving difficult integration issues, or analysis tasks such as simulation and verification. Rather, repository operations

such as search and compare only require that mechanisms be represented in a manner strong enough to differentiate designs and capture basic properties of a device.

- Although more favorable than in higher order logics, reasoning in first order logic is still not advantageous in that it is NP-Complete and only semi-decidable. In addition, it seems plausible that the complexity of a strong ontology in this domain might overcome practical capabilities of current first order logic reasoners. These points are important in light of the potentially large volumes of designs and knowledge that a repository might work with, especially when tasked with organizing and maintaining legacy data.

Each of these issues can be addressed by utilizing a subset of first order logic. This work has studied the use of description logics; in particular the logic $\mathcal{ALEN}$[1]. Their lower expressivity limits the ability to define detailed semantics for a domain. However, in return they make several positive tradeoffs, including desirable computability and tractability results [1]. Within the repository application, this loss of expressiveness is acceptable so long as enough design knowledge can be captured to enable necessary operations, such as matching devices against adequately sophisticated searches and comparing mechanisms.

This work employs a representation based on function and *flow*—the materials, energies, and signals on which functions operate. Heavily based on current functional modeling research within engineering design theory [12, 7, 15], this effort differs in that the representation is placed within a formal framework, description logic, enabling automated reasoning. The representation is broken up into two parts: a core ontology, defining the basic structure of a function-based mechanism description language, and vocabulary extensions, providing terminologies with which rich descriptions can be written in the language.

The core ontology defines the universe of objects as consisting of assemblies, components, functions, and flows. Several relations are also given, for example to associate assemblies with functions and functions with their input and output flows. The vocabulary extensions provide taxonomies of functions and flows derived from those presented in [12] and [7], which were developed from large surveys of engineers. Figure 2 illustrates this representation with a description of a CDS Cell, a common sensor shown in Figure 2(a). A graphical form of the description is given in Figure 2(b) as a *function and flow diagram*, similar to those presented in [12] but not identical. This diagram is interpretable as statements in a description logic model as in Figure 2(c).

This representation does not rigorously and unambiguously capture the semantics of mechanisms. Instead, it provides a language expressive enough to describe and distinguish devices while maintaining efficiency and computability. It is neither so formal as to prevent practical computing, nor so informal as to prohibit automated reasoning. The following section outlines the use of such reasoning in a design repository.

3 Reasoning for Design Repositories

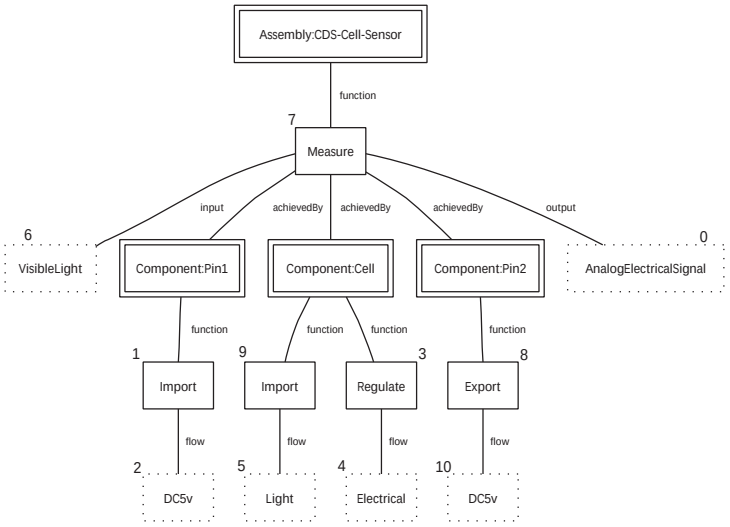
Standard description logic inferences can be used in a number of repository reasoning tasks. Figure 3 demonstrates the most obvious of these, *classification*. In addition to being interpretable as statements in a description logic model, the function and flow diagrams presented in this paper may also be interpreted as concept descriptions. In this way, they may be used to manually create a categorization against which devices in the repository are sorted. Figure 3(a) shows such a hierarchy for some simple sensors and effectors. In this example, the CDS-Cell from Figure 2 has been classified as a light sensor because it matches the definition of the class, given in Figure 3(b)[2]. Such reasoning can greatly improve current, slow, error prone, and limited manual classification.

Search functionality may be similarly implemented, treating the query as a concept description and classifying device instances against that definition. Description logic *subsumption* may be used to manually construct a categorization in a bottom-up, generally more intuitive fashion. Less standard inferences may also be applied, for example induction of the *least common subsumer* of sets of classes [9] may be used to automatically create

[1] A good reference for description logic expressivity notations is [1].

[2] In the DL language $\mathcal{EL}$.

(a) Typical CDS Cell.



(b) Simplified function and flow diagram (numbering on anonymous nodes corresponds to skolemization scheme used in Figure 2(c)).

⟨ ⟨ *CDS-Cell-Sensor type Assembly* ⟩ ⟨ *Cell function node3* ⟩ ⟨ *node7 type Measure* ⟩ ⟨ *Pin2 type Component* ⟩ ⟨ *node8 flow node10* ⟩
⟨ *CDS-Cell-Sensor function node7* ⟩ ⟨ *node2 type DC5v* ⟩ ⟨ *node1 type Import* ⟩ ⟨ *node9 type Import* ⟩ ⟨ *node8 type Export* ⟩
⟨ *Pin2 function node8* ⟩ ⟨ *node5 type Light* ⟩ ⟨ *node7 input node6* ⟩ ⟨ *node4 type Electrical* ⟩ ⟨ *node7 output node0* ⟩
⟨ *node0 type AnalogElectricalSignal* ⟩ ⟨ *node6 type VisibleLight* ⟩ ⟨ *Pin1 type Component* ⟩ ⟨ *Cell function node9* ⟩ ⟨ *Pin1 function node1* ⟩
⟨ *node1 flow node2* ⟩ ⟨ *node7 achievedBy Cell* ⟩ ⟨ *Cell type Component* ⟩ ⟨ *node7 achievedBy Pin2* ⟩ ⟨ *node10 type DC5v* ⟩
⟨ *node3 flow node4* ⟩ ⟨ *node3 type Regulate* ⟩ ⟨ *node7 achievedBy Pin1* ⟩ ⟨ *node9 flow node5* ⟩ ⟩

(c) Figure 2(b) in equivalent description-logic model form.

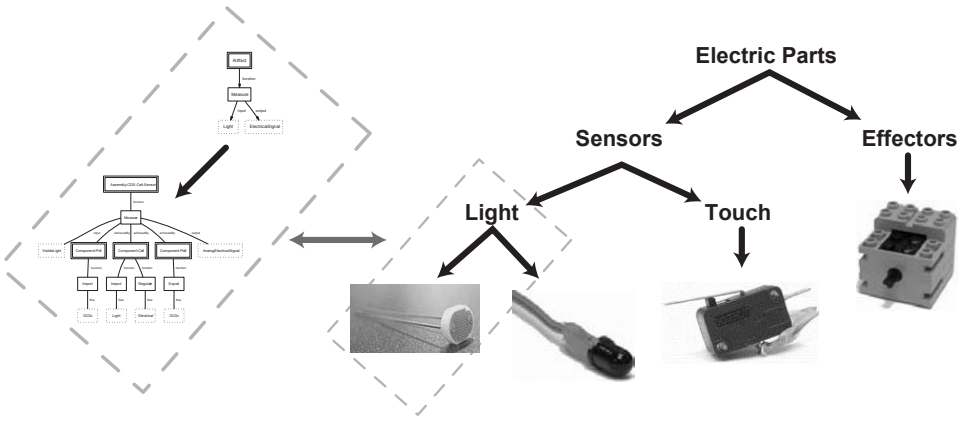
Figure 2: Function and flow modeling of a Cadmium Sulfide (CDS) Cell, a common photoresistor.

a categorization from a collection of devices. This has the potential to be a powerful information management tool once techniques have been developed to address several issues, e.g. the density of the generated hierarchy.

The capacity of the underlying reasoning mechanisms to operate effectively in a system the size of a full-sized design repository is a significant concern. As noted above, this is a primary motivation in using description logics. A formal logic is necessary for defining and reasoning with the semantics of a devices at an abstract, knowledge-based level. Employing a formal logic as the representation framework enables:

- The ability to infer implicit knowledge and data.
- Proofs or characterizations of a reasoning algorithm's soundness and completeness properties.
- Well-defined mechanisms for manipulating knowledge, e.g. deriving the least common subsumer.
- Improved integration and interoperation between heterogenous systems.

Description logics strike a favorable balance between providing a level of expressiveness suitable for these tasks and maintaining efficient, scalable reasoning [2]. Algorithms and implementations do exist for these and other logics which support effective, large-scale reasoning, e.g. [6, 8, 4, 10, 3]. Scalability and practical usability is a legitimate major concern in any effort operating on data at the scale of a design repository. However, this does not necessarily prohibit the use of knowledge-based techniques.



(a) Example part hierarchy with function and flow diagram for the class of light sensors.

$$LightSensor \equiv Artifact \sqcap \exists function.[Measure \sqcap \exists input.Light \sqcap \exists output.ElectricalSignal].$$

(b) Definition of class of light sensors from diagram in Figure 3(a).

Figure 3: Reasoning example: classifying against a hierarchy.

4 Semantic Web

The focus in this work is mostly on the repository reasoning tasks enabled by using a representation grounded in description logic. However, Semantic Web technology has been used in trial implementations of this system, in particular the W3C Ontology Web Language (OWL)[3]. OWL is a description logic language which may be embedded in Web content. Figure 4 provides examples of the two primary uses of OWL in this work: defining ontologies as in Figure 4(a), and encoding the representation for a particular device as in Figure 4(b). The underlying description logic of OWL contains more expressivity than appears to be necessary for this work.

The Semantic Web introduces several new capabilities to the repository setting. One is that unlike existing centralized database architectures, a Semantic Web-based repository may keep its data distributed across the Internet or an intranet. Only device markup and links to the original sources and associated data must be stored by the repository service. More interestingly, the ability to markup device-related Web content with the representation outlined in this paper makes it straightforward to post a wide variety of data—text, CAD, images, simulations—and publish it to any number of repositories and other reasoning engines. Within engineering design, this makes it easier for loosely coupled design teams to collaborate, for example by avoiding redundant design efforts. Another application is for part catalogs and similar materials to be marked up on corporate websites. That information may then be aggregated by a portal site providing an interface to many vendors. With a repository running at the core of the site, large amounts of data may be collected, organized, and utilized in more robust and sophisticated ways than under current practice.

Such a portal site also has applications outside of formal engineering design. With appropriate interfaces, the same techniques could be used to provide portals for e-commerce users shopping for household appliances, electronics, and other purchases. Repositories could support hobbyists in pooling a community's knowledge. They could also be applied in educational settings, for example as a medium for design students to exchange ideas, as well as to search for information online.

[3] `http://www.w3.org/2001/sw/WebOnt/`

```
<owl:Class rdf:about="&eng;#Artifact" />
<owl:Class rdf:about="&eng;#Function" />
<owl:Class rdf:about="&eng;#Flow" />

<owl:Class rdf:about="&eng;#Assembly">
  <rdfs:subClassOf rdf:resource="&eng;#Artifact" />
</owl:Class>

<rdf:Property rdf:about="#function">
  <rdfs:domain rdf:resource="#Artifact" />
  <rdfs:range rdf:resource="#Function" />
</rdf:Property>

        ...

<owl:Class rdf:about="&eng;#Function">
  <owl:disjointWith rdf:resource="&eng;#Flow" />
  <owl:disjointWith rdf:resource="&eng;#Artifact" />
</owl:Class>

        ...

<owl:Class rdf:about="&flow;#AnalogElectricalSignal">
  <rdfs:subClassOf rdf:resource="&flow;#Electrical" />
  <rdfs:subClassOf rdf:resource="&flow;#Signal" />
</owl:Class>
```

(a) Snippets from core ontology along with function and flow vocabulary extensions.

```
<eng:Assembly rdf:about="#CDSCellSensor">
  <eng:function>
    <func:Measure>
      <eng:input><flow:VisibleLight /></eng:input>

      <eng:achievedBy>
        <eng:Component rdf:about="#Pin1">
          <eng:function>
            <func:Import>
            <eng:flow>
              <flow:DC5v />
            </eng:flow>
            </func:Import>
          </eng:function>
        </eng:Component>
      </eng:achievedBy>

      ...

      <eng:output>
        <flow:AnalogElectricalSignal />
      </eng:output>
    </func:Measure>
  </eng:function>
</eng:Assembly>
```

(b) Portion of CDS Cell function and flow representation as given in Figure 2.

Figure 4: Ontologies and data in OWL form.

5 Conclusions

This paper has briefly introduced work on applying description logics to constructing design repositories. An ontology of devices based on function and flow has been presented along with an outline of the use of several associated reasoning mechanisms in supporting such an application. The authors feel this style of ontology is one that could serve the Semantic Web well—a light, easy to author form which provides enough reasoning abilites to be useful without incurring the costs of more sophisticated ontology. For at least the near future, more complex ontologies will be difficult to develop, stalling successful deployment and adoption, will overwhelm potential users and limit the utility of the Semantic Web, and may potentially burden automated reasoners. This paper has also briefly discussed applications of this work to the Semantic Web. It is the authors' hope that this technology will develop into a mature, usable application in both the engineering design and home-use domains and provide an example of new capabilities enabled by the Semantic Web.

Acknowledgements: This work supported in part by National Science Foundation (NSF) Knowledge & Distributed Intelligence (KDI) Grant CISE/IIS-9873005 and CAREER Award CISE/IIS-9733545 as well as Office of Naval Research (ONR) Grant N00014-01-1-0618. All opinions, findings, and conclusions expressed in this material are those of the author(s) and not necessarily those of the supporting organizations.

References

[1] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.

[2] Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[3] Paul R. Cohen, Robert Schrag, Eric K. Jones, Adam Pease, Albert Lin, Barbara Starr, David Gunning, and Murray Burke. The DARPA high-performance knowledge bases project. *AI Magazine*, 19(4):25–49, 1998.

[4] M.P. Evett, J.A. Hendler, and L. Spector. Parallel knowledge representation on the connection machine. *Journal of Parallel and Distributed Computing*, 22:168–184, 1994.

[5] James E. Fowler. Variant design for mechanical artifacts: A state-of-the-art survey. *Engineering with Computers*, 12:1–15, 1996.

[6] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nedel, editor, *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, August 4–10, Seattle, Washington 2001. pgs. 161–166.

[7] J. Hirtz, R. Stone, D. McAdams, Szykman S, and K. Wood. Evolving a functional basis for engineering design. In *ASME Design Engineering Technical Conferences, 13th conference on Design Theory and Methodology*, New York, NY, USA, September 9-11, Pittsburgh, PA 2001. ASME, ASME Press. DETC01/DTM-21688.

[8] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705, pages 161–180. Springer-Verlag, 1999.

[9] Ralf Küsters and Ralf Molitor. Computing least common subsumers in $\mathcal{ALEN}$. In Bernhard Nebel, editor, *Seventeenth International Joint Conference on Artificial Intelligence*, pages 219–224. Morgan Kaufmann, 2001.

[10] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):168–184, 1995.

[11] S. Szykman, C. Bochenek, J. W. Racz, J. Senfaute, and R. D. Sriram. Design repositories: Engineering design's new knowledge base. *IEEE Intelligent Systems*, 15(3):48–55, May/June 2000.

[12] S. Szykman, J. W. Racz, and R. D. Sriram. The representation of function in computer-based design. In *ASME Design Engineering Technical Conferences, 11th International Conference on Design Theory and Methodology*, New York, NY, USA, September 12-16, Las Vegas, NV 1999. ASME, ASME Press. DETC99/DTM-8742.

[13] Simon Szykman, Ram D. Sriram, and William C. Regli. The role of knowledge in next-generation product development systems. *ASME Transactions, the Journal of Computer and Information Science in Engineering*, 1(1):3–11, March 2001.

[14] David G. Ullman. *The Mechanical Design Process*. McGraw-Hill, Inc., 1997. ISBN 0-07-065756-4.

[15] William H. Wood and Manish Verma. Functional modeling: Toward a common language for design and reverse engineering. In *ASME Design Engineering Technical Conferences, 15th International Conference on Design Theory and Methodology*, New York, NY, USA, September 2-6, Chicago, Illinois 2003. ASME, ASME Press. DETC03/DTM-48660.

The Role of Semantic Web Technology in Enterprise Application Integration

Christoph Bussler
Digital Enterprise Research Institute (DERI)
Chris.Bussler@DERI.ie

Abstract

Enterprise Application Integration (EAI) in today's form focuses on the syntactic integration of application interfaces on an implementation technology level. Current EAI technology focuses on the "plumbing" aspect by emphasizing different technologies like queuing [5, 8] or remote invocations [12, 14]. A formal description of the semantics of the interfaces necessary to make integration work flawlessly is not supported by applications or integration technology in any explicit form yet, for example, as process models or formal ontologies. Instead, implementing the correct integration semantics is a human designer task and the result is consequently subject to mistakes and mis-interpretations. This article introduces a first step toward making formal semantics explicit in EAI.

1 Introduction

The main goal of enterprise application integration (EAI) [2] is the semantically correct integration of enterprise application systems like ERP (Enterprise Resource Planning) systems [7, 13]. Enterprise application systems have technical interfaces that are implemented in various forms. Examples are programming language-defined interfaces (e.g., Java [6]) or XML schema definitions [17].

While this allows the correct syntactic integration through passing valid instance data on a technical level between enterprise application systems, their semantics are not formally defined. Instead, the engineer integrating the enterprise application systems has to know the meaning of the low-level data structures in order to implement a semantically correct integration. This means that the correctness of the integration depends completely on the knowledge of the engineer and his flawless implementation of the integration. No formal definition of the interface data exists to assist him, let alone any formal automatic verification. Due to this lack of automated support in defining integration, it takes a long time for a human engineer to define semantically correct integration. In addition to the development effort, inconsistent business data due to faulty integration are costly for an organization due to potential loss of business caused by faulty downstream business decisions.

The advent of Semantic Web technology like ontology languages (such as RDF [10], RDF/S [11], OWL [9]) promises to improve this bleak situation significantly by enabling the formal semantic description of the semantics of the data structures exchanged with enterprise application systems. The concepts as well as the relationships between the concepts are defined through an ontology language. Based on this semantic definition

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

the knowledge about the meaning of the data is formally encoded and the engineer can rely on this. Instead of learning the meaning of the data based on experience and instead of interpreting the data manually the engineer has access to a formal definition of the data. The semantic definition of the data is part of the enterprise application system and therefore leaves the engineer with few, if any, mis-interpretations.

This paper outlines a general enterprise application integration architecture and its elements related to the semantic data definition of enterprise application interfaces. It focuses subsequently on the data interpretation and transformation problem which lies at the heart of every enterprise application integration effort. It defines a general schema for augmenting the enterprise application integration architecture with Semantic Web technology in order to support the semantically correct enterprise application system integration.

2 Enterprise Application Integration

This section characterizes the EAI problem briefly from the viewpoint of interfaces and the definition of data passed through them.

2.1 Applications are HAD systems

An appropriate characterization of application systems in context of EAI is that applications are HAD (heterogeneous, autonomous and distributed) systems:

- **Heterogeneous** refers to the fact that each application system implements its own data model defining its relevant concepts in its own way. One application might implement a customer as a separate object while another application system might implement customer as a special case of trading partner. In both cases there might be similar as well as dissimilar attributes of the concept customer.
- **Autonomous** means that application systems update their internal instance data independent of any other application systems. Furthermore, new versions of the application system software might become available at any point in time changing the underlying data model and consequently the instance data representations. An upgrade to this new version usually happens independently of upgrades of other systems.
- **Distributed** means that applications systems in general implement their data model in their own data repository (usually in form of a database management system) and these are not shared with other application systems.

In summary, application systems are standalone software entities that are defined in isolation and are operated autonomously. Any form of integration of one application systems with another application system happens "outside" of the application systems involved, meaning, that they are not aware of the fact that they are synchronized in any form with each other.

2.2 The Need for Integration

The need for integration of application systems arises when they are "concerned" about the same concepts and their instance data. This "concern" can have different forms:

- **Copy Synchronization.** If an organization deploys several application systems then it might be the case that the same business instance is maintained in several of them concurrently. For example, a particular customer may be represented in two different systems. A change of one has to be propagated to the other.
- **Replica Synchronization.** In the copy case the propagation is "non-transactional," meaning, that before the propagation of change two variants of an instance data are visible. In the replica synchronizing case this is avoided which ensures that all copies are synchronized before being made available (replica).

- **Business Process Participation.** A business process changes the states of objects in a pre-defined sequence as it is executed. These are normally related since they are concerned about a business case (for example request for quotation, purchase order processing, invoicing). The integration therefore ensures that the appropriate application systems are involved as the business process progresses and therefore implements data flow across the application systems.

Independent of the particular reason for integrating application systems it is necessary to access the application systems' interfaces in order to manipulate their internal data instances in order to achieve the integration. This manipulation is the "classic" insert, update, delete access. However, in general, more than one data instance is modified and so their combined modification must leave a consistent state behind. Application systems implement different interface mechanisms on a technical level and these are discussed next.

2.3 Application Interfaces

Application systems provide a myriad of different forms of interface technologies through which they can be accessed. Some examples are persistent and transactional queuing systems, file systems, data base tables, or remote procedure invocation. Each of the interface technologies has its own interface description approach (if at all). For example, data base tables are defined through system tables that can be queried. In contrast, interfacing through a file system does not provide any interface definition at all; the meta data for the data within the interfacing files is not explicitly managed by the file system but outside of it, if at all. The reason is that file systems do not provide a mechanism to define the files' content type definition.

A variety of data formats for data instances can be found as well as different formats for meta-data definitions. Independent of the particular mechanism, today usually no mechanism exposing semantic meta-data about the interfaces or concepts behind interfaces is provided. All definitions are created from a syntactic programming viewpoint, not from a semantics viewpoint. For example, each interfacing message type in a queuing system is defined individually as a data structure. No shared concepts are explicitly called out that are used in different message types (like purchase order identifiers or addresses).

In general, data instances are passed to and from application systems by-value. All relevant data instances are part of the data structure that is passed. No references are passed which avoids the de-referencing problem once data leaves the application systems. However, this introduces the copy management problem since copies are passed around, not replicas. Any update of data extracted will not be visible by systems holding copies.

Application systems interfaces are fixed and in general cannot be changed to adjust to particular access needs by the integration technology. The same applies to the application code; it is immutable, too. Hence, integration technology has to cope with application systems as they are built by their respective vendors.

2.4 EAI is Integrating HAD systems

Fundamentally, EAI is integrating HAD application systems that are immutable. All integration functionality has to be implemented outside the application systems by the application integration technology. In general, the only way to access an application system is through its interfaces and they cannot be altered at all.

In consequence this means that the EAI technology itself is a HAD system that has to control other HAD systems by only accessing their interfaces. No state transition an application systems performs can be forced, delayed or avoided by the integration technology except by calling the application system interfaces appropriately. An application system does not ask the EAI technology for permission.

2.5 EAI Technology

EAI technology is implemented as a HAD system itself with specialized interfaces. These specialized interfaces are built with enormous flexibility. The reason is that these interfaces have to be able to adapt to any application

system interface that can be potentially found when integrating application systems. EAI technology interfaces must be able to adapt to the interfaces of the enterprise application systems for it to integrate them.

However, not only the syntactical and technical integration must be possible. The semantics level must be addressed, too, and this is not focus of today's EAI technology. While this is not current focus, it is essential in order to achieve consistent integration so that the application systems that are integrated operate on a consistent set of all data involved.

Since the application system interfaces are immutable, both from a syntactic as well as semantics perspective, the EAI technology must be able to mediate between technical formats of different application systems on a data structure level as well as on a semantics level. On a technical level the EAI technology must be able to receive any particular data format from an application system as well as to pass any particular data format to an application system. If these are different, then the EAI technology must transform the data format representation into each other. This also has to happen on a semantics level. If the same concept is represented in different semantical terms then the EAI technology must transform between these. This semantic transformation is also called mediation. Mediation must ensure that the semantics of the involved concepts does not change at all, i.e., it must be semantics preserving.

3 Semantic Application Interface Integration

The syntactical integration of application system interfaces can be accomplished today using the already mentioned technologies like queueing systems, programming languages or XML. Therefore, it is not further discussed in the following. Each syntactical element (like a data structure in a message queue definition) has a semantic meaning in context of the application system it is part of. This meaning has to be preserved once the data structure leaves the application system through an application system interface and enters the EAI technology. In order to formally define this meaning the proposal is to encode the meaning of the data structures in formal ontology languages like [9] or [11]. If the meaning of the data structures passed through application system interfaces is encoded in a formal ontology then the EAI technology can refer to it. Any designer that is responsible for the integration of application systems can rely on these ontologies and does not have to understand the application system's internals any more.

Of course, the ontologies have to be expressed in such a way that the meaning can be inferred. A bad example would be to introduce a concept "Address" with two attributes, "address–line–1" and "address–line–2", both of type string. In this case the structure of the two attributes would not be made explicit (i.e., not normalized) and the internal structure of them is anybody's guess. However, ontologies are meant to leave out the guessing part. Instead, several attributes, each referring to other address concepts might be the better design, e.g., "street–number", "city", etc., each referring to a separate concept "Street–Number" and "City." For those concepts that can have enumerable instances, the attributes domain has to range over those, tying down the semantics even more. In addition, there are constraints between attributes, like cities have specific zip codes (or no zip codes, like in Ireland). These constraints, when expressed explicitly, will make the interface definition semantically more precise.

Application systems themselves, however, since they are immutable, will not present ontologies as part of their syntactical interface definitions today. While this can be expected in the future, today this cannot be found at all. Therefore, the EAI technology must store the application systems' semantic interface definitions within its own data model. In that sense the application system interfaces' definition is stored as a proxy interface definition. Once the application systems' interfaces are stored, they can be reused in any integration defined within the EAI technology.

In summary, EAI technology has to be able to connect to application systems' interfaces syntactically as well as define those semantically. Once this is achieved, different types of integrations can be defined based on the stored definitions. In general, as discussed above, different application systems will express the same or

similar concepts in very different ways. Since only the EAI technology is aware of these differences (not the applications themselves due to their autonomy) it has the difficult task to mediate between the differences in case one application system has to be integrated with another one.

4 Mediation

In general, every application system defines its own concepts and stores them in its own data model. The providers of application systems do not coordinate the data model design of their applications, but define them independently of each other. This approach results in the situation that every application system has a different implementation of meta-data in its data model even though they may be implementing the same concepts like "customer". Basically, concepts might be shared, but not their specific way of implementation.

For example, one system implements the "customer" concept directly while in another system the "customer" is a subconcept of "trading partner" inheriting all its attributes. While both are representing customers, their attributes might be very different. If both application systems have to be integrated and a customer has to be copied over from one system into another system then the concepts have to be mapped onto each other in all detail in order to ensure that both customer representations are consistent with each other.

In terms of ontologies, mediation is the re-representation of concepts from one ontology in terms of another ontology. If both ontologies are the same, then the mediation reduces itself to a copy functionality. If the ontologies are different, rules have to be put in place that allow the re-representation of concepts. For example, if one ontology has an explicit address concept whereas the other ontology adds the address attributes directly to the customer concept then the rules have to map the attributes of different concepts to each other in order to ensure semantic consistency.

The role of ontologies is therefore to provide a sound and formal as well as explicit representation based on which the mapping between the concepts can be achieved. While ontologies by themselves do not achieve mediation directly, their availability makes mediation easier and more reliable in terms of consistency.

5 Shared Integration Ontologies

A huge improvement over the current situation would be if application systems would share ontologies between each other for the definition and implementation of concepts. If they not only share ontologies, but also use explicit and formal representations then the integration task would be a lot easier. First, ontologies would not have to be extracted from application systems after the fact. Second, since the ontologies are shared, the mediation problem will be reduced (or will not be there any more in the best case) since the same concepts are represented in exactly the same way. Both cases improve interoperability since the uncertainty of establishing semantic correctness would be removed.

However, this almost ideal situation cannot be expected to happen anytime soon. First of all, application systems are in use today and will not be replaced any time soon on a global scale. Second, the sharing of ontologies is not current practise at all. However, describing the semantics of interfaces using ontologies can be expected in the near future and this is a very good start.

6 Web Services and EAI

Web Services, defined as a combination of WSDL [16] and SOAP [14] are presently the proposed method for remote invocation of application systems. Web Services provide a means to synchronously invoke application logic remotely over the HTTP protocol with XML as the transmission format for the transmitted data. Web Services have two major drawbacks: first, they do not provide any ontology support, and second, they do not

provide support for modeling application system interface behavior. As argued earlier, ontology support is extremely important to provide system support for integration. Application system interface behavior is relevant to invoke the exposed interfaces in the correct order. Otherwise errors would occur prohibiting interoperability. Application system interface behavior is discussed next.

6.1 Application System Interface Behavior

Using ontologies as a formal and explicit way of defining interfaces is a very important and essential first step. However, this by itself does not ensure data consistency across the integrated application systems. Different systems require different interface invocation protocols (i.e., which application interface has to be invoked when) in order to achieve a consistent state transition within the application system. While one application system might provide one interface to create a new customer, another might provide different interfaces, one for creating the customer object, one for inserting the name, another one for inserting the address and one for activating the customer in the sense that he can start purchasing goods. In the first case, one invocation completely defines a customer whereas in the second case several interface invocations are necessary. However, these invocations have to follow a specific order: activating the customer cannot be the first invocation, but must be the last after the customer is established.

Like in case of the concepts, different application systems can require different invocation protocols and this might result into an invocation heterogeneity. As in the example of the previous paragraph, one application might provide one interface for customer creation whereas the other application requires several invocations. If both are integrated, behavior heterogeneity occurs (one vs. several interface invocations). This so-called behavior heterogeneity has to be addressed, too, and this is done through an approach called behavior mediation [4]. Behavior mediation ensures that the invocation sequence of one application system is obeyed and so is the one of the other application system. Any mismatch will be mediated through additional invocations or invocations that will not be propagated across systems.

6.2 Process Languages

The definition of behavior is focus of several efforts like DAML-S [3] or BPEL [1]. These efforts try to address the lack of behavior definition in Web Services through process definition languages. They provide language constructs that support the ordering of Web Service invocations. Most of them, however, treat the behavior problem like a workflow problem not distinguishing the external interface behavior (“public process”) of application systems from the internal implementations (“private process”). In addition, most of them do not provide language constructs for either data or behavior mediation. In light of the previously discussed properties and requirements in context of application systems and their integration these process languages can only be seens as a first step towards a practical solution.

7 Outlook

The semantic integration of application systems (EAI) is a hard problem due to the need of semantic description of application interfaces. Not only the data have to be described semantically meaningful, but also the behavior of application system interfaces. First steps toward the semantic description of data communicated by application interfaces can be achieved using ontologies. In addition, the external or public behavior of application system interfaces needs to be established. While for the data aspect ontologies from the Semantic Web research community are available, the process aspect is far from being solved. However, efforts like WSMF [4] or SWSI [15] realized the lack of semantic interface behavior support and are working towards practical solutions.

8 Research at the Digital Enterprise Research Institute (DERI)

The mission of the Digital Enterprise Research Institute (DERI, www.deri.ie) is to combine two fundamental technologies: Semantic Web technology and Web Service technology. The result, called Semantic Web Services, is the foundation for various application areas like Knowledge Management, Enterprise Application Integration and e-Business. For each of these five areas a dedicated group, called cluster, is established focusing on research in the respective area of expertise as well as applying the research results in real-world applications provided through industrial partners. DERI is funded nationally by the Science Foundation Ireland. In addition to Irish funding, international funding is acquired through international projects like European Union 6th Framework projects (see www.deri.ie for the current list).

References

[1] Business Process Execution Language for Web Services, Version 1.1, 5 May 2003, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems

[2] Bussler, c.: B2B Integration. Springer Verlag, 2003

[3] www.daml.org/services/

[4] Fensel, D.; Bussler, C.: The Web Service Modeling Framework WSMF. In: Electronic Commerce Research and Applications, Vol. 1, Issue 2, Elsevier Science B.V., Summer 2002

[5] IBM Webshpere MQ. www-3.ibm.com/software/integration/wmq/

[6] Java. java.sun.com/

[7] www.oracle.com/applications/index.html

[8] Oracle9i. Application Developers Guide - Advanced Queuing, Release 2 (9.2), March 2002, Part No. A96587-01

[9] OWL Web Ontology Language Overview, W3C Candidate Recommendation, 18 August 2003

[10] Resource Description Framework(RDF), Model and Syntax Specification, W3C Recommendation, 22 February 1999

[11] RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 10 October 2003

[12] RPC: Remote Procedure Call, Protocol Specification, Version 2, www.ietf.org/rfc/rfc1057.txt?number=1057

[13] SAP. www.sap.com

[14] SOAP Version 1.2 Part 0: Primer, W3C Recommendation, 24 June 2003

[15] Semantic Web Services Initiative. www.swsi.org/

[16] Web Services Description Language (WSDL) Version 1.2., W3C Working Draft, 3 March 2003.

[17] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001

Sponsored by the
IEEE Computer Society

Supported by
Microsoft Research,
BEA, IBM, MITRE, Sun

CALL FOR PARTICIPATION

20th International Conference on Data Engineering

March 30 - April 2, 2004
Omni Parker House Hotel, Boston, USA

http://www.cse.uconn.edu/cse/icde04

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 20th International Conference on Data Engineering will be held in Boston, Massachusetts, USA -- an academic and technological center with a variety of historical and cultural attractions of international prominence within walking distance.

ICDE 2004 HIGHLIGHTS

We have an exciting program designed to appeal both to researchers and to data engineering professions. It includes:

- Three keynote talks;
- Panels and advanced technology seminars (no additional fee);
- Presentations of 63 research papers (out of 441 submissions);
- Industrial talks, demos of new technology, poster presentations;
- Banquet at the New England Aquarium.

Complete program and registration information can be found on the ICDE 2004 web site.

KEYNOTE TALKS

Steven Hagan, Engineering VP, Server Technologies
New England Development Center, Oracle

David H. Lehman, Senior VP for Information and Technology
The MITRE Corporation

Dr. Eric K. Neumann, VP Strategic Informatics, Head of Knowledge Research
Beyond Genomics

PANELS

Database Kernel Research: What, if anything, is left to do?
Chair: Dave Lomet (Microsoft Research)
Panelists: Michael Brodie, David DeWitt, H.V.Jagadish, Gerhard Weikum

Database Research in the Current Millennium
Chair: Daniela Florescu (BEA Systems)
Panelists: Ioana Manolescu, Anastassia Ailamaki, Jai Shanmugasundaram, Zack Ives (all panelists have Doctorates awarded since 1/1/2000)

Querying the Past, the Present, and the Future
Chair: Dieter Gawlick (Oracle Corporation)
Panelists: Adam Bosworth, Michael Franklin, Christian Jensen

ADVANCED TECHNOLOGY SEMINARS

Implementation and Research Issues in Query Processing for Wireless Sensor Networks (Wei Hong, Sam Madden)

'My Personal Web': A Tutorial on Personalization and Privacy for Web and Converged Services (Irini Fundulaki, Richard Hull, Bharat Kumar, Daniel Lieuwen, Arnaud Sahuguet)

XML Query Processing (Daniela Florescu, Donald Kossman)

Data Mining for Intrusion Detection: Techniques, Applications and Systems (Jian Pei, Shambhu Upadhyaya, Faisal Farooq, Vebugopal Govindaraju)

Data Management in Location-Dependent Information Services (Baihua Zheng, Jianliang Xu, Wang-Chien Lee)

Meta Data Management (Phil Bernstein, Sergey Melnik)

Similarity Search in Multimedia Databases(Daniel Keim, Benjamin Bustos)

ABOUT ICDE 2004

The ICDE 2004 International Conference on Data Engineering provides a premier forum for:

- sharing research solutions to problems of today's information society;
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these;
- raising awareness in the research community of the problems of practical applications of data engineering;
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers;
- identifying new issues and directions for future research and development work.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398