B. Omelayenko, M. Klein (eds.)

Proceedings of the Workshop on Knowledge Transformation for the Semantic Web KTSW 2002

Workshop W7 at the 15-th European Conference on Artificial Intelligence

23 July 2002, Lyon, France

`www.cs.vu.nl/˜borys/events/KTSW02`

Preface

The vision of the Semantic Web envisages the Web enriched with numerous domain ontologies, which specify formal semantics of data, allowing various intelligent services to perform knowledge-level information transformation, search and retrieval. Recent successful projects in the ontology area have resulted at creation of thousands ontologies, development of several ontology-based annotation tools and inference engines.

However, the absence of an efficient transformation technology for distributed and evolving knowledge hampers further developments of the Semantic Web area. Preliminary non-automated knowledge transformation approaches, experimental research prototypes and early proposals of transformation languages need to evolve into a working technology with solid theoretical grounds and powerful tool support.

The workshop attracted a number of high-quality submissions concerning different transformation issues and models presented in the present book. The book is opened with an extended abstract of the invited talk of F. Casati presenting a discussion about the role of services at the Semantic Web.

The first section of the proceedings is devoted to model transformation approaches. The paper on 'Effective schema conversions between XML and relational models' by D. Lee, M. Mani, and W. Chu is followed by the paper on 'Transforming UML domain descriptions into configuration knowledge bases for the Semantic Web' by A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. Generic model transformation issues are discussed in the paper 'On modeling conformance for flexible transformation over data models' by S. Bowers and L. Declambre.

Specific modeling issues are again discussed in the second section. Namely, the problem of 'Tracking changes in RDF(S) repositories' by A. Kiryakov and D. Ognyanov, 'Tracing data lineage using schema transformation pathways' by H. Fan and A. Poulovassilis, and 'An algebra for the composition of ontologies' by P. Mitra and G. Wiederhold.

The next section of the book is devoted to the papers on mapping conceptual models. First, 'Knowledge representation and transformation in ontology-based data integration' by S. Castano and A. Ferrara, then 'MAFRA -An Ontology MApping FRAmework in the context of the Semantic Web' by A. Maedche, B. Motik, N. Silva and R. Volz. These are followed by application-driven approaches 'Conceptual normalization of XML data for interoperability in tourism' by O. Fodor, M. Dell'Erba, F. Ricci, A. Spada and H. Werthner; and 'RDFT: a mapping meta-ontology for business integration' by B. Omelayenko.

The fourth section contains the papers discussing configuration issues: 'Enabling services for distributed environments: ontology extraction and knowledge-base characterization' by D. Sleeman, D. Robertson, S. Potter and M. Schorlemmer; 'The 'Family of Languages' approach to semantic interoperability' by J. Euzenat and H. Stuckenschmidt; and 'A logic programming approach on RDF document and query transformation' by J. Peer.

The last section is devoted to poster presentations and system demonstrations: 'Information retrieval system based on graph matching' by T. Miyata and K. Hasida; 'Formal knowledge management in distributed environments' by M. Schorlemmer, S. Potter, D. Robertson, and D. Sleeman; 'Distributed semantic perspectives' by O. Hoffmann and M. Stumptner; 'The ontology translation problem' by O. Corcho.

We would like to thank the authors for their contributions and wish you to enjoy reading the book.

June 2002

Borys Omelayenko,
Michel Klein,
co-chairs of workshop

Organization

The workshop on Knowledge Transformation for the Semantic Web was held on July 23-th during the 15-th European Conference on Artificial Intelligence, Lyon, France, 21-26 July 2002.

Program Commitee

Michael Blaha	OMT Associates, USA
Harold Boley	German Research Center for Artificial Intelligence, Germany
Christoph Bussler	Oracle Corporation, USA
Hans Chalupsky	University of Southern California (ISI), USA
Detlef Plump	The University of York, UK
Dieter Fensel	Vrije Universiteit Amsterdam, NL
Natasha F. Noy	Stanford University (SMI), USA
Michel Klein	Vrije Universiteit Amsterdam, NL
Borys Omelayenko	Vrije Universiteit Amsterdam, NL
Alex Poulovassilis	University of London (Birkbeck Colledge), UK
Chantal Reynaud	University Paris-Sud, France
Michael Sintek	German Research Center for Artificial Intelligence, Germany
Heiner Stuckenschmidt	Vrije Universiteit Amsterdam, NL
Gerd Stumme	University of Karsruhe (AIFB), Germany

Additional referees

Danny Ayers
Shawn Bowers
Jeen Broekstra
Mario Cannataro
Wesley Chu
Oscar Corcho
Jérôme Euzenat
Hao Fan
Alfio Ferrara
Oliver Fodor
Oliver Hoffmann
Alexander Mädche
Prasenjit Mitra
Takashi Miyata
Damyan Ognyanoff
Borys Omelayenko
Joachim Peer
Stephen Potter
Rafael Pulido
Marco Schorlemmer
Ronny Siebes
Carlo Wouters
Markus Zanker

Sponsoring Institutions

OntoWeb thematic Network
`http://www.ontoweb.org/`

Bibliographic Reference

Proceedings of the Workshop on Knowledge Transformation for the Semantic for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW-2002), Lyon, France, 23 July 2002. Available online at `http://www.cs.vu.nl/˜borys/events/ktsw2002.pdf`

Workshop Homepage

`http://www.cs.vu.nl/˜borys/events/KTSW02`

Table of Contents

Invited Talk

A Conversation on Web Services: what's new, what's true, what's hot. And what's not 1
Fabio Casati

Modeling I

Effective Schema Conversions between XML and Relational Models 3
Dongwon Lee, Murali Mani, Wesley W. Chu

Transforming UML domain descriptions into Configuration Knowledge Bases for the Semantic Web 11
Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, Markus Zanker

On Modeling Conformance for Flexible Transformation over Data Models 19
Shawn Bowers and Lois Delcambre

Modeling II

Tracking Changes in RDF(S) Repositories 27
Atanas Kiryakov, Damyan Ognyanov

Tracing Data Lineage Using Schema Transformation Pathways 36
Hao Fan, Alexandra Poulovassilis

An Algebra for the Composition of Ontologies 43
Prasenjit Mitra and Gio Wiederhold

Mapping

Knowledge Representation and Transformation in Ontology-based Data Integration 51
Silvana Castano, Alfio Ferrara

MAFRA — A MApping FRAmework for Distributed Ontologies in the Semantic Web 60
Alexander Maedche, Boris Motik, Nuno Silva, Raphael Volz

Conceptual Normalisation of XML Data for Interoperability in Tourism 69
Oliver Fodor, Mirella Dell'Erba, Francesco Ricci, Antonella Spada, Hannes Werthner

RDFT: A Mapping Meta-Ontology for Business Integration 77
Borys Omelayenko

Configuring

Enabling Services for Distributed Environments: Ontology Extraction and Knowledge Base Characterisation 85
Derek Sleeman, Stephen Potter, Dave Robertson, W. Marco Schorlemmer

The 'Family of Languages' Approach to Semantic Interoperability 93
Jérôme Euzenat, Heiner Stuckenschmidt

A Logic Programming Approach To RDF Document And Query Transformation 101
Joachim Peer

Posters

Information Retrieval System Based on Graph Matching 110
Takashi Miyata, Kôiti Hasida

Formal Knowledge Management in Distributed Environments 111
W. Marco Schorlemmer, Stephen Potter, David Robertson, Derek Sleeman

Distributed Semantic Perspectives 112
Oliver Hoffmann and Markus Stumptner

A framework to solve the ontology translation problem 114
Oscar Corcho

Author Index 115

A Conversation on Web Services: what's new, what's true, what's hot. And what's not

Fabio Casati

Hewlett-Packard
1501 Page Mill Road, MS 1142
Palo Alto, CA, USA, 94304
Fabio_Casati@hp.com

Hi Tim, what are you doing?

I am writing a paper on Web Services. They are the next wave of Internet-based applications.

Oh! I heard about them, but I was never really able to understand what they are. What's a web service?

Ah. I get this question a lot. It reminds me of when people were asking me "what is an agent?". Well, a Web service is an application that exposes functionalities accessible via the Internet, using standard Web protocols (that's why they are called *Web* services). In particular, the names that are always made are those of XML, SOAP, and WSDL. If your application has an interface described in WSDL, and interacts with clients by exchanging XML messages encapsulated into SOAP envelopes, then it is a web service.

I see. Doesn't seem too exciting, anyway. What's new about it? Sounds just like good old RPC over the Web, only under a different form.

Well, that's true. Conceptually, and technologically, there is nothing particularly new. Perhaps, the biggest difference is that these languages and protocols are supported by pretty much every big software player. This level of support is unprecedented. You don't have to deal with things such as CORBA vs DCOM, java vs C++ vs C#, Solaris vs Windows vs HP-UX vs Linux. With web services standards you go across platforms, from the top to the bottom of the software stack. Application integration becomes easier, because everybody speaks the same language, or at least they use the same grammar. Think about it: One of the problems you have in application integration is that enterprise processes need to access many different systems, each supporting its own language and protocols. Therefore, either you write ad-hoc code for each of them, or you buy an integration platform along with system-specific adapters that hide the heterogeneity and show to the integrating application a uniform view of an otherwise diverse world. But, with XML, SOAP, and WSDL, these adapters will become much simpler, considerably less expensive, and easier to deploy. After all, if Web services become reality, what adapters will end up doing are translations between different XML formats.

Another aspect to keep in mind is that all these languages and protocols are simple. Simplicity is paramount. If you try to make standards too complex, they won't fly. They will be difficult to understand and difficult to implement. SOAP and WSDL are just at the right level to gain acceptance and stimulate the development of design and runtime tools.

mmmm. Yes, makes sense. So, they simplify enterprise application integration and reduce the need for integration platforms. That's a great benefit. Indeed, it's one of the biggest headaches in many of my projects. But tell me one more thing: I never really hear about web services in the context of enterprise application integration. Everybody seems to talk about "dynamic discovery", "loosely-coupled", "Semantic", and that's where the hype seems to be.

Yes, Web services were not born with enterprise application integration in mind. The original goal was (and still is, to some extent) to get to a do-it-for-me Internet. Basically, you should be able to tell your "agent" what you need. Then, this agent will search the Web for the available service that best suits your need, finds out if and how it can talk to the service, invokes the desired functionality, pays for the service, and then brings the results back to you.

Wow! Sounds like magic. How is it done?

Well, with Web services, not only you describe the application interface in a standard language (WSDL) and access its functionalities through a standard protocol (SOAP), but you can also describe it in Internet registries, structured according to another standards, called UDDI. In this way, clients requiring a service can just go to an UDDI directory, enter their search criteria, retrieve the list of services that satisfy their needs, and access these service.

OK, but didn't you have that with other naming and directory services? JNDI and CORBA for example have similar capabilities

Yes. One of the differences, however, lies in the way UDDI is designed. In fact, its purpose is to enable the dynamic discovery of services over the Web, across platforms and across organizations. It's been created from the start with this purpose in mind. Entries in the directory can be posted by any company, and services can be deployed on all sorts of platforms. Therefore, the description needs to be independent of specific languages or platform. Other issues are the need for flexibility and extensibility. You don't want to fix a service description language, data structure, or ontology because you just don't know what will be needed to describe a particular web service or set of web services. For example, sometimes in the future a shoe store standardization consortium may define a standard set of properties of shoes and shoe stores, as well as a description of the behavior that Web shoe stores should have. Right now, not only we do not have a clue about what are the characteristics that users will need to describe shoes and Web shoe stores, but we do not even know what language will be suited to specify their behaviors. Maybe these standardization consortia will want or need to define the semantics in a very detailed manner, using some language that we cannot imagine right now. UDDI let's you do it with the notion of *tModel*: any UDDI client (the standardization body in this example) can define a document (the tModel) that describes the properties that a web shoe store may or must have, in terms of attributes, interfaces, supported protocols, transactionality, and other attributes that maybe we cannot even imagine right now, but that will be important in the future. The structure of this document is open for the most part, and is not interpreted by UDDI. Therefore, you can write specifications in any language. Let's assume that this tModel has been defined, and assigned some identifier (say, 643).

When you describe a web service, you can specify that your service has the property *tModel 643*, meaning that you are compliant with that tModel, and therefore with the specification by the shoe standardization consortium. In this way, clients that have been designed to interact with web shoe stores can look for service provider that supports tModel 643. You can even go into more details, for example specifying that you sell shoes that, according to the definition of "color" given in tModel 643, are "yellow".

Another important characteristic of UDDI is that it also defines how to operate and maintain global directories. You need this if you want client applications to be able to find and access services wherever they are, based only on their properties and not on whether you can locate them or not. It's yet another manifestation of the democracy of the Internet! Big vendors and small shops will look alike, you only select them based on what they offer.

Well, I am a little skeptical about this, Tim. I am sure that big guys will find a way to make you buy from them. But let me understand this tModel. From what you are saying, client applications are not really going to read tModel 643. They just want to know whether a service is compliant with it or not. Basically, it is a human that, when developing the client application, reads the tModel to understand how to interact with web shoe stores, and then writes the application code in a way that it can communicate with such web services. So, the tModel description is meant for humans, isn't it?

That's one use of the tModel. It has benefits in its own right. However, you can use tModels in a more powerful way. For example, if your tModel specifies a WSDL interface, then you can think of tools that simplify the development efforts by reading a tModel and automatically generating the stubs to be plugged into your client application. The next (and most interesting) step consists in formalizing more aspects of a web service within a tModel. In this way, applications could be able to read the tModel associated to a service, find out the interfaces and interaction protocols supported by this service, and understand how to invoke the desired functionality.

See, Tim this is what looks like magic to me. I hear this a lot, but I don't see how it can happen. Let me tell you about my last project. We had to automate our supply chain operations, invoking our business partners automatically for such things as sending and receiving quotes, executing purchase orders, and the like. We decided to use the RosettaNet standard to perform these B2B interactions. As you probably know, RosettaNet defines a large number of very detailed interfaces and protocols for supply chain operations in the IT domain. It has full industry support, it has been carefully designed by all industry leaders, and it has gone through several revisions so that it is now at a good level of maturity. There are also many commercial platforms that support RosettaNet out-of-the-box, and integrate B2B conversations with the execution of your internal processes. Our partners and us had two different platforms supporting this standard. When we tried to perform the B2B interactions, well, nothing worked!! Even if both platforms supported RosettaNet, unless both of us had the same system from the same vendors, we could not communicate.

But that was only one of the problems! Even with identical platforms, we still had to do a lot of work to get things going. The fact is that, even in mature vertical standards, specifications are often ambiguous. In addition, many practical cases have needs that are not supported by the standard. For example, in this project we had to meet face-to-face several times with our partners to actually agree on what is the exact meaning of what we write in the RosettaNet-compliant XML documents that are exchanged. Furthermore, in some cases there were some attributes that we needed to transfer, and there was no place for them in the XML document as designed by RosettaNet. For example, we agreed that we would use a "date" field to enter a line item number.

That's why I am skeptical about all this "dynamic interaction" and "semantic specifications". In many practical situations, not only you are not able to dynamically discover how to talk to your partner, but you are not even able to invoke a service that follows the exact same interface and protocol that your application has been designed to support.

I see. That's an interesting perspective. So, you think that it is not possible to perform any kind of dynamic B2B discovery and interaction over the Web?

Well, no, I would not go that far. I think that you can indeed use UDDI to dynamically search for a service that supports the standard your client application has been designed to interact with. And the support you have in UDDI seems just fine to me. What I am saying is that this can happen for relatively simple cases and for services that are not mission-critical. I would not use it to dynamically find my supply chain partners and interact with them, but I can use it for a PS to PDF converter, or for finding out the movie schedule. Even there, if you put payments into the picture, things become more complex. And not many companies will provide web services for free, given that since the interaction is automated, they cannot even show advertisements to you. The other point you made, about dynamically discovering how to interact with a newly discovered service implementing a protocol that my client was not designed to support, well, that I think will not happen for quite some time. You may find some simple cases for which it works, but I doubt you can have any real deployment around it.

From what you say, this is a generic problem, independent of Web services, SOAP, or UDDI.

Yes the problem is always the same. It's hard to do business automatically with people you don't know and with whom you do not have a contract in place. Not to mention the problem of resolving disputes. But I can see that there are many contexts in which Web service technology is applicable. Enterprise application integration is one of them. You have convinced me that Web services provide significant benefits there. I can see how I can integrate quickly and with lower costs. The same concept, I think, can be extended to closed communities of business partners, where agreements are in place before the interaction starts, and where the details can be worked out by humans.

After all, do you think that Web services are here to stay?

Yes, definitely. They are here to stay.

Effective Schema Conversions between XML and Relational Models

Dongwon Lee[⋆], Murali Mani[⋆⋆], and Wesley W. Chu

UCLA, Computer Science Department,
{dongwon, mani, wwc}@cs.ucla.edu

Abstract. As Extensible Markup Language (XML) is emerging as the data format of the Internet era, there is an increasing need to efficiently store and query XML data. At the same time, as requirements change, we expect a substantial amount of conventional *relational* data to be converted or published as XML data. One path to accommodate these changes is to transform XML data into relational format (and vice versa) to use the mature relational database technology.
In this paper, we present three semantics-based schema transformation algorithms towards this goal: 1) CPI converts an XML schema to a relational schema while preserving semantic constraints of the original XML schema, 2) NeT derives a nested structured XML schema from a flat relational schema by repeatedly applying the $nest$ operator so that the resulting XML schema becomes hierarchical, and 3) CoT takes a relational schema as input, where multiple tables are interconnected through inclusion dependencies and generates an equivalent XML schema as output.

1 Introduction

Recently, XML [1] has emerged as the *de facto* standard for data format on the web. The use of XML as the common format for representing, exchanging, storing, and accessing data poses many new challenges to database systems. Since the majority of everyday data is still stored and maintained in relational database systems, we expect that the needs to convert data format between XML and relational models will grow substantially. To this end, several schema transformation algorithms have been proposed (e.g., [2,3,4,5]). Although they work well for the given applications, the XML-to-Relational or Relational-to-XML transformation algorithms only capture the *structure* of the original schema and largely ignore the hidden *semantic constraints*. Consider the following example for XML-to-Relational conversion case.

Example 1. Consider a DTD that models conference publications:

```
<!ELEMENT conf(title,soc,year,mon?,paper+)>
<!ELEMENT paper(pid,title,abstract?)>
```

Suppose the combination of `title` and `year` uniquely identifies the `conf`. Using the hybrid inlining algorithm [4], the DTD would be transformed to the following relational schema:

```
conf  (title,soc,year,mon)
paper (pid,title,conf_title,conf_year,
       abstract)
```

While the relational schema correctly captures the structural aspect of the DTD, it does not enforce correct semantics. For instance, it cannot prevent a tuple t_1: `paper(100,'DTD...','ER',3000,'...')` from being inserted. However, tuple t_1 is inconsistent with the semantics of the given DTD since the DTD implies that the paper cannot exist without being associated with a conference and there is apparently no conference "ER-3000" yet. In database terms, this kind of violation can be easily prevented by an *inclusion dependency* saying "`paper[conf_title,conf_year]` $\subseteq$ `conf[title,year]`".

The reason for this inconsistency between the DTD and the transformed relational schema is that most of the proposed transformation algorithms, so far, have largely ignored the hidden *semantic constraints* of the original schema.

1.1 Related Work

Between XML and Non-relational Models: Conversion between different models has been extensively investigated. For instance, [6] deals with transformation problems in OODB area; since OODB is a richer environment than RDB, their work is not readily applicable to our application. The logical database design methods and their associated transformation techniques to other data models have been extensively studied in ER research. For instance, [7] presents an overview of such techniques. However, due to the differences between ER and XML models, those transformation techniques need to be modified substantially. More recently, [8] studies a generic mapping between arbitrary models with the focus of developing a framework for model management, but is not directly relevant to our problems.

From XML to Relational: From XML to relational schema, several conversion algorithms have been proposed recently. STORED [2] is one of the first significant attempts to store XML data in relational databases. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and using the DTD, converts XML documents to relational format. Because [9] discusses template language-based transformation from DTD to relational schema, it requires human experts to write an XML-based transformation rule. [4] presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, [3] studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. Unlike these, we propose a method where the hidden semantic constraints in DTDs are systematically found and translated into relational formats [10]. Since the method is orthogonal to the structure-oriented conversion method, it can be used along with algorithms in [2,9,4,3].

From Relational to XML: There have been different approaches for the conversion from relational model to XML

[⋆] This author is partially supported by DARPA contract No. N66001-97-C-8601.

[⋆⋆] This author is partially supported by NSF grants 0086116, 0085773, 9817773.

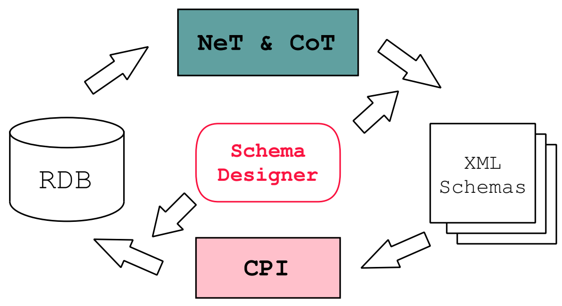


Fig. 1. Overview of our schema translation algorithms.

model, such as XML Extender from IBM, XML-DBMS, SilkRoute [11], and XPERANTO [5]. All the above tools require the user to specify the mapping from the given relational schema to XML schema. In XML Extender, the user specifies the mapping through a language such as DAD or XML Extender Transform Language. In XML-DBMS, a template-driven mapping language is provided to specify the mappings. SilkRoute provides a declarative query language (RXL) for viewing relational data in XML. XPERANTO uses XML query language for viewing relational data in XML. Note that in SilkRoute and XPERANTO, the user has to specify the query in the appropriate query language.

2 Overview of Our Schema Translation Algorithms

In this paper, we present three schema transformation algorithms that not only capture the structure, but also the semantics of the original schema. The overview of our proposals is illustrated in Figure 1.

1. CPI (Constraints-preserving Inlining Algorithm): identifies various semantics constraints in the original XML schema and preserves them by rewriting them in the final relational schema.
2. NeT (Nesting-based Translation Algorithm): derives a nested structure from a flat relational schema by repeatedly applying the $nest$ operator so that the resulting XML schema becomes hierarchical. The main idea is to find a more intuitive element content model of the XML schema that utilizes the regular expression operators provided by the XML schema specification (e.g., "*" or "+").
3. CoT (Constraints-based Translation Algorithm): Although NeT infers hidden characteristics of data by nesting, it is only applicable to a single table at a time. Therefore, it is unable to capture the overall picture of relational schema where multiple tables are interconnected. To remedy this problem, CoT considers inclusion dependencies during the translation, and merges multiple interconnected tables into a coherent and hierarchical parent-child structure in the final XML schema.

3 The CPI Algorithm

Transforming a hierarchical XML model to a flat relational model is not a trivial task due to several inherent difficulties such as non-trivial 1-to-1 mapping, existence of set values, complicated recursion, and/or fragmentation issues [4]. Most XML-to-Relational transformation algorithms (e.g., [9,2,3,4]) have so far mainly focused on the issue of structural conversion, largely ignoring the semantics already existed in the original XML schema. Let us first describe various semantic constraints that one can mine from the DTD. Throughout the discussion, we will use the example DTD and XML document in Tables 1 and 2.

```
<!ELEMENT conf    (title,date,editor?,paper*)>
<!ATTLIST conf    id     ID          #REQUIRED>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT date    EMPTY>
<!ATTLIST date    year   CDATA       #REQUIRED
                  mon    CDATA       #REQUIRED
                  day    CDATA       #IMPLIED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids   IDREFS      #IMPLIED>
<!ELEMENT paper   (title,contact?,author,cite?)>
<!ATTLIST paper   id     ID          #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid    IDREF       #REQUIRED>
<!ELEMENT author  (person+)>
<!ATTLIST author  id     ID          #REQUIRED>
<!ELEMENT person  (name,(email|phone)?)>
<!ATTLIST person  id     ID          #REQUIRED>
<!ELEMENT name    EMPTY>
<!ATTLIST name    fn     CDATA       #IMPLIED
                  ln     CDATA       #REQUIRED>
<!ELEMENT email   (#PCDATA)>
<!ELEMENT phone   (#PCDATA)>
<!ELEMENT cite    (paper*)>
<!ATTLIST cite    id     ID          #REQUIRED
                  format (ACM|IEEE)  #IMPLIED>
```

Table 1. A DTD for `Conference`.

3.1 Semantic Constraints in DTDs

Cardinality Constraints: In a DTD declaration, there are only 4 possible cardinality relationships between an element and its sub-elements as illustrated below:

```
<!ELEMENT article (title, author+,
                   ref*, price?)>
```

1. (0,1): An element can have either zero or one sub-element. (e.g., sub-element `price`)
2. (1,1): An element must have one and only one sub-element. (e.g., sub-element `title`)
3. (0,N): An element can have zero or more sub-elements. (e.g., sub-element `ref`)
4. (1,N): An element can have one or more sub-elements. (e.g., sub-element `author`)

Following the notations in [7], let us call each cardinality relationship as type (0,1), (1,1), (0,N), (1,N), respectively. From these cardinality relationships, mainly three constraints can be inferred. First is whether or not the sub-element can be null. We use the notation "$X \nrightarrow \emptyset$" to denote that an element X cannot be null. This constraint is easily enforced by the `NULL` or `NOT NULL` clause in SQL. Second is whether or not more than one sub-element can occur. This is also known as *singleton constraint* in [12] and is one kind of equality-generating dependencies. Third, given an element, whether or not its sub-element should occur. This is one kind of tuple-generating dependencies. The second and third types will be further discussed below.

Inclusion Dependencies (INDs): An *Inclusion Dependency* assures that values in the columns of one fragment must also

```
<conf id="er05">
  <title>Int'l Conf. on Conceptual Modeling</title>
  <date>
    <year>2005</year> <mon>May</mon> <day>20</day>
  </date>
  <editor eids="sheth bossy">
    <person id="klavans">
      <name fn="Judith" ln="Klavans"/>
      <email>klavans@cs.columbia.edu</email>
    </person>  </editor>
  <paper id="p1">
    <title>Indexing Model for Structured...</title>
    <contact aid="dao"/>
    <author>
      <person id="dao"><name fn="Tuong" ln="Dao"/>
    </author>
  </paper>
  <paper id="p2">
    <title>Logical Information Modeling...</title>
    <contact aid="shah"/>
    <author>
      <person id="shah">
        <name fn="Kshitij" ln="Shah"/>
      </person>
      <person id="sheth">
        <name fn="Amit" ln="Sheth"/>
        <email>amit@cs.uga.edu</email>
      </person>
    </author>
    <cite id="c100" format="ACM">
      <paper id="p3">
        <title>Making Sense of Scientific...</title>
        <author>
          <person id="bossy">
            <name fn="Marcia" ln="Bossy"/>
            <phone>391.4337</phone>
          </person>
        </author> </paper> </cite> </paper>
</conf>
<paper id="p7">
  <title>Constraints-preserving Trans...</title>
  <contact aid="lee"/>
  <author>
    <person id="lee">
      <name fn="Dongwon" ln="Lee"/>
      <email>dongwon@cs.ucla.edu</email>
    </person> </author>
  <cite id="c200" format="IEEE"/>
</paper>
...
```

Table 2. An example XML document conforming to the DTD in Table 1.

appear as values in the columns of other fragments and is a generalization of the notion of *referential integrity*.

Trivial form of INDs found in the DTD is that "given an element X and its sub-element Y, Y must be included in X (i.e., $Y \subseteq X$)". For instance, from the `conf` element and its four sub-elements in the `Conference` DTD, the following INDs can be found as long as `conf` is not null: $\{$`conf.title` $\subseteq$ `conf`, `conf.date` $\subseteq$ `conf`, `conf.editor` $\subseteq$ `conf`, `conf.paper` $\subseteq$ `conf`$\}$. Another form of INDs can be found in the attribute definition part of the DTD with the use of the `IDREF(S)` keyword. For instance, consider the `contact` and `editor` elements in the `Conference` DTD shown below:

```
<!ELEMENT person  (name,(email|phone)?>
<!ATTLIST person  id   ID     #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid  IDREF  #REQUIRED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids IDREFS #IMPLIED>
```

The DTD restricts the `aid` attribute of the `contact` element such that it can only point to the `id` attribute of the `person` element[1]. Further, the `eids` attribute can only point to multiple `id` attributes of the `person` element. As a result, the following INDs can be derived: $\{$`editor.eids` $\subseteq$ `person.id`, `contact.aid` $\subseteq$ `person.id`$\}$. Such INDs can be best enforced by the "foreign key" if the attribute being referenced is a primary key. Otherwise, it needs to use the `CHECK`, `ASSERTION`, or `TRIGGERS` facility of SQL.

[1] Precisely, an attribute with `IDREF` type does not specify which element it should point to. This information is available only by human experts. However, new XML schema languages such as XML-Schema and DSD can express where the reference actually points to [13].

Equality-Generating Dependencies (EGDs): The *Singleton Constraint* [12] restricts an element to have "at most" one sub-element. When an element type X satisfies the singleton constraint towards its sub-element type Y, if an element instance x of type X has *two* sub-elements instances y_1 and y_2 of type Y, then y_1 and y_2 must be the same. This property is known as *Equality-Generating Dependencies (EGDs)* and denoted by "$X \rightarrow Y$" in database theory. For instance, two EGDs: $\{$`conf` $\rightarrow$ `conf.title`, `conf` $\rightarrow$ `conf.date`$\}$ can be derived from the `conf` element in Table 1. This kind of EGDs can be enforced by SQL `UNIQUE` construct. In general, EGDs occur in the case of the (0,1) and (1,1) mappings in the cardinality constraints.

Tuple-Generating Dependencies (TGDs): TGDs in a relational model require that some tuples of a certain form be present in the table and use the "$\twoheadrightarrow$" symbol. Two useful forms of TGDs from DTD are the *child* and *parent constraints* [12].

1. **Child constraint:** `"Parent` $\twoheadrightarrow$ `Child"` states that every element of type $Parent$ must have at least one child element of type $Child$. This is the case of the (1,1) and (1,N) mappings in the cardinality constraints. For instance, from the DTD in Table 1, because the `conf` element must contain the `title` and `date` sub-elements, the child constraint `conf` $\twoheadrightarrow$ $\{$`title, date`$\}$ holds.
2. **Parent constraint:** `"Child` $\twoheadrightarrow$ `Parent"` states that every element of type $Child$ must have a parent element of type $Parent$. According to XML specification, XML documents can start from any level of element without necessarily specifying its parent element, when a root element is not specified by `<!DOCTYPE root>`. In the DTD in Table 1, for instance, the `editor` and `date` elements can have the `conf` element as their parent. Further, if we know that all XML documents were started at the `conf` element level, rather than the `editor` or `date` level, then the parent constraint $\{$`editor, date`$\}$ $\twoheadrightarrow$ `conf` holds. Note that the `title` $\twoheadrightarrow$ `conf` does not hold since the `title` element can be a sub-element of either the `conf` or `paper` element.

3.2 Discovering and Preserving Semantic Constraints from DTDs

The CPI algorithm utilizes a structure-based conversion algorithm as a basis and identifies various semantic constraints described in Section 3.1. We will use the *hybrid* algorithm [4] as the basis algorithm. CPI first constructs a *DTD graph* that represents the structure of a given DTD. A DTD graph can be constructed when parsing the given DTD. Its nodes are elements, attributes, or operators in the DTD. Each element appears exactly once in the graph, while attributes and operators appear as many times as they appear in the DTD. CPI then annotates various cardinality relationships (summarized in Table 3) among nodes to each edge of the DTD graph. Note that the cardinality relationship types in the graph consider not only element vs. sub-element relationships but also element vs. attribute relationships. Figure 2 illustrates an example of such annotated DTD graph for the `Conference` DTD in Table 1.

Relationship	Symbol	not null	EGDs	TGDs
(0,1)	?	no	yes	no
(1,1)		yes	yes	yes
(0,N)	*	no	no	no
(1,N)	+	yes	no	yes

Table 3. Cardinality relationships and their corresponding semantic constraints.

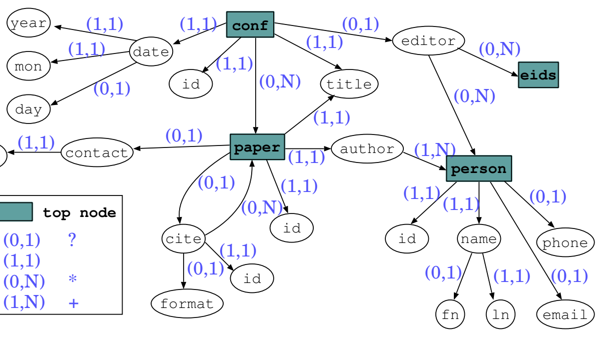


Fig. 2. An annotated *DTD graph* for the `Conference` DTD in Table 1.

Once the annotated DTD graph is constructed, CPI follows the basic navigation method provided by the *hybrid* algorithm; it identifies **top nodes** [4,10] that are the nodes: 1) not reachable from any nodes (e.g., source node), 2) direct child of "*" or "+" operator node, 3) recursive node with indegree > 1, or 4) one node between two mutually recursive nodes with indegree $= 1$. Then, starting from each top node T, *inline* all the elements and attributes at *leaf nodes* reachable from T unless they are other top nodes. In doing so, each annotated cardinality relationship can be properly converted to its counterpart in SQL syntax as described in Section 3.1. The details of the algorithm is beyond the scope of this paper and interested readers are referred to [10]. For instance, Figure 3 and Table 4 are such output relational schema and data in SQL notation, automatically generated by the CPI algorithm.

4 The NeT Algorithm

The simplest Relational-to-XML translation method, termed as FT (Flat Translation) in [14], is to translate 1) tables in a relational schema to elements in an XML schema and 2) columns in a relational schema to attributes in an XML schema. FT is a simple and effective translation algorithm. However, since FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner, it does not utilize several basic "non-flat" features provided by the XML model for data modeling such as representing *repeating subelements* through regular expression operators (e.g., "*", "+"). To remedy the shortcomings of FT, we propose the NeT algorithm that utilizes various *element content models* of the XML model. NeT uses the *nest* operator [15] to derive a "good" element content model.

Informally, for a table t with a set of columns C, *nesting* on a non-empty column $X \in C$ collects all tuples that agree on the remaining columns $C - X$ into a set[2]. Formally,

[2] Here, we only consider single attribute nesting.

```
CREATE TABLE paper (
  id           NUMBER      NOT NULL,
  title        VARCHAR(50) NOT NULL,
  contact_aid  VARCHAR(20),
  cite_id      VARCHAR(20),
  cite_format  VARCHAR(50)
    CHECK (VALUE IN ("ACM", "IEEE")),
  root_elm     VARCHAR(20) NOT NULL,
  parent_elm   VARCHAR(20),
  fk_cite      VARCHAR(20)
    CHECK (fk_cite IN
      (SELECT cite_id FROM paper)),
  fk_conf      VARCHAR(20),
  PRIMARY KEY (id),
  UNIQUE (cite_id),
  FOREIGN KEY (fk_conf)
    REFERENCES conf(id),
  FOREIGN KEY (contact_aid)
    REFERENCES person(id)
);
```

Fig. 3. Final relational "schema" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

Definition 1 (Nest). *[15]. Let t be a n-ary table with column set C, and $X \in C$ and $\overline{X} = C - X$. For each $(n-1)$-tuple $\gamma \in \Pi_{\overline{X}}(t)$, we define an n-tuple γ^* as follows: $\gamma^*[\overline{X}] = \gamma$, and $\gamma^*[X] = \{\kappa[X] \mid \kappa \in t \wedge \kappa[\overline{X}] = \gamma$. Then, $nest_X(t) = \{\gamma^* \mid \gamma \in \Pi_{\overline{X}}(t)\}$.*

After $nest_X(t)$, if column X has only a set with "single" value $\{v\}$ for all the tuples, then we say that **nesting failed** and we treat $\{v\}$ and v interchangeably (i.e., $\{v\}$ = v). Thus when nesting failed, the following is true: $nest_X(t) = t$. Otherwise, if column X has a set with "multiple" values $\{v_1, ..., v_k\}$ with $k \geq 2$ for at least one tuple, then we say that **nesting succeeded**.

Example 2. Consider a table R in Table 5. Here we assume that the columns A, B, C are non-nullable. In computing $nest_A(R)$ at (b), the first, third, and fourth tuples of R agree on their values in columns (B, C) as (a, 10), while their values of the column A are all different. Therefore, these different values are grouped (i.e., nested) into a set $\{$1,2,3$\}$. The result is the first tuple of the table $nest_A(R)$ – ($\{$1,2,3$\}$, a, 10). Similarly, since the sixth and seventh tuples of R agree on their values as (b, 20), they are grouped to a set $\{$4,5$\}$. In computing $nest_B(R)$ at (c), there are no tuples in R that agree on the values of the columns (A, C). Therefore, $nest_B(R) = R$. In computing $nest_C(R)$ at (d), since the first two tuples of R – (1, a, 10) and (1, a, 20) – agree on the values of the columns (A, B), they are grouped to (1, a, $\{$10,20$\}$). Nested tables (e) through (j) are constructed similarly.

Since the $nest$ operator requires scanning of the entire set of tuples in a given table, it can be quite expensive. In addition, as shown in Example 2, there are various ways to nest the given table. Therefore, it is important to find an efficient way (that uses the $nest$ operator minimum number of times) of obtaining an acceptable element content model. For a detailed description on the various properties of the $nest$ operator, the interested are referred to [14,16].

paper								
id	root_elm	parent_elm	fk_conf	fk_cite	title	contact_aid	cite_id	cite_format
p1	conf	conf	er05	–	Indexing ...	dao	–	–
p2	conf	conf	er05	–	Logical ...	shah	c100	ACM
p3	conf	cite	–	c100	Making ...	–	–	–
p7	paper	–	–	–	Constraints ...	lee	c200	IEEE

Table 4. Final relational "data" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

	A	B	C
#1	1	a	10
#2	1	a	20
#3	2	a	10
#4	3	a	10
#5	4	b	10
#6	4	b	20
#7	5	b	20

(a) R

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(b) $nest_A(R)$

A	B	C
1	a	10
1	a	20
2	a	10
3	a	10
4	b	10
4	b	20
5	b	20

(c) $nest_B(R) = R$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(d) $nest_C(R)$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(e) $nest_B(nest_A(R)) = nest_C(nest_A(R))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(f) $nest_A(nest_C(R))$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(g) $nest_B(nest_C(R))$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(h) $nest_C(nest_B(nest_A(R))) = nest_B(nest_C(nest_A(R)))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(i) $nest_B(nest_A(nest_C(R))) = nest_A(nest_B(nest_C(R)))$

Table 5. A relational table R and its various nested forms. Column names containing a set after nesting (i.e., nesting succeeded) are appended by "+" symbol.

Lemma 1. *Consider a table t with column set C, candidate keys, $K_1, K_2, \ldots, K_n \subseteq C$, and column set K such that $K = K_1 \cap K_2 \cap \ldots \cap K_n$. Further, let $|C| = n$ and $|K| = m$ ($n \geq m$). Then, the number of necessary nestings, N, is bounded by $N \leq \sum_{k=1}^{m} m^{\underline{k}}$*

Lemma 1 implies that when candidate key information is available, one can avoid unnecessary nestings substantially. For instance, suppose attributes A and C in Table 5 constitute a key for R. Then, one needs to compute only: $nest_A(R)$ at (b), $nest_C(R)$ at (d), $nest_C(nest_A(R))$ at (e), $nest_A(nest_C(R))$ at (f) in Table 5.

After applying the $nest$ operator to the given table repeatedly, there can be still several nested tables where nesting succeeded. In general, the choice of the final schema should take into consideration the semantics and usages of the underlying data or application and this is where user intervention is beneficial. By default, without further input from users, NeT chooses the nested table where the most number of nestings succeeded as the final schema, since this is a schema which provides low "data redundancy". The outline of the NeT algorithm is as follows:

1. For each table t_i in the input relational schema $\mathbb{R}$, apply the $nest$ operator repeatedly until no nesting succeeds.
2. Choose the best nested table based on the selected criteria. Denote this table as $t'_i(c_1, \ldots, c_{k-1}, c_k, \ldots, c_n)$, where nesting succeeded on the columns $\{c_1, \ldots, c_{k-1}\}$.
   (a) If $k = 1$, follow the FT translation.
   (b) If $k > 1$,
      i. For each column c_i ($1 \leq i \leq k-1$), if c_i was nullable in $\mathbb{R}$, use c_i^* for the element content model, and c_i^+ otherwise.
      ii. For each column c_j ($k \leq j \leq n$), if c_i was nullable in $\mathbb{R}$, use $c_j^?$ for the element content model, and c_j otherwise.

5 The CoT Algorithm

The NeT algorithm is useful for decreasing data redundancy and obtaining a more intuitive schema by 1) removing redundancies caused by multivalued dependencies, and 2) performing grouping on attributes. However, NeT considers tables one at a time, and cannot obtain a *overall picture* of the relational schema where many tables are interconnected with each other through various other dependencies. To remedy this problem, we propose the CoT algorithm that uses Inclusion Dependencies (INDs) of relational schema. General forms of INDs are difficult to acquire from the database automatically. However, we shall consider the most pervasive form of INDs, foreign key constraints, which can be queried through ODBC/JDBC interface.

The basic idea of the CoT is the following: For two distinct tables s and t with lists of columns X and Y, respectively, suppose we have a foreign key constraint $s[\alpha] \subseteq t[\beta]$, where $\alpha \subseteq X$ and $\beta \subseteq Y$. Also suppose that $K_s \subseteq X$ is the key for s. Then, different cardinality binary relationships between s and t can be expressed in the relational model by a combination of the following: 1) α is unique/not-unique, and 2) α is nullable/non-nullable. Then, the translation of two tables s, t with a foreign key constraint works as follows:

1. If α is non-nullable (i.e., none of the columns of α can take null values), then:

`student(Sid, Name, Advisor)`
`emp(Eid, Name, ProjName)`
`prof(Eid, Name, Teach)`
`course(Cid, Title, Room)`
`dept(Dno, Mgr)`
`proj(Pname, Pmgr)`
`student(Advisor)` $\subseteq$ `prof(Eid)`
`emp(ProjName)` $\subseteq$ `proj(Pname)`
`prof(Teach)` $\subseteq$ `course(Cid)`
`prof(Eid, Name)` $\subseteq$ `emp(Eid, Name)`
`dept(Mgr)` $\subseteq$ `emp(Eid)`
`proj(Pmgr)` $\subseteq$ `emp(Eid)`

Table 6. An example schema with associated INDs.

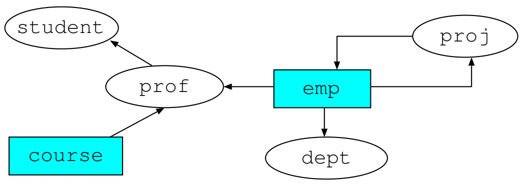


Fig. 4. The IND-Graph representation of the schema in Table 6 (*top nodes* denoted by rectangular nodes).

(a) If α is unique, then there is a $1:1$ relationship between s and t, and can be captured as `<!ELEMENT t (Y, s?)>`.
(b) If α is not-unique, then there is a $1:n$ relationship between s and t, and can be captured as `<!ELEMENT t (Y, s*)>`.

2. If s is represented as a sub-element of t, then the key for s will change from K_s to $(K_s - \alpha)$. The key for t will remain the same.

Extending this to the general case where multiple tables are interconnected via INDs, consider the schema with a set of tables $\{t_1, ..., t_n\}$ and INDs $t_i[\alpha_i] \subseteq t_j[\beta_j]$, where $i, j \leq n$. We consider only those INDs that are foreign key constraints (i.e., β_j constitutes the primary key of the table t_j), and where α_i is non-nullable. The relationships among tables can be captured by a graph representation, termed as IND-Graph.

Definition 2 (IND-Graph). *An* IND-Graph $G = (V, E)$ *consists of a node set V and a directed edge set E, such that for each table t_i, there exists a node $V_i \in V$, and for each distinct IND $t_i[\alpha] \subseteq t_j[\beta]$, there exists an edge $E_{ji} \in E$ from the node V_j to V_i.*

Note the edge direction is reversed from the IND direction for convenience. Given a set of INDs, the IND-Graph can be easily constructed. Once an IND-Graph G is constructed, CoT needs to decide the starting point to apply translation rules. For that purpose, we use the notion of **top nodes**. Intuitively, an element is a top node if it *cannot* be represented as a sub-element of any other element. Let T denote the set of top nodes. Then, CoT traverses G, using say Breadth-First Search (BFS), until it traverses all the nodes and edges, while capturing the INDs on edges as either sub-elements (when the node is visited for the first time) or IDREF attributes (when the node was visited already).

Example 3. Consider a schema and its associated INDs in Table 6. The IND-Graph with two top nodes is shown in Figure 4: 1) `course`: There is no node t, where there is an IND of the form `course`$[\alpha] \subseteq t[\beta]$, and 2) `emp`: There is a cyclic set of INDs between `emp` and `proj`, and there exists no node t such that there is an IND of the form `emp`$[\alpha] \subseteq t[\beta]$ or `proj`$[\alpha] \subseteq t[\beta]$. Then,

– First, starting from a top node `course`, do BFS scan. Pull up a reachable node `prof` into `course` and make it as sub-element by `<!ELEMENT course (Cid, Title, Room, prof`*`)>`. Similarly, the node `student` is also pulled up into its parent node `prof` by `<!ELEMENT prof (Eid, Name, student`*`)>`. Since the node `student` is a leaf, no nodes can be pulled in: `<!ELEMENT student (Sid, Name)>`. Since there is no more unvisited reachable node from `course`, the scan stops.
– Next, starting from another top node `emp`, pull up neighboring node `dept` into `emp` similarly by `<!ELEMENT emp (Eid, Name, ProjName, dept`*`)>` and `<!ELEMENT dept (Dno, Mgr)>`. Then, visit a neighboring node `prof`, but `prof` was visited already. To avoid data redundancy, an attribute `Ref_prof` is added to `emp` accordingly. Since attributes in the left-hand side of the corresponding IND, $prof(Eid, Name) \subseteq emp(Eid, Name)$, form a super key, the attribute `Ref_prof` is assigned type `IDREF`, and not `IDREFS`: `<!ATTLIST prof Eid ID>` and `<!ATTLIST emp Ref_prof IDREF>`.
– Next, visit a node `proj` and pull it up to `emp` by `<!ELEMENT emp (Eid, Name, ProjName, dept`*`, proj`*`)>` and `<!ELEMENT proj (Pname)>`. In next step, visit a node `emp` from `prof`, but since it was already visited, an attribute `Ref_emp` of type `IDREFS` is added to `proj`, and scan stops.

It is worthwhile to point out that there are several places in CoT where human experts can help to find a better mapping based on the semantics and usages of the underlying data or application.

6 Experimental Results

6.1 CPI Results

CPI was tested against DTDs gathered from OASIS[3]. For all cases, CPI successfully identified hidden semantic constraints from DTDs and correctly preserved them by rewriting them in SQL. Table 7 shows a summary of our experimentation. Note that people seldom used the `ID` and `IDREF(S)` constructs in their DTDs except the `XMI` and `BSML` cases. The number of tables generated in the relational schema was usually smaller than that of elements/attributes in DTDs due to the inlining effect. The only exception to this phenomenon was the `XMI` case, where extensive use of types (0,N) and (1,N) cardinality relationships resulted in many top nodes in the ADG.

The number of semantic constraints had a close relationship with the design of the DTD hierarchy and the type of

[3] http://www.oasis-open.org/cover/xml.html

DTD Semantics		DTD Schema		Relational Schema			
Name	Domain	Elm/Attr	ID/IDREF(S)	Table/Attr	$\rightarrow$	$\twoheadrightarrow$	$\nrightarrow$ $\emptyset$
`novel`	literature	10/1	1/0	5/13	6	9	9
`play`	Shakespeare	21/0	0/0	14/46	17	30	30
`tstmt`	religious text	28/0	0/0	17/52	17	22	22
`vCard`	business card	23/1	0/0	8/19	18	13	13
`ICE`	content synd.	47/157	0/0	27/283	43	60	60
`MusicML`	music desc.	12/17	0/0	8/34	9	12	12
`OSD`	s/w desc.	16/15	0/0	15/37	2	2	2
`PML`	web portal	46/293	0/0	41/355	29	36	36
`Xbel`	bookmark	9/13	3/1	9/36	9	1	1
`XMI`	metadata	94/633	31/102	129/3013	10	7	7
`BSML`	DNA seq.	112/2495	84/97	104/2685	99	33	33

Table 7. Summary of CPI algorithm.

cardinality relationship used in the DTD. For instance, the XMI DTD had many type (0,N) cardinality relationships, which do not contribute to the semantic constraints. As a result, the number of semantic constraints at the end was small, compared to that of elements/attributes in the DTD. This was also true for the OSD case. On the other hand, in the ICE case, since it used many type (1,1) cardinality relationships, it resulted in many semantic constraints.

6.2 NeT Results

Our preliminary results comparing the goodness of the *XSchema* obtained from NeT and FT with that obtained from DB2XML v 1.3 [17] appeared in [14]. We further applied our NeT algorithm on several test sets drawn from UCI KDD[4] / ML[5] repositories, which contain a multitude of single-table relational schemas and data. Sample results are shown in Table 8. Two metrics are used as follows:

$$\text{NestRatio} = \frac{\text{\# of successful nesting}}{\text{\# of total nesting}}$$
$$\text{ValueRatio} = \frac{\text{\# of original data values}}{\text{\# of data values D in the nested table}}$$

where D is the number of individual data values present in the table. For example, the D in the row $(\{1, 2, 3\}, a, 10)$ of a nested table is 5. High value for *NestRatio* shows that we did not perform unnecessary nesting and high value for *ValueRatio* shows that the nesting removed a lot of redundancy.

In our experimentation[6], we observed that most of the attempted nestings are successful, and hence our optimization rules are quite efficient. In Table 8, we see that nesting was useful for all the data sets except for the Bupa data set. Also nesting was *especially* useful for the Car data set, where the size of the nested table is only 6% of the original data set. Time required for nesting is an important parameter, and it jointly depends on the number of attempted nestings and the number of tuples. The number of attempted nestings depends on the number of attributes, and increases drastically as the number of attributes increases. This is observed for the Flare data set, where we have to do nesting on 13 attributes.

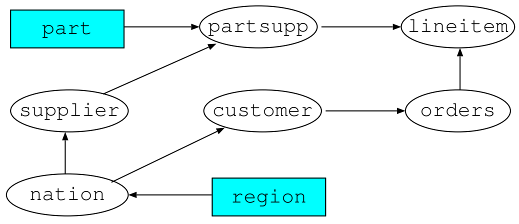


Fig. 5. The IND-Graph representation of TPC-H schema.

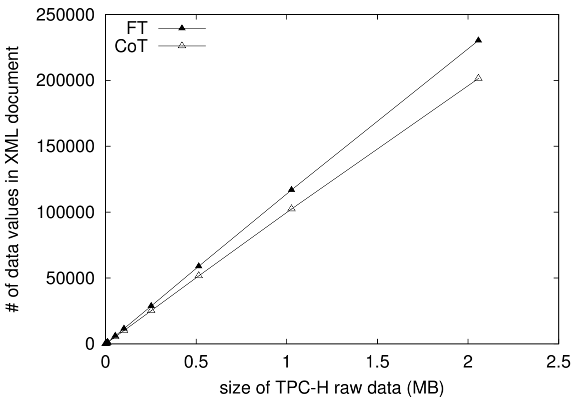


Fig. 6. Size comparison of two algorithms.

6.3 CoT Results

For testing CoT, we need some well-designed relational schema where tables are interconnected via inclusion dependencies. For this purpose, we use the TPC-H schema v 1.3.0[7], which is an ad-hoc, decision support benchmark and has 8 tables and 8 inclusion dependencies. The IND-Graph for the TPC-H schema is shown in Figure 5. CoT identified two top-nodes – `part` and `region`, and eventually generated the XML document having interwoven hierarchical structures; six of the eight inclusion dependencies are mapped using sub-element, and the remaining two are mapped using IDREF attributes.

Figure 6 shows a comparison of the number of data values originally present in the database, and the number of data

[4] http://kdd.ics.uci.edu/
[5] http://www.ics.uci.edu/∼mlearn/MLRepository.html
[6] Available at http://www.cs.ucla.edu/∼mani/xml
[7] http://www.tpc.org/tpch/spec/h130.pdf

Test Set	# of attr. / tuple	NestRatio	ValueRatio	Size before / after	# of nested attr.	Time (sec.)
`Balloons1`	5 / 16	42 / 64	80 / 22	0.455 / 0.152	3	1.08
`Balloons2`	5 / 16	42 / 64	80 / 22	0.455 / 0.150	3	1.07
`Balloons3`	5 / 16	40 / 64	80 / 42	0.455 / 0.260	3	1.14
`Balloons4`	5 / 16	42 / 64	80 / 22	0.455 / 0.149	3	1.07
`Hayes`	6 / 132	1 / 6	792 / 522	1.758 / 1.219	1	1.01
`Bupa`	7 / 345	0 / 7	2387 / 2387	7.234 / 7.234	0	4.40
`Balance`	5 / 625	56 / 65	3125 / 1120	6.265 / 2.259	4	21.48
`TA_Eval`	6 / 110	253 / 326	660 / 534	1.559 / 1.281	5	24.83
`Car`	7 / 1728	1870 / 1957	12096 / 779	51.867 / 3.157	6	469.47
`Flare`	13 / 365	11651 / 13345	4745 / 2834	9.533 / 5.715	4	6693.41

Table 8. Summary of NeT experimentations.

values in the XML document generated by FT and CoT. Because FT is a flat translation, the number of data values in the XML document generated by FT is the same as the number of data values in the original data. However, CoT is able to decrease the number of data values in the generated XML document by more than 12%.

7 Conclusion

We have presented a method to transform a relational schema to an XML schema, and two methods to transform an XML schema to a relational schema, both in *structural* and *semantic* aspects. All three algorithms are "correct" in the sense that they all have preserved the original information of relational schema. For instance, using the notion of information capacity [18], a theoretical analysis for the correctness of our translation procedures is possible; we can actually show that CPI, NeT and CoT algorithms are *equivalence preserving transformations*.

Despite the difficulties in conversions between XML and relational models, there are many practical benefits. We strongly believe that devising more accurate and efficient conversion methodologies between XML and relational models is important. The prototypes of our algorithms are available at: http://www.cobase.cs.ucla.edu/projects/xpress/

References

1. Bray, T., Paoli, J., Sperberg-McQueen (Eds), C.M.: "Extensible Markup Language (XML) 1.0 (2nd Edition)". W3C Recommendation (2000) http://www.w3.org/TR/2000/REC-xml-20001006.
2. Deutsch, A., Fernandez, M.F., Suciu, D.: "Storing Semistructured Data with STORED". In: ACM SIGMOD, Philadephia, PA (1998)
3. Florescu, D., Kossmann, D.: "Storing and Querying XML Data Using an RDBMS". IEEE Data Eng. Bulletin **22** (1999) 27–34
4. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: "Relational Databases for Querying XML Documents: Limitations and Opportunities". In: VLDB, Edinburgh, Scotland (1999)
5. Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., Subramanian, S.: "XPERANTO: Publishing Object-Relational Data as XML". In: Int'l Workshop on the Web and Databases (WebDB), Dallas, TX (2000)
6. Christophides, V., Abiteboul, S., Cluet, S., Scholl, M.: "From Structured Document to Novel Query Facilities". In: ACM SIGMOD, Minneapolis, MN (1994)
7. Batini, C., Ceri, S., Navathe, S.B.: "Conceptual Database Design: An Entity-Relationship Approach". The Benjamin/Cummings Pub. (1992)
8. Bernstein, P., Halevy, A., Pottinger, R.: "A Vision of Management of Complex Models ". ACM SIGMOD Record **29** (2000) 55–63
9. Bourret, R.: "XML and Databases". Web page (1999) http://www.rpbourret.com/xml/XMLAndDatabases.htm.
10. Lee, D., Chu, W.W.: "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". J. Data & Knowledge Engineering (DKE) **39** (2001) 3–25
11. Fernandez, M.F., Tan, W.C., Suciu, D.: "SilkRoute: Trading between Relations and XML". In: Int'l World Wide Web Conf. (WWW), Amsterdam, Netherlands (2000)
12. Wood, P.T.: "Optimizing Web Queries Using Document Type Definitions". In: Int'l Workshop on Web Information and Data Management (WIDM), Kansas City, MO (1999) 28–32
13. Lee, D., Chu, W.W.: "Comparative Analysis of Six XML Schema Languages". ACM SIGMOD Record **29** (2000) 76–87
14. Lee, D., Mani, M., Chiu, F., Chu, W.W.: "Nesting-based Relational-to-XML Schema Translation". In: Int'l Workshop on the Web and Databases (WebDB), Santa Barbara, CA (2001)
15. Jaeschke, G., Schek, H.J.: "Remarks on the Algebra of Non First Normal Form Relations". In: ACM PODS, Los Angeles, CA (1982)
16. Lee, D., Mani, M., Chiu, F., Chu, W.W.: "NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints". Technical report, UCLA Computer Science Dept. (2002)
17. Turau, V.: "Making Legacy Data Accessible for XML Applications". Web page (1999) http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html.
18. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: "Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice (Extended Abstract)". In: EDBT, Cambridge, UK (1994)

Transforming UML domain descriptions into Configuration Knowledge Bases for the Semantic Web

Alexander Felfernig[1], Gerhard Friedrich[1], Dietmar Jannach[1], Markus Stumptner[2], and Markus Zanker[1]

[1] Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria,
{felfernig,friedrich,jannach,zanker}@ifit.uni-klu.ac.at
[2] University of South Australia, Advanced Computing Research Centre, 5095 Mawson Lakes (Adelaide), SA, Australia
mst@cs.unisa.edu.au

Abstract. The Semantic Web will provide the conceptual infrastructure to allow new forms of business application integration. This paper presents the theoretical basis for integrating Web-based sales systems for highly complex customizable products and services (configuration systems) making use of upcoming descriptive representation formalisms for the Semantic Web. In today's economy a trend evolves towards highly specialized solution providers cooperatively offering configurable products and services to their customers. This paradigm shift requires the extension of current *standalone* configuration technology with capabilities for cooperative problem solving. Communication between product configurators, however, necessitates the existence of an agreed upon definition of the configuration problem itself and the sharing of knowledge. A standardized representation language is therefore needed in order to tackle the challenges imposed by heterogeneous representation formalisms of currently available state-of-the-art configuration environments (e.g. description logic or predicate logic based configurators). Furthermore, it is important to integrate the development and maintenance of configuration systems into industrial software development processes. Therefore, we present a set of rules for transforming UML models (built conforming to a configuration domain specific profile) into configuration knowledge bases specified by languages such as OIL or DAML+OIL which become the future standards for the representation of semantics in the Web.

1 Introduction

Configuration is one of the most successful AI application areas, but easy knowledge acquisition and use of an appropriate set of modeling primitives remain major research areas. Increasing demand for applications in various domains such as telecommunications industry, automotive industry, or financial services can be noticed that results in a set of corresponding configurator implementations (e.g. [1,2,3,4]). Informally, configuration can be seen as a special kind of design activity [5], where the configured product is built from a predefined set of component types and attributes, which are composed conforming to a set of corresponding constraints.

Triggered by the trend towards highly specialized solution providers cooperatively offering configurable products and services, joint configuration by a set of business partners is becoming a key application of knowledge-based configuration systems. However, when it comes to integration issues of the configuration systems of different business entities, the heterogeneity of configuration knowledge representation is a major obstacle. One of the guiding application scenarios of the EC-funded research project CAWICOMS[3] is for example the provision of highly complex IP-VPN (IP-protocol based virtual private network) services by a dynamically formed consortium of telecommunication companies [6]. To perform such a configuration task, where the required knowledge is distributed over a flexibly determined set of separate entities, the paradigm of Web services is adopted to accomplish this form of business application integration [7]. In order to realize a dynamic matchmaking between service requestors and service providers, a brokering main configurator determines which configuration services of the participating organisations are capable of contributing to the problem solving and cooperates with them. Currently developed declarative languages (e.g., DAML-S[4]) for semantically describing the capabilities of a Web-service are based on DAML+OIL, that is why we show how the concepts needed for describing configuration knowledge can be represented using semantic markup languages such as OIL [8] or DAML+OIL [9].

From the viewpoint of industrial software development, the integration of construction and maintenance of knowledge-based systems is an important prerequisite for a broader application of AI technologies. When considering configuration systems, formal knowledge representation languages are difficult to communicate to domain experts. The so-called knowledge acquisition bottleneck is obvious, since configuration knowledge acquisition and maintenance are only feasible with the support of a knowledge engineer who can handle the formal representation language of the underlying configuration system.

The Unified Modeling Language (UML) [10] is a widely adopted modeling language in industrial software development. Based on our experience in building configuration knowledge bases using UML [11], we show how to effectively support the construction of Semantic Web configuration knowledge bases using UML as a knowledge acquisition frontend. The provided UML concepts constitute an ontology consisting of concepts contained in de facto standard configuration ontologies [11,12]. Based on a descrip-

[3] CAWICOMS is the acronym for Customer-Adaptive Web Interface for the Configuration of products and services with Multiple Suppliers (EC-funded project IST-1999-10688).

[4] See http://www.daml.org/services for reference.

tion logic based definition of a configuration task we provide a set of rules for automatically transforming UML configuration models into a corresponding OIL representation[5].

The approach presented in this paper permits the use of standard design notations by Software Engineers, vice versa, reasoning support for Semantic Web ontology languages can be exploited for checking the consistency of the UML configuration models. The resulting configuration knowledge bases enable knowledge interchange between heterogenous configuration environments as well as distributed configuration problem solving in different supply chain settings. The presented concepts are implemented in a knowledge acquisition workbench which is a major part of the CAWICOMS configuration environment.

The paper is organized as follows. In Section 2 we give an example of a UML configuration knowledge base which is used for demonstration purposes throughout the paper. In Section 3 we give a description logic based definition of a configuration task - this definition serves as basis for the translation of UML configuration models into a corresponding OIL-based representation (Section 4). Section 5 discusses related work.

2 Configuration knowledge base in UML

The Unified Modeling Language (UML) [10] is the result of an integration of object-oriented approaches of [13,14,15] which is well established in industrial software development. UML is applicable throughout the whole software development process from the requirements analysis phase to the implementation phase. In order to allow the refinement of the basic meta-model with domain-specific modeling concepts, UML provides the concept of *profiles* - the configuration domain specific modeling concepts presented in the following are the constituting elements of a UML *configuration profile* which can be used for building configuration models. UML profiles can be compared with ontologies discussed in the AI literature, e.g. [16] defines an ontology as a theory about the sorts of objects, properties of objects, and relationships between objects that are possible in a specific domain. UML *stereotypes* are used to further classify UML meta-model elements (e.g. classes, associations, dependencies). Stereotypes are the basic means to define domain-specific modeling concepts for profiles (e.g. for the configuration profile). In the following we present a set of rules allowing the automatic translation of UML configuration models into a corresponding OIL representation.

For the following discussions the simple UML configuration model shown in Figure 1 will serve as a working example. This model represents the generic product structure, i.e. all possible variants of a configurable $Computer$. The basic structure of the product is modeled using classes, generalization, and aggregation. The set of possible products is restricted through a set of constraints which are related to technical restrictions, economic factors, and restrictions according to the production process. The used concepts stem from connection-based [17], resource-based [3], and structure-based [18] configuration approaches. These configuration domain-specific concepts represent a basic set useful for building configuration knowledge bases and mainly correspond to those defined in the de facto standard configuration ontologies [11,12]:

Component types. Component types represent the basic building blocks a final product can be built of. Component types are characterized by attributes. A stereotype *Component* is introduced, since some limitations on this special form of class must hold (e.g. there are no methods).

Generalization hierarchies. Component types with a similar structure are arranged in a generalization hierarchy (e.g. in Figure 1 a *CPU1* is a special kind of *CPU*).

Part-whole relationships. Part-whole relationships between component types state a range of how many subparts an aggregate can consist of (e.g. a *Computer* contains at least one and at most two motherboards - *MBs*).

Compatibilities and requirements. Some types of components must not be used together within the same configuration - they are incompatible (e.g. an *SCSIUnit* is incompatible with an *MB1*). In other cases, the existence of one component of a specific type requires the existence of another specific component within the configuration (e.g an *IDEUnit* requires an *MB1*). The compatibility between different component types is expressed using the stereotyped association *incompatible*. Requirement constraints between component types are expressed using the stereotype *requires*.

Resource constraints. Parts of a configuration task can be seen as a resource balancing task, where some of the component types produce some resources and others are consumers (e.g., the consumed hard-disk capacity must not exceed the provided hard-disk capacity). Resources are described by a stereotype *Resource*, furthermore stereotyped dependencies are introduced for representing the producer/consumer relationships between different component types. Producing component types are related to resources using the *produces* dependency, furthermore consuming component types are related to resources using the *consumes* dependency. These dependencies are annotated with values representing the amount of production and consumption.

Port connections. In some cases the product topology - i.e., exactly how the components are interconnected - is of interest in the final configuration. The concept of a port (stereotype *Port*) is used for this purpose (e.g. see the connection between *Videocard* and *Screen* represented by the stereotype *conn* and the ports $videoport$ and $screenport$).

3 Description logic based definition of a configuration task

The following description logic based definition of a configuration task [19] serves as a foundation for the formulation of rules for translating UML configuration models into a corresponding OIL representation[6]. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [20], where $\mathcal{RN}$ is a disjunctive union of roles and features.

[5] Note that OIL text is used for presentation purposes - the used concepts can simply be transformed into a DAML+OIL representation.

[6] In the following we assume that the reader is familiar with the concepts of OIL. See [8] for an introductory text.

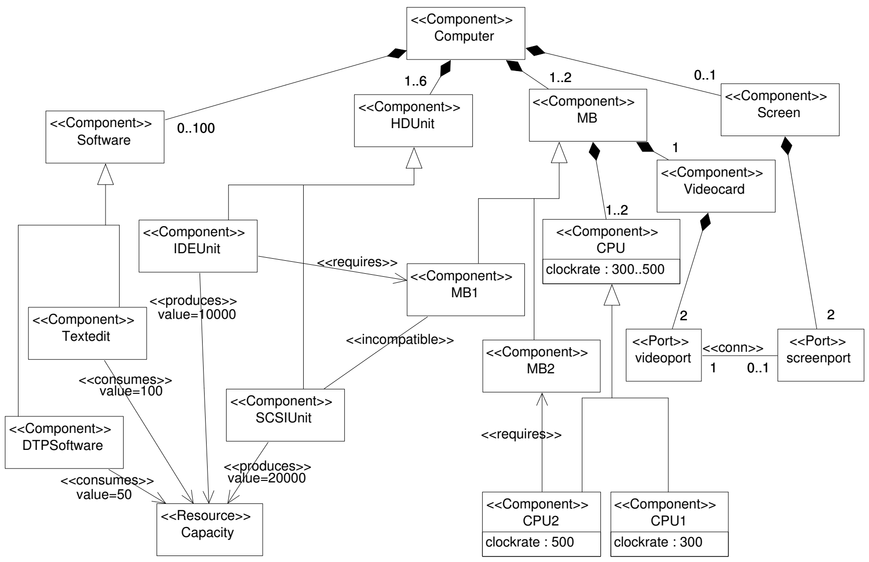


Fig. 1. Example configuration model

Definition 1 (Configuration task): In general we assume a configuration task is described by a triple (DD, SRS, $CLANG$). DD represents the domain description of the configurable product and SRS specifies the particular system requirements defining an individual configuration task instance. $CLANG$ comprises a set of concepts $C_{Config} \subseteq \mathcal{CN}$ and a set of roles $R_{Config} \subseteq \mathcal{RN}$ which serve as a configuration language for the description of actual configurations. A configuration knowledge base KB = $DD \cup SRS$ is constituted of sentences in a description language. □

In addition we require that roles in $CLANG$ are defined over the domains given in C_{Config}, i.e. $range(R_i)$ = $CDom$ and $dom(R_i)$ = $CDom$ must hold for each role $R_i \in R_{Config}$, where $CDom \doteq \bigsqcup_{C_i \in C_{config}} C_i$. We impose this restriction in order to assure that a configuration result only contains individuals and relations with corresponding definitions in C_{Config} and R_{Config}. The derivation of DD will be discussed in Section 4, an example for SRS could be "two $CPUs$ of type $CPU1$ and one CPU of type $CPU2$", i.e. SRS={(instance-of $c1$, $CPU1$), (instance-of $c2$, $CPU1$), (instance-of $c3$, $CPU2$)}, where $CLANG$={$CPU1$, $CPU2$, ...}.
Based on this definition, a corresponding configuration result (solution) is defined as follows [19], where the semantics of description terms are given using an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a domain of values and $(\cdot)^{\mathcal{I}}$ is a mapping from concept descriptions to subsets of $\Delta^{\mathcal{I}}$ and from role descriptions to sets of 2-tuples over $\Delta^{\mathcal{I}}$.

Definition 2 (Valid configuration): Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, (\cdot)^{\mathcal{I}} \rangle$ be a model of a configuration knowledge base KB, $CLANG = C_{config} \cup R_{config}$ a configuration language, and $CONF = COMPS \cup ROLES$ a description of a configuration. $COMPS$ is a set of tuples $\langle C_i, INDIVS_{C_i} \rangle$ for every $C_i \in C_{config}$, where $INDIVS_{C_i} = \{ci_1, \ldots, ci_{n_i}\} = C_i^{\mathcal{I}}$ is the set of individuals of concept C_i. These individuals identify components in an actual configuration. $ROLES$ is a set of tuples $\langle R_j, TUPLES_{R_j} \rangle$ for every $R_j \in R_{config}$ where $TUPLES_{R_j} = \{\langle rj_1, sj_1 \rangle, \ldots, \langle rj_{m_j}, sj_{m_j} \rangle\} = R_j^{\mathcal{I}}$ is the set of tuples of role R_j defining the relation of components in an actual configuration.□

A valid configuration for our example domain is $CONF$={$\langle CPU1, \{c1, c2\}\rangle$, $\langle CPU2, \{c3\}\rangle$, $\langle MB1, \{m1\}\rangle$, $\langle MB2, \{m2\}\rangle$, ⟨*mb-of-cpu*, {$\langle m1, c1\rangle$, $\langle m1, c2\rangle$, $\langle m2, c2\rangle$}⟩, ...}.
The automatic derivation of an OIL-based configuration knowledge base requires a clear definition of the semantics of the used UML modeling concepts. In the following we define the semantics of UML configuration models by giving a set of corresponding translation rules into OIL. The resulting knowledge base restricts the set of possible configurations, i.e. enumerates the possible instance models which strictly correspond to the UML class diagram defining the product structure.

4 Translation of UML configuration models into OIL

In the following we present an approach which allows the application of the Unified Modeling Language (UML) [10] to configuration knowledge acquisition and interchange. UML configuration models can automatically be translated into a corresponding OIL [8] or DAML+OIL [9] based representation. This enables a standardized representation of configuration models and configuration knowledge interchange between different configuration environments using standard Web technologies. The usage of UML allows the integration of configuration technology into industrial software development processes, furthermore a standard graphical knowledge acquisition frontend is provided which is crucial for effective development and maintenance of configuration knowledge bases especially in the context of distributed configuration problem solving [21]. For the modeling concepts discussed in Section 2 (component types, generalization hierarchies, part-whole relationships, compatibility and requirement constraints, resource constraints, and port connections) we present a set of rules for translating those concepts into an OIL-based representation. UML is based on a graphical notation - therefore our translation starts from such a graphical description of a configuration domain. In the following, $GREP$ denotes the graphical representation of the UML configuration model.

Rule 1 (Component types): Let c be a component type, a an attribute of c, and d be the domain of a in $GREP$, then DD is extended with

class-def c.
slot-def a.
c: slot-constraint a cardinality 1 d.

For those component types $c_i, c_j \in \{c_1, ..., c_m\}$ ($c_i \neq c_j$), which do not have any supertypes in $GREP$, DD is extended with

disjoint c_i, c_j. □

Example 1 (Component type CPU):
class-def CPU.
slot-def $clockrate$.
CPU: slot-constraint $clockrate$ cardinality 1 ((min 300) and (max 500)).
disjoint $CPU\ MB$.
disjoint $MB\ Screen$.
... □

Subtyping in the configuration domain means that attributes and roles of a given component type are inherited by its subtypes. In most configuration environments a disjunctive and complete semantics is assumed for generalization hierarchies, where the disjunctive semantics can be expressed using the $disjoint$ axiom and the completeness can be expressed by forcing the superclass to conform to one of the given subclasses as follows.

Rule 2 (Generalization hierarchies): Let u and $d_1, ..., d_n$ be classes (component types) in $GREP$, where u is the superclass of $d_1, ..., d_n$, then DD is extended with
$d_1, ..., d_n$: subclass-of u.
u: subclass-of (d_1 or ... or d_n).
$\forall\ d_i, d_j \in \{d_1, ..., d_n\}$ ($d_i \neq d_j$) : disjoint $d_i\ d_j$. □

Example 2 ($CPU1$, $CPU2$ subclasses of CPU):
$CPU1$: subclass-of CPU.
$CPU2$: subclass-of CPU.
CPU: subclass-of ($CPU1$ or $CPU2$).
disjoint $CPU1\ CPU2$. □

Part-whole relationships are important model properties in the configuration domain. In [22,23,12] it is pointed out that part-whole relationships have quite variable semantics depending on the regarded application domain. In most configuration environments, a part-whole relationship is described by the two basic roles *partof* and *haspart*. Depending on the intended semantics, different additional restrictions can be placed on the usage of those roles. In the following these two basic roles (which can be refined with domain specific semantics if needed) are introduced. We discuss two facets of part-whole relationships which are widely used for configuration knowledge representation and are also provided by UML, namely *composite* and *shared* part-whole relationships. In UML composite part-whole relationships are denoted by a black diamond, shared part-whole relationships are denoted by a white diamond[7]. If a component is a compositional part of another component then strong ownership is required, i.e., it must be part of exactly one component. If a component is a non-compositional (shared) part of another component, it can be shared between different components. Multiplicities used to describe a part-whole relationship denote how many parts the aggregate can consist of and between how many aggregates a part can be shared if the aggregation is non-composite. The basic structure of a part-whole relationship is shown in Figure 2.

Rule 3 (Part-whole relationships): Let w and p be

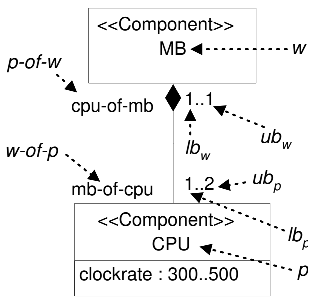


Fig. 2. Part-whole relationships

component types in $GREP$, where p is a part of w and ub_p is the upper bound, lb_p the lower bound of the multiplicity of the part, and ub_w is the upper bound, lb_w the lower bound of the multiplicity of the whole. Furthermore let *w-of-p* and *p-of-w* denote the names of the roles of the part-whole relationship between w and p, where *w-of-p* denotes the role connecting the part with the whole and *p-of-w* denotes the role connecting the whole with the part, i.e., *p-of-w* $\sqsubseteq$ *haspart*, *w-of-p* $\sqsubseteq Partof_{mode}$, where $Partof_{mode} \in \{partof_{composite}, partof_{shared}\}$. The roles $partof_{composite}$ and $partof_{shared}$ are assumed to be disjoint, where $partof_{composite} \sqsubseteq partof$ and $partof_{shared} \sqsubseteq partof$. DD is extended with

slot-def *w-of-p* subslot-of $Partof_{mode}$ inverse *p-of-w* domain p range w.
slot-def *p-of-w* subslot-of *haspart* inverse *w-of-p* domain w range p.
p: slot-constraint *w-of-p* min-cardinality $lb_w\ w$.
p: slot-constraint *w-of-p* max-cardinality $ub_w\ w$.
w: slot-constraint *p-of-w* min-cardinality $lb_p\ p$.
w: slot-constraint *p-of-w* max-cardinality $ub_p\ p$. □

Remark: The semantics of *shared* part-whole relationships ($partof_{shared} \sqsubseteq partof$) are defined by simply restricting the upper bound and the lower bound of the corresponding roles. In addition the following restriction must hold for each concept using partof relationships:

[7] Note that in our $Computer$ configuration example we only use composite part-whole relationships - as mentioned in [12], composite part-whole relationships are often used when modeling physical products, whereas shared part-whole relationships are used to describe abstract entities such as services. The lower and upper bounds of the whole are not explicitly modeled (see Figure 1) - if not explicitly mentioned we assume multiplicity 1.

(((*slot-constraint* $partof_{composite}$ *cardinality 1 top*) *and* (*slot-constraint* $partof_{shared}$ *cardinality 0 top*)) *or* (*slot-constraint* $partof_{composite}$ *cardinality 0 top*)).

This restriction denotes the fact that a component which is connected to a whole via composite relationship must not be connected to any other component. □

Example 3 (MB partof $Computer$):
slot-def *computer-of-mb* subslot-of $partof_{composite}$
inverse *mb-of-computer*
domain MB range $Computer$.
slot-def *mb-of-computer* subslot-of $haspart$
inverse *computer-of-mb*
domain $Computer$ range MB.
MB: slot-constraint *computer-of-mb* min-cardinality 1 $Computer$.
MB: slot-constraint *computer-of-mb* max-cardinality 1 $Computer$.
$Computer$: slot-constraint *mb-of-computer* min-cardinality 1 MB.
$Computer$: slot-constraint *mb-of-computer* max-cardinality 2 MB. □

Necessary part-of structure properties. In the following we show how the constraints contained in a product configuration model (e.g., an $IDEUnit\ requires$ an $MB1$) can be translated into a corresponding OIL representation. For a consistent application of the translation rules it must be ensured that the components involved are parts of the same sub-configuration, i.e., the involved components must be connected to the same instance of the component type that represents the common root[8] for these components (the components are within the same mereological context [12]). This can simply be expressed by the notion that component types in such a hierarchy must each have a unique superior component type in $GREP$. If this uniqueness property is not satisfied, the meaning of the imposed (graphically represented) constraints becomes ambiguous, since one component can be part of more than one substructure and consequently the scope of the constraint becomes ambiguous.
For the derivation of constraints on the product model we introduce the macro $navpath$ as an abbreviation for a navigation expression over roles. For the definition of $navpath$ the UML configuration model can be interpreted as a directed graph, where component types are represented by vertices and part-whole relationships are represented by edges.

Definition 3 (Navigation expression): Let $path(c_1, c_n)$ be a path from a component type c_1 to a component type c_n in $GREP$ represented through a sequence of expressions of the form $haspart(C_i, C_j,\ Name_{Ci})$ denoting a direct partof relationship between the component types C_i and C_j. Furthermore, $Name_{Ci}$ represents the name of the corresponding $haspart$ role. Such a path in $GREP$ is represented as

$path(c_1, c_n) = < haspart(c_1, c_2, name_{c1}),$
$haspart(c_2, c_3, name_{c2}), ...,$
$haspart(c_{n-1}, c_n, name_{cn-1}) >$
Based on the definition of $path(c_1, c_n)$ we can define the macro $navpath(c_1, c_n)$ as
slot-constraint $name_{c1}$
has-value(slot-constraint $name_{c2}$ ...
has-value(slot-constraint $name_{cn-1}$ has-value c_n)...).
□

Example 4 ($navpath(Computer, CPU1)$):
$Computer$: slot-constraint *mb-of-computer*
has-value (slot-constraint *cpu-of-mb* has-value $CPU1$).
□

The concept of a *nearest common root* is based on the definition of $navpath$ as follows.

Definition 4 (Nearest common root): A component type r is denoted as nearest common root of the component types c_1 and c_2 in $GREP$, *iff* there exist paths $path(r, c_1)$, $path(r, c_2)$ and there does not exist a component type r', where r' is a part[9] of r with paths $path(r', c_1)$, $path(r', c_2)$. □

When regarding the example configuration model of Figure 1, MB is the nearest common root of CPU and $Videocard$. Conform to Definition 4 the component type $Computer$ is not a nearest common root of CPU and $Videocard$. As shown in Figure 3, the component type $Computer$ is the nearest common root of the component types $IDEUnit$ and $CPU1$.

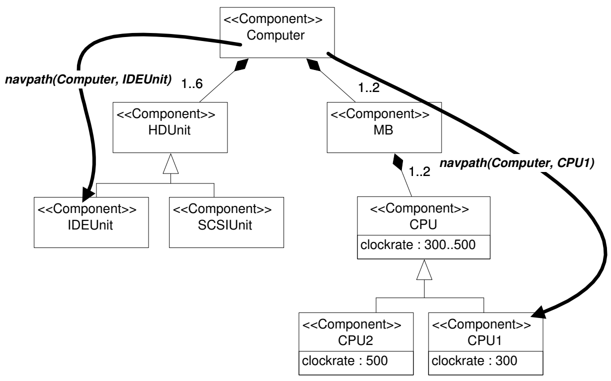


Fig. 3. Navigation paths from $Computer$ to $CPU1$ and $IDEUnit$

Requirement constraints. A *requires* constraint between two component types c_1 and c_2 in $GREP$ denotes the fact that the existence of an instance of component type c_1 requires that an instance of component type c_2 exists and is part of the same (sub)configuration.

Rule 4 (Requirement constraints): Given the relationship c_1 *requires* c_2 between the component types c_1 and c_2 in $GREP$ with r as common root of c_1 and c_2, then DD is extended with
r: ((not($navpath(r, c_1)$)) or $navpath(r, c_2)$). □

[8] In Figure 3 the component type *Computer* is the unique common root of the component types *IDEUnit* and *CPU1*.

[9] In this context $partof$ is assumed to be transitive.

The condition part of the implication describes a path from the common root to the component c_1; the consequent contains a corresponding path to the required component c_2.

Example 5 ($IDEUnit$ requires $MB1$):

$Computer$: ((not (slot-constraint *hdunit-of-computer* has-value $IDEUnit$)) or (slot-constraint *mb-of-computer* has-value $MB1$)) □

Compatibility constraints. A *compatibility* constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ denotes the fact that the existence of a tuple of instances corresponding to the types in c is not allowed in a final configuration (result).

Rule 5 (Compatibility constraints): Given a compatibility constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ with r as common root of $\{c_1, c_2, ..., c_n\}$, then DD is extended with

r: (not(($navpath(r, c_1)$) and ($navpath(r, c_2)$) and ... and ($navpath(r, c_n)$))). □

Example 6 ($SCSIUnit$ incompatible with $MB1$):

$Computer$: (not ((slot-constraint *hdunit-of-computer* has-value $SCSIUnit$) and (slot-constraint *mb-of-computer* has-value $MB1$))). □

Resource constraints. *Resource* constraints can be modeled in UML using stereotyped classes representing types of resources and stereotyped dependencies with a corresponding tagged value indicating resource production and consumption. Resource balancing tasks [3] are defined within a (sub)tree (context) of the configuration model. To map a resource balancing task into DD, additional attributes (res_p and res_c in the following) have to be defined for the component types acting as producers and consumers. In addition we have to introduce aggregate functions as representation concepts, which are currently supported neither in OIL nor DAML+OIL. However, there exist proposals [24] to extend description logics by concepts which allow the modeling of such resource constructs. The following representation of aggregate functions is based on the formalism presented in [24], where a set of predicates P associated with binary relations (e.g., $\leq$, $\geq$, $<$, $>$) over a value domain $dom(D)$ and a set of aggregation functions $agg(D)$ (e.g., $count, min, max, sum$) are defined. Let ϕ be the path leading to the concept whose features are aggregated. Then the definitions of [24] require that all but the last one of the roles in ϕ must be features, i.e. functional relations.

Rule 6 (Resource constraints): Let $p = \{p_1, p_2, ..., p_n\}$ be producing component types and $c = \{c_1, c_2, ..., c_m\}$ be consuming component types of resource res in $GREP$. Furthermore, let res_p be a feature common to all component types in p, and res_c be a feature common to the types in c, where the values of res_p and res_c are defined by the tagged values of the consumes and produces dependencies in $GREP$.

A resource constraint for DD_{DL} can be expressed as

r: $P(r_1^1\ r_2^1\ ...\ r_{n-1}^1\ \Sigma(r_n^1 \circ res_p), r_1^2\ r_2^2\ ...\ r_{m-1}^2\ \Sigma(r_m^2 \circ res_c))$

where r represents the nearest common root of the elements in c and p, P is a binary relation, and ¡$r_1^1, r_2^1, ..., r_n^1$¿, ¡$r_1^2, r_2^2, ..., r_m^2$¿ represent navigation paths from r to the elements of p and c. □

Example 7 (Capacity needed by $Software$ $\leq$ Capacity provided by $HDUnit$):

$DTPSoftware$: slot-constraint $Capacity$ cardinality 1 (equal 50).
$Textedit$: slot-constraint $Capacity$ cardinality 1 (equal 100).
$SCSIUnit$: slot-constraint $Capacity$ cardinality 1 (equal 20000).
$IDEUnit$: slot-constraint $Capacity$ cardinality 1 (equal 10000).
$Computer$: lesseq (sum(*sw-of-computer*$\circ Capacity$), sum(*hdunit-of-computer*$\circ Capacity$)). □

Port connections. Ports in the UML configuration model (see Figure 4) represent physical connection points between components (e.g., a $Videocard$ can be connected to a $Screen$ using the port combination $videoport_1$ and $screenport_2$). In UML we introduce ports using classes with stereotype $Port$ - these ports are connected to component types using part-whole relationships.
In order to represent port connections in OIL, we introduce them via a separate concept $Port$[10]. The role $compnt$ indicates the component concept that the port belongs to, the role $portname$ determines its name, and the role $conn$ describes the relation to the counterpart port concept of the connected component.

Rule 7 (Port connections): Let $\{a,\ b\}$ be component types in $GREP$, $\{pa,\ pb\}$ be the corresponding connected port types, $\{m_a,\ m_b\}$ the multiplicities of the port types with respect to $\{a,\ b\}$[11], and $\{\{lb_{pa},\ ub_{pa}\ \},\ \{lb_{pb},\ lb_{pb}\}\}$ the lower bound and upper bound of the multiplicities of the port types with respect to $\{pa,\ pb\}$, then DD is extended with

class-def pa subclass-of $Port$.
class-def pb subclass-of $Port$.
pa: slot-constraint $portname$ cardinality 1 (one-of pa_1 ... pa_{ma}).
pa: slot-constraint $conn$ min-cardinality lb_{pa} pb.
pa: slot-constraint $conn$ max-cardinality ub_{pa} pb.
pa: slot-constraint $conn$ value-type pb.
pa: slot-constraint $compnt$ cardinality 1 a.
pb: slot-constraint $portname$ cardinality 1 (one-of pb_1 ... pb_{mb}).
pb: slot-constraint $conn$ min-cardinality lb_{pb} pa.
pb: slot-constraint $conn$ max-cardinality ub_{pb} pa.
pb: slot-constraint $conn$ value-type pa.
pb: slot-constraint $compnt$ cardinality 1 b. □

Example 8 ($Videocard$ connected to $Screen$):

[10] Note, that in OIL there are only predicates with arity 1 or 2 available, therefore the representation of port connections must be realized by the definition of additional concepts.

[11] In this context no differentiation between lower and upper bound is needed since the number of ports of a component is exactly known beforehand.

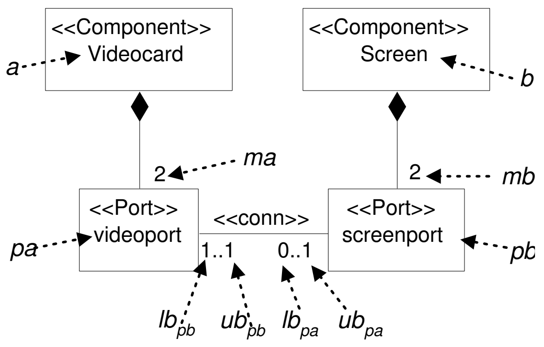


Fig. 4. Ports in the configuration model

class-def $videoport$ subclass-of $Port$.
class-def $screenport$ subclass-of $Port$.
$videoport$: slot-constraint $portname$ cardinality 1 one-of ($videoport_1$ $videoport_2$).
$videoport$: slot-constraint $conn$ min-cardinality 0 $screenport$.
$videoport$: slot-constraint $conn$ max-cardinality 1 $screenport$.
$videoport$: slot-constraint $conn$ value-type $screenport$.
$videoport$: slot-constraint $compnt$ cardinality 1 $Videocard$.
$screenport$: slot-constraint $portname$ cardinality 1 (one-of $screenport_1$ $screenport_2$).
$screenport$: slot-constraint $conn$ min-cardinality 1 $videoport$.
$screenport$: slot-constraint $conn$ max-cardinality 1 $videoport$.
$screenport$: slot-constraint $conn$ value-type $videoport$.
$screenport$: slot-constraint $compnt$ cardinality 1 $Screen$.
□

Using the defined structure for port connections, the constraint "a $Videocard$ must be connected via $videoport_1$ with a $Screen$ via $screenport_1$" can be formulated as follows.

Example 9: $Videocard$:
(slot-constraint *videoport-of-videocard* has-value
 ((slot-constraint $portname$ has-value
  (one-of $videoport_1$)) and
 (slot-constraint $conn$ has-value
  ((slot-constraint $compnt$ has-value $Screen$) and
  (slot-constraint $portname$ has-value
   (one-of $screenport_1$)))))). □

The application of the modeling concepts presented in this paper has its limits when building configuration knowledge bases in some domains there exist complex constraints that do not have an intuitive graphical representation, i.e. cannot be modeled using the graphical constraints presented in this paper. Happily (with some minor restrictions), we are able to represent such constraints using languages such as OIL or DAML+OIL[12]. UML itself has an integrated constraint language (Object Constraint Language - OCL [25]) which allows the formulation of constraints on object structures. The translation of OCL constraints into representations of Semantic Web ontology languages is the subject of future work, a translation into a predicate logic based representation of a configuration problem has already been discussed in [26]. The current version of our prototype workbench supports the generation of OIL-based configuration knowledge bases from UML models which are built using the modeling concepts presented in this paper, i.e. concepts for designing the product structure and concepts for defining basic constraints (e.g. *requires*) on the product structure.

[12] In [19] those restrictions are discussed in detail - basically aggregate functions, roles with arity greater than two and an assertional language allowing the usage of variables were identified as concepts additionally needed for building configuration models.

5 Related Work

The definition of a common representation language to support knowledge interchange between and integration of different knowledge-based systems are important issues in the configuration domain. In [12] one approach to collect relevant concepts for modeling configuration knowledge bases is presented. The defined ontology is based on Ontolingua [27] and represents a synthesis of resource-based, function-based, connection-based, and structure-based configuration approaches. This ontology is a kind of meta-ontology which is similar to the UML profile for configuration models presented in this paper. Conforming to the definition of [16] a UML configuration model is an ontology, i.e. it restricts the sorts of objects relevant for the domain, defines the possible properties of objects and the relationships between objects. Compared to the approach presented in this paper, [12] do not provide a formal semantics for the proposed modeling concepts.

The work of [28] shows some similarities to the work presented in this paper. Starting with a UML ontology (which is basically represented as a class diagram) corresponding JAVA classes and RDF documents are generated. The work presented in this paper goes one step further by providing a UML profile for the configuration domain and a set of rules allowing the automatic derivation of executable configuration knowledge bases. The correspondence between Semantic Web ontology languages and UML is shown on the object level as well as on the constraint level, where a set of domain specific constraints (e.g. *requires*) are introduced as stereotypes in the configuration profile - for these constraints a representation in OIL has been shown.

Most of the required means for expressing configuration knowledge are already provided by current versions of Semantic Web knowledge representation languages. However, in order to provide full fledged configuration knowledge representation, certain additional expressivity properties must be fulfilled - this issue is discussed in [19], where aggregation functions, n-ary relationships, and the provision of variables have been identified as the major required add-ons for ontology languages such as DAML+OIL. Within the Semantic Web community there are ongoing efforts to increase the expressiveness of Web ontology languages. DAML-L [29] is a language which builds upon the basic concepts of DAML. RuleML [30], CIF (Constraint Interchange Format) [31], or TRIPLE [32] are similar approaches with the goal to provide rule languages for the Semantic Web - an investigation of those languages w.r.t. to configuration knowledge representation is the subject of future work.

6 Conclusions

In this paper we presented an approach to integrate the development of configuration knowledge bases for the Semantic Web into standard industrial software development processes. Founded on a description logic based definition of a configuration task, we presented a set of rules for translating UML configuration models into a corresponding OIL-based representation enabling model checking for UML configuration models and knowledge sharing between different configurators in Web-based environments. Our approach supports the design of customizable product and service models in an intuitive graphical manner, which eases the integration task in the context of distributed configuration problem solving. The concepts presented in this paper are implemented in a corresponding configuration knowledge acquisition workbench.

References

1. Barker, V., O'Connor, D., Bachant, J., Soloway, E.: Expert systems for configuration at Digital: XCON and beyond. Communications of the ACM **32** (1989) 298–318
2. Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H., Stumptner, M.: Configuring Large Systems Using Generative Constraint Satisfaction. IEEE Intelligent Systems **13** (1998) 59–68
3. M. Heinrich, E.J.: A resource-based paradigm for the configuring of technical systems from modular components. In: Proceedings of the 7^{th} IEEE Conference on AI applciations (CAIA), Miami, FL, USA (1991) 257–264
4. Wright, J., Weixelbaum, E., Vesonder, G., Brown, K., Palmer, S., Berman, J., Moore, H.: A Knowledge-Based Configurator that supports Sales, Engineering, and Manufacturing at AT&T Network Systems. AI Magazine **14** (1993) 69–80
5. Sabin, D., Weigel, R.: Product Configuration Frameworks - A Survey. In Faltings, B., Freuder, E., eds.: IEEE Intelligent Systems, Special Issue on Configuration. Volume 13. IEEE (1998) 50–58
6. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: Web-based Configuration of Virtual Private Networks with Multiple Suppliers, Cambridge, UK, Kluwer Academic Publisher (2002)
7. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: Semantic Configuration Web Services in the CAWICOMS Project, Sardinia, Italy (2002)
8. Fensel, D., vanHarmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems **16** (2001) 38–45
9. vanHarmelen, F., Patel-Schneider, P., Horrocks, I.: A Model-Theoretic Semantics for DAML+OIL. www.daml.org (March 2001)
10. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)
11. Felfernig, A., Friedrich, G., Jannach, D.: UML as domain specific language for the construction of knowledge-based configuration systems. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) **10** (2000) 449–469
12. Soininen, T., Tiihonen, J., Mnnist, T., Sulonen, R.: Towards a General Ontology of Configuration. AI Engineering Design Analysis and Manufacturing Journal, Special Issue: Configuration Design **12** (1998) 357–372
13. Booch, G.: Object-Oriented Analysis and Design with Applications. Addison-Wesley Object Technology Series (1994)
14. Jacobson, I., Christerson, M., vergaard, G.: Object-oriented Software Engineering - A Use-Case Driven Approach. Addison- Wesley (1992)
15. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design, New Jersey, USA (1991)
16. Chandrasekaran, B., Josephson, J., Benjamins, R.: What Are Ontologies, and Why do we Need Them? IEEE Intelligent Systems **14,1** (1999) 20–26
17. Mittal, S., Frayman, F.: Towards a Generic Model of Configuration Tasks. In: Proceedings 11^{th} International Joint Conf. on Artificial Intelligence, Detroit, MI (1989) 1395–1401
18. Stumptner, M.: An overview of knowledge-based configuration. AI Communications **10** (June, 1997)
19. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., Zanker, M.: A Joint Foundation for Configuration in the Semantic Web. Technical Report KLU-IFI-02-05 (2001)
20. Borgida, A.: On the relative expressive power of description logics and predicate calculus. Artificial Intelligence **82** (1996) 353–367
21. Ardissono, L., Felfernig, A., Friedrich, G., Jannach, D., Zanker, M., Schfer, R.: Customer-Adaptive and Distributed Online Product Configuration in the CAWICOMS Project. In Proceedings of the Workshop on Configuration, in conjunction with the 17^{th} International Conference on Artificial Intelligence (IJCAI-2001) (to appear) (2001)
22. Artale, A., Franconi, E., Guarino, N., Pazzi, L.: Part-Whole Relations in Object-Centered Systems: An Overview. Data & Knowledge Engineering **20** (1996) 347–383
23. Sattler, U.: Description Logics for the Representation of Aggregated Objects. In: Proceedings of the 14^{th} European Conference on Artificial Intelligence (ECAI 2000), Berlin, Germany (2000) 239–243
24. Baader, F., Sattler, U.: Description Logics with Concrete Domains and Aggregation. In: Proceedings of the 13^{th} European Conference on Artificial Intelligence (ECAI '98), Brighton, UK (1998) 336–340
25. Warmer, J., Kleppe, A.: The Object Constraint Language - Precise Modeling with UML. Addison Wesley Object Technology Series (1999)
26. Felfernig, A., Friedrich, G., Jannach, D.: Generating product configuration knowledge bases from precise domain extended UML models. In: Proceedings of the 12^{th} International Conference on Software Engineering and Knowledge Engineering (SEKE'2000), Chicago, USA (2000) 284–293
27. Gruber, T.: Ontolingua: A mechanism to support portable ontologies. Technical Report KSL 91-66 (1992)
28. Cranefield, S.: UML and the Semantic Web. In: Semantic Web Working Symposium, Stanford, CA, USA (2001)
29. McIlraith, S., Son, T., Zeng, H.: Mobilizing the Semantic Web with DAML-Enabled Web Services. In: Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web, Seattle, WA (2001) 29–39
30. Grosof, B.: Standardizing XML Rules. In: Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web, Seattle, WA (2001) 2–3
31. Gray, P., Hui, K., Preece, A.: An Expressive Constraint Language for Semantic Web Applications. In: Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web, Seattle, WA (2001) 46–53
32. Sintek, M., Decker, S.: TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In: to appear: Proceedings of International Semantic Web Conference (ISWC), Sardinia, Italy (2002)

On Modeling Conformance for Flexible Transformation over Data Models[⋆]

Shawn Bowers and Lois Delcambre

OGI School of Science & Engineering, OHSU
Beaverton, Oregon 97006, USA
{shawn, lmd}@cse.ogi.edu

Abstract. Many useful and commonly used data models exist for structured information. Such models often use only a limited number of *basic structures* to represent information such as sets, bags, lists, attribute-value pairs, and scalars. They differ, however, in whether or not they *permit* the specification of schema (or DTD or template), whether they permit *more than one* schema, and whether they *require* schema. Some even allow multiple levels of schema-instance conformance (*e.g.*, RDF and Topic Maps) where the type of an instance is an instance, and so on. Support for transforming between such varying representation schemes remains a significant challenge. We extend our earlier work on generic representation and transformation of model-based information by introducing a richer metamodel and abstract framework for representation. We also introduce several steps toward our vision of high-level transformations where complex mapping rules are defined using transformation patterns and by exploiting inherent constraints.

1 Introduction

Taking information in one representation scheme (such as XML) and extracting some or all of it for use in another scheme (such as a Topic Map, a Relational database, or as RDF triples) is a suprisingly difficult task. In fact, few tools exist to help perform such transformations, which means that (potentially) complex, special purpose programs must be written for even very simple mappings. One reason such conversions are difficult is that representation schemes differ in the basic structural constructs and schema constraints they provide for organizing information. As a consequence, straightforward, one-to-one mappings between schemes rarely exist. In addition, the absence of high-level languages for expressing transformations between schemes places the burden of conversion on these special-purpose programs, making it difficult to define, maintain, and reason about transformations.

We observe, however, that structured information is typically based on a small set of structural constructs, composed in various ways. When described explicitly, these structural constructs can be exploited directly for transformation. For example, the constructs offered by the Relational model can be viewed as a set of tables, each consisting of a bag of rows, where each row has a set of named atomic or scalar attribute values. Similarly, an XML document can be viewed as a set of elements, possibly with attributes, where each element has a list of sub-elements or scalar values (PCDATA). We propose that a wide range of useful transformations can be performed through mappings between representation scheme constructs, *e.g.*, elements to tuples in tables, and so on.

To this end, we present a uniform framework that allows the description of arbitrary representation schemes along with a transformation language that can easily convert information within and across schemes. The goals of our research are to:

1. capture the representation scheme or data model explicitly—by instantiating and composing the basic structures (or construct types) of a metamodel,
2. support representation schemes with varying types of conformance, *i.e.*, we model schema-instance relationships for representation schemes where schema (or type information) can be required, optional, or can occur at multiple levels, and
3. express transformation rules declaratively, *e.g.*, by allowing users to indicate simple correspondences between data model constructs and converting the associated information automatically.

This paper is organized as follows. Section 2 describes various data models and representation schemes, with emphasis on how each differs in their use of conformance relationships. Section 3 presents the uniform framework for representing structured information, along with the four relationships of interest. Section 4 introduces transformation rules that exploit this representation and also discusses the possibility of higher-level transformation rules, making rule specification easier. Section 5 discusses related work and we conclude in Section 6 with a brief discussion of work in progress.

2 Data Models and Conformance

Database systems are *schema-first* in that a schema must be defined before any data can be placed in the database. The role of the schema is to introduce and name application-specific structures (such as the "Employee" table with "SSN" and "Name" as attributes for Employee) that will be used to hold application data. A schema imposes uniform structure and constraints on application data. Various tools such as a query processor, in turn, can exploit this uniform structure.

A number of data models exists that are not schema-first, including XML and other semi-structured models, RDF, the Topic Map Model, and various hypertext data models [1,2]. Both RDF and XML are models where the schema, *i.e.*, the RDF schema or DTD, respectively, is optional. More than that, even with a schema, the Topic Map model, RDF, and XML (with an "open" DTD) permit additional, schema-free structures to be freely mixed with structures that conform to a schema.

We model *conformance relationships* explicitly, as appropriate, for each data model of interest. Table 1 provides

[⋆] This work supported in part by the National Science Foundation through grants EIA 9983518 and IIS 9817492.

example data models with some of their associated structures that participate in conformance. For example, in object-oriented models, objects conform to classes, whereas in XML, elements conform to element types, and so on.

Table 1. Example data model structures involved in conformance

Data Model	*Selected structures, related by the conformance relationship*
Object-Oriented	Objects conform to Classes.
Relational	A Table of Tuples conforms to a Relation Scheme.
XML w/ DTD	Elements conform to Element Types and Attributes to Attribute Types.
RDF w/ RDFS	Objects (resources) conform to Classes and Properties to Property Types.
XTM	Topics conform to Topic Types (Topics), Associations conform to Association Types (Topics), and Occurences conform to Occurence Types (Topics).
E-R	Entities conform to Entity Types, Relationships conform to Relationship Types, and Values conform to Value Types.

Table 2 describes some of the properties of conformance that are present in the data models of Table 1. First, we consider how the conformance relationship is established with three possibilities: implicitly, upon request, and explicitly as shown in the first section of Table 2. In the next section of Table 2, we see that the schema-first models also require conformance for their data values whereas the other models do not. And the models with optional conformance are also open: they permit conforming and non-conforming data to be freely mixed, as shown in the third section of Table 2. Some models allow a construct to conform to several structures, namely all but the relational model (fourth section of Table 2). Finally, we see that all of the non-schema-first models except XML permit multiple levels of conformance.

To represent a wide range of data models explicitly, we must support variations in the conformance relationship as shown by Table 2. We discuss the details of our representation in the next section, discuss transformation in Section 4, and further consider related work in Section 5.

3 The Uniform Framework

The main component of the uniform framework is a three-level, metamodel architecture that incorporates conformance relationships. In this section, we describe our metamodel architecture, introduce a concrete metamodel (*i.e.*, a specific set of basic structures), and present a logic-based description language that enables uniform data model, schema, and instance description.

3.1 The Generic Metamodel Architecture

Figure 1 exemplifies a typical metamodel architecture. As shown, the architecture has four levels: the metamodel, data model, schema, and data levels, where each item in a level is considered an instance of an item in the level above it.

Table 2. Examples properties of the conformance relationship

Conformance Properties	*Data Model*
How is the conformance relationship established?	
when data is created (implicitly)	Object-Oriented, Relational, and E-R
when computed (upon request)	XML w/ DTD (documents must be validated)
when conformance is declared (explicitly)	RDF w/ RDFS (via `rdf:type` property) and XTM (via class-instance association)
Is conformance required?	
required	Object-Oriented, Relational, and E-R
optional	RDF (RDF Schema is not required), XTM (type information not required), and XML (DTD is not required)
Is conformance open?	
no	Object-Oriented, Relational, and E-R
yes	RDF w/ RDFS, XTM, and XML
Is multiple conformance allowed?	
no	Relational
yes	Object-Oriented (inheritance), RDF w/ RDFS (Property/Class with multiple inheritance), XTM (Topics can be instances of multiple Topic Types), and E-R
How many levels of conformance are permitted?	
one	Relational, E-R, and Object-Oriented (when not in the next entry)
two (sometimes)	Object-Oriented (some models support classes as objects)
zero to any number	XML (zero or one), XTM, and RDF

As shown, typical metamodel architectures are characterized by their assumption that data models are schema-first, *i.e.*, schema items must be created prior to data items.

In contrast, Figure 2 shows the three-level metamodel architecture we employ. The top level (denoted as the metamodel) consists of *construct types*, which represent the basic structures used within data models for defining schema and data. Examples include types such as collection, name-value pair, atomic, and so on. The middle layer defines data model and schema *constructs* along with their composition. For example, both a `class` and `object` construct along with an explicit conformance definition between them (represented by the relationship type labeled `conf`) are defined in the middle layer. In this way, the metamodel is used to define all the constructs of the data model, not just those for creating schema.

Finally, the bottom level represents *instances* and (data-level) *conformance relationships*. For example, an object (with the value “steve”) would be connected with a particular class (with name “person”) using the `d-inst` relationship.

The architecture distinguishes three kinds of instance-of relationships, in addition to the conformance relationship, as shown in Figure 2. Constructs introduced in the middle layer are necessarily an instance (`ct-inst`, read as “construct-type instance”) of a metamodel construct type. (Note that the metamodel currently has a range of basic structures as construct types and the set of construct types can be easily extended if needed.) Similarly, any value introduced in the bottom layer is necessarily an instance (`c-inst`, read

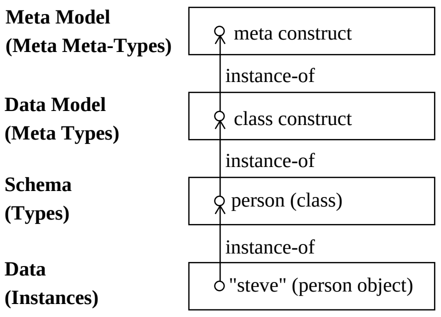


Fig. 1. A typical four-level metamodel architecture.

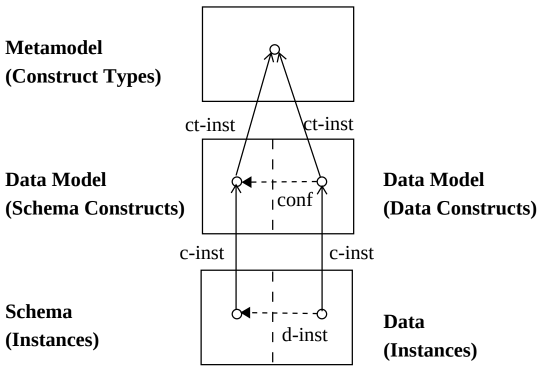


Fig. 2. The three-level metamodel architecture.

as "construct instance") of the constructs introduced in the middle layer. The flexibility of the framework is due to the conformance relationships. Conformance relationships are expressed within the middle layer (*e.g.*, to indicate that an XML element can optionally conform to an XML element type) and the corresponding data-level instance-of relationships (`d-inst`, read as "data instance") expressed within the bottom layer (*e.g.*, to indicate that a particular element is an instance of another element type).

To make our framework concrete, we introduce the notion of a *configuration*, which consists of a particular choice of metamodel (*i.e.*, set of construct types), a particular data model (*i.e.*, constructs) defined using the construct types, and a collection of instances defined using the constructs. A configuration serves as input to a transformation. Thus, a transformation takes one (or possibly more) configurations and generates a new configuration.

A configuration $\mathcal{C}$ is a tuple (M, D, I, T) whose components are defined as follows.

– M is a set of named construct types in the top layer. (*The Metamodel*)
– D is a tuple (C, R) where C is a set of named constructs defined using the types of M, and R is a set of ordered pairs (c_1, c_2) for $c_1, c_2 \in C$ denoting a conformance definition and is read as "instances of c_1 can conform to instances of c_2." (*The Middle Layer*)
– I is a set of named instances. The name of an instance acts as its unique identifier. (*The Bottom Layer*)
– T is a tuple (T_{ct}, T_c, T_d) in which $T_{ct} \cup T_c \cup T_d$ is the set of type-instance relationships of the configuration where T_{ct}, T_c, and T_d are disjoint sets of ordered pairs (i_1, i_2) such that for $t \in T_{ct}$ we require $i_1 \in C$ and $i_2 \in M$ (*i.e.*, `ct-inst`), for $t \in T_c$ we require $i_1 \in I$ and $i_2 \in C$ (*i.e.*, `c-inst`), and for $t \in T_d$ we require $i_1, i_2 \in I$ (*i.e.*, `d-inst`) such that $(i_1, c_1), (i_2, c_2) \in T_C$ and $(c_1, c_2) \in R$. (*Cross-Layer Connections and Data Instances*)

We informally define construct type, construct, and instance as follows.[1]

– A construct type ct is a tuple (adt, P) such that adt represents an abstract (or paramaterized) data type and P is a set of restriction properties.
– A construct c is a tuple (ct, ρ) such that $ct \in M$ is a construct type for which c is an instance (and hence $(c, ct) \in T_{ct}$) and ρ is a set of restrictions that obey P and serve to restrict the instances of the construct and define its compositional aspects.
– An instance i is a tuple (c, val) such that $c \in C$ and val is a valid instance of its associated construct type's adt and obeys the restrictions ρ of c.

Intuitively, a construct type specificies a basic data structure. Thus, the value of a construct instance is an instance of its construct type's basic data structure. A description of composition constraints on the basic structures (*i.e.*, constructs) of a model is also given. For example, a particular construct might represent a collection of name-value pairs.

3.2 Example Description Language and Basic Structures

Here we introduce an example metamodel, *i.e.*, a concrete set of metamodel construct types, and demonstrate their use for describing data models. We also present a logic-based language for representing constructs and instances.

The construct types in this metamodel are collection, (specifically, $\texttt{set}_{ct}$, $\texttt{list}_{ct}$, $\texttt{bag}_{ct}$), $\texttt{struct}_{ct}$, and $\texttt{scalar}_{ct}$, representing collections, name-value pairs, and atomic data (like strings and integers), respectively. We should note that this choice of construct types is only one possible set of structures that are sufficient for the data models we describe in this paper. Figure 3 shows the XML, Relational, RDF, and Topic Map data model constructs expressed using our construct types (note that we use `COLL` to represent the collection construct types). Conformance is shown explicitly, *e.g.*, XML elements can conform to element types, Relational tables to schemes, RDF resources to resources, and Topic Map associations to topics.

We use the following predicates for representing the instance and conformance relationships of configurations.

```
ct-inst(c,ct).
conf(c1,c2).
c-inst(d,c).
d-inst(d1,d2).
```

The *`ct-inst`* predicate relates a construct (shown as `c`) to a construct type (shown as `ct`), which must be either $\texttt{set}_{ct}$, $\texttt{list}_{ct}$, $\texttt{bag}_{ct}$, $\texttt{struct}_{ct}$, or $\texttt{scalar}_{ct}$. The *`conf`*

[1] We are currently investigating methods for formally representing construct types and constructs, however, such a formal description is not required for the purpose of this paper.

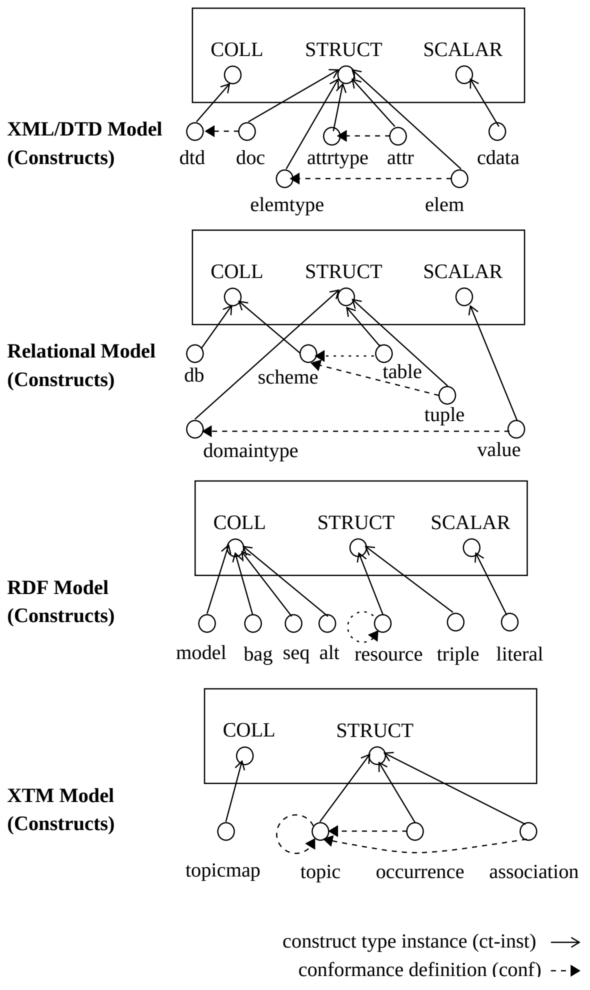


Fig. 3. Relational, RDF, and XTM data models described within the generic framework.

predicate specifies that a construct (shown as `c1`) can conform to another construct (shown as `c2`). The *`c-inst`* predicate relates an instance (shown as `d`) with a construct (shown as `c`). Finally, the *`c-inst`* predicate defines a conformance relationship between instances (shown as `d1` and `d2`).

We introduce the *`comp`* predicate for describing composition constraints over model constructs. The predicate can take the following forms.

```
comp(c1,setof(c1,...,cn)).
comp(c2,listof(c1,...,cn)).
comp(c3,bagof(c1,...,cn)).
comp(c4,structof(a1→c1,...,an→cn)).
cmp(c5,union(c1,...,cn)).
```

As shown, each kind of *`comp`* formula uses an additional formula to specify composition constraints, namely, *`setof`*, *`listof`*, *`bagof`*, or *`structof`*. For example, `c1` is assumed to be defined as *`ct-inst`*`(c1,`*set*$_{ct}$`)` whose instances are sets that can contain instances of constructs `c1`, `c2`, ..., `cn` (*i.e.*, the *`setof`* predicate defines the homogeneity of the construct's instances). We also permit the definition of convenience-constructs (as shown by `c5` above), which do not have corresponding *`ct-inst`* formulas and must be associated with a *`union`* predicate. Such constructs serve as a place-holder for the union of the corresponding constructs.

The last predicate we define is *`val`*, which gives the value of an instance. Each ground *`val`* formula must have an associated *`c-inst`* formula (which is also ground) of the appropriate form, as shown below.

```
val(d1,set(v1,...,vn)).
val(d2,list(v1,...,vn)).
val(d3,bag(v1,...,vn)).
val(d4,struct(a1=v1,...,an=vn)).
val(d5,scalar(v)).
```

To illustrate the definition language, Figures 4, 5, 6, and 7 define (simplified versions of) the XML with DTD, Relational, Topic Map, and RDF data models, respectively. Note that we assume `string` is a default scalar construct for each model and that the juxtaposition of entries is for exposition only. Finally, Figure 8 shows example instances for the XML model.

```
ct-inst(dtd,set_ct).                 ct-inst(elem-t,struct_ct).
ct-inst(attr-t,struct_ct).           ct-inst(pcdata,scalar_ct).
ct-inst(subelem-t,struct_ct).        ct-inst(attr-tset,set_ct).
ct-inst(doc,struct_ct).              ct-inst(elem,struct_ct).
ct-inst(attr,struct_ct).             ct-inst(attrset,set_ct).
ct-inst(nestedelem,struct_ct).       ct-inst(cdata,scalar_ct).
conf(doc,dtd).                       conf(elem,elem-t).
conf(attr,attr-t).

comp(dtd,setof(elem-t)).
comp(elem-t,structof(tag→string,attrtypes→attr-tset)).
comp(attr-t,structof(name→string)).
comp(subelem-t,structof(parent→elem-t,child→elem-t)).
comp(attr-tset,setof(attr-t)).
comp(doc,structof(root→elem)).
comp(elem,structof(tag→string,attrs→attrset)).
comp(attr,structof(name→string,val→cdata)).
comp(attrset,setof(attr)).
comp(node,union(elem,pcdata)).
comp(nestedelem,structof(parent→elem,child→node).
```

Fig. 4. Simplified description of the XML with DTD data model.

```
ct-inst(db,set_ct).                  ct-inst(scheme,set_ct).
ct-inst(table,struct_ct).            ct-inst(tuples,bag_ct).
ct-inst(tuple,set_ct).               ct-inst(fieldtype,struct_ct).
ct-inst(field,struct_ct).            ct-inst(domaintype,struct_ct).
ct-inst(value,scalar_ct).            conf(taple,scheme).
conf(tuple,scheme).                  conf(value,domaintype).

comp(db,setof(scheme,table)).
comp(scheme,setof(fieldtype)).
comp(table,structof(name→string,rows→tuples)).
comp(tuples,bagof(tuple)).
comp(tuple,setof(field)).
comp(fieldtype,structof(name→string,dom→domaintype)).
comp(field,structof(name→string,val→value)).
comp(domaintype,structof(name→string)).
```

Fig. 5. Simplified description of the Relational data model.

3.3 Specifying Constraints

In addition to describing composition of basic structures, we also provide a mechanism for defining the constraints of a data model. Namely, we leverage the logic-based description language for constraints by allowing constraint expressions (*i.e.*, rules). We denote constraints using the annotation `constraint`, *e.g.*, the following rule defines a conformance constraint over the Relational model.

```
ct-inst(tm,set_ct).              ct-inst(association,struct_ct).
ct-inst(topic,struct_ct).        ct-inst(assoc-mem,struct_ct).
ct-inst(topic-occ,struct_ct).    ct-inst(occurrence,struct_ct).
ct-inst(member,struct_ct).       ct-inst(mem-topic,struct_ct).
ct-inst(resource,struct_ct).     conf(topic,topic).
conf(association,topic).         conf(occurrence,topic).

comp(tm,setof(topic,assoc)).
comp(topic,structof(name→string)).
comp(topic-occ,structof(top→topic,occ→occurrence)).
comp(occ-val,union(resource,string)).
comp(occurrence,structof(val→occ-val)).
comp(association,structof()).
comp(assoc-mem,structof(assoc→association,mem→member)).
comp(member,structof(role→string)).
comp(resource,structof(uri→string)).
```

Fig. 6. Simplified description of the Topic Map (XTM) data model.

```
ct-inst(rdfmodel,set_ct).       ct-inst(uriref,struct_ct).
ct-inst(literal,scalar_ct).     ct-inst(blank,struct_ct).
ct-inst(alt,set_ct).            ct-inst(seq,list_ct).
ct-inst(bag,bag_ct).            ct-inst(triple,struct_ct).
conf(resource,resource).

comp(rdfmodel,setof(literal,blank,triple,coll)).
comp(resource,union(uriref,blank,coll,triple)).
comp(node,union(resource,literal)).
comp(uriref,structof(uri→string)).
comp(blank,structof()).
comp(coll,union(set,bag,alt)).
comp(alt,setof(node)).
comp(seq,listof(node)).
comp(bag,bagof(node)).
comp(triple,structof(pred→uriref,subj→resource,
                     obj→node)).
```

Fig. 7. Simplified description of the RDF data model.

(a).

```
<!ELEMENT activity (profession | hobbyist | ...)*>
<!ELEMENT profession (surgeon | professor | ...)*>
<!ELEMENT professor (#PCDATA)>
```

(b).

```
<activity>
   <profession>
      <professor>Lois</professor>
   </profession>
</activity>
```

(c).

```
c-inst(t,dtd).               val(t,set(et1,et2,et3)).
c-inst(t1,elem-t).           val(t1,struct(name='activity')).
c-inst(t2,elem-t).           val(t2,struct(name='profession')).
c-inst(t3,elem-t).           val(t3,struct(name='professor')).
c-inst(s1,subelem-t).        val(s1,struct(parent=t1,child=t2)).
c-inst(s2,subelem-t).        val(s2,struct(parent=t2,child=t3)).
c-inst(d, doc).              val(d,struct(root=e1)).
c-inst(e1,elem).             val(e1,struct(tag='activity')).
c-inst(e2,elem).             val(e2,struct(tag='profession')).
c-inst(e3,elem).             val(e3,struct(tag='professor')).
c-inst(n1,nestedelem).       val(n1,struct(parent=e1,child=e2)).
c-inst(n2,nestedelem).       val(n2,struct(parent=e2,child=e3)).
c-inst(l,pcdata).            val(l,scalar('Lois')).
c-inst(n3,nestedelem).       val(n3,struct(parent=e3,child=l)).
c-inst(d,t).                 d-inst(e1,t1).
d-inst(e2,t2).               d-inst(e3,t3).
```

Fig. 8. An example (a) XML DTD, (b) conforming document, and (c) both represented in the description language.

```
constraint: d-inst(X,Y) :-
   c-inst(X,table) & c-inst(Y,scheme) &
   not(d-inst(X,Yp) & Y≠Yp) &
   not(d-inst(Xp,Y) & X≠Xp) &
   val(X,B) & member(V,B) & d-inst(V,Y).
```

The constraint is read as: "X can conform to Y if X is a Table, Y is a Scheme, X doesn't conform to any other Schemes, Y doesn't have any other conforming Tables, and every Row in X conforms to Y." Note that specifying constraints in this way provides a powerful mechanism for describing complex data model descriptions.

4 Transforming Representation Schemes

In this section, we present a language for transforming representation schemes based on horn-clause logic rules, provide example mappings, and discuss higher-level transformations for simplifying the specification and implementation of complex transformations.

4.1 A Transformation Langauge

A transformation is applied to one or possibly more configurations in which the result is a new configuration. Here, we only consider the case where a single configuration is transformed. We define a transformation as a function $\sigma : M \times \mathcal{C} \rightarrow \mathcal{C}'$ where M is a set of *mapping rules*, $\mathcal{C}$ is the source configuration, and $\mathcal{C}'$ is the new configuration called the "destination" of the transformation. The transformation function σ computes the fix-point of the mapping rules applied to the source and destination configurations.

Mapping rules can take the following form:

$$\texttt{dp}_1 \;\&\; \ldots \;\&\; \texttt{dp}_n \;\texttt{:-}\; \texttt{p}_1 \;\&\; \ldots \;\&\; \texttt{p}_m,$$

where both `dp` and `p` are (a) formulas (using the predicates of Section 3) with ground literals or variables as arguments or (b) the formula *ID*(`I,id`). The *ID* predicate takes a list `I` of input constants and provides a constant (*i.e.*, `id`) that serves as a unique identifier for the input list. The *ID* predicate is implemented as a skolem function.[2] The syntax we use for mapping rules is shorthand for rules having a single, outer formula in the head of the program clause (via the `head` formula):

$$\textit{head}(\texttt{dp}_1, \ldots, \texttt{dp}_n) \;\texttt{:-}\; \texttt{p}_1 \;\&\; \texttt{p}_2 \;\&\; \ldots \;\&\; \texttt{p}_m.$$

Mapping rules are read in the normal way, *i.e.*, if each $\texttt{p}_i$ for $1 \leq i \leq n$ is true, then each $\texttt{dp}_j$ for $1 \leq j \leq m$ is true. Additionally, we permit two annotations on mapping rule formulas. The `add` annotation indicates that the formula be added to the destination configuration. The `dest` annotation matches formulas against those in the destination configuration (the source configuration is the defaul). We do not allow transformations to modify the source configuration.

[2] We use the abbreviation *ID*(`v1+v2+...,id`) to denote the list `[v1, v2, ...]`

```
% element types become schemes
add:c-inst(S,scheme) :-
  c-inst(ET,elem-t) & ID(ET,S).
% attribute types become field types
add:c-inst(FT,fieldtype) :-
  c-inst(AT,attr-t) & ID(AT,FT).
% elements become tuples
add:c-inst(T,tuple) :-
  c-inst(E,elem) & ID(E,T).
% attribute become fields
add:c-inst(F,field) :-
  c-inst(A,attr) & ID(A,F).
% cdata to values
add:c-inst(V2,value) :-
  c-inst(V1,cdata) & ID(V1,V2).
% sub element types to schemes
add:c-inst(S,scheme) :-
  c-inst(ST,subelem-t) & ID(ST,S).
% field types for sub element type schemes
add:c-inst(FT1,fieldtype) &
add:c-inst(FT2,fieldtype) :-
  c-inst(ST,subelem-t) &
  val(ST,struct(parent→P,child→C) &
  ID(ST+P,FT1) & ID(ST+C,FT2).
% nested elements become tuples
add:c-inst(T,tuple) :-
  c-inst(NE,nestedelem) & ID(NE,T).
% each scheme has a table
add:c-inst(T,table) :-
  dest:c-inst(S,scheme) & ID(S,T).
% tables conform to their schemes
add:d-inst(T,S) :-
  dest:d-inst(S,scheme) & ID(S,T).
% values conform to the cdatatype domain type
add:d-inst(V,cdatatype) :-
  dest:c-inst(V,value).
% tuples conform to their schemes
add:d-inst(TP,S) :-
  c-inst(E,elem) & ID(E,TP) & d-inst(E,ET) & ID(ET,S).
```

Fig. 9. Rules for a model-to-model transformation.

```
% create an empty scheme for each element type
add:val(S,set()) :-
  c-inst(ET,elem-t) & ID(ET,S).
% add a field type for each attribute type
add:val(FT,struct(name=N,dom=cdatatype)) :-
  c-inst(AT,attr-t) & val(AT,struct(name=N)) &
  ID(AT,FT).
% add a field type for element ids
add:val(FT,struct(name='id',dom=stringtype)) :-
  c-inst(ET,elemtype) & ID(ET+'id',FT).
% add field types to associated element type schemes
add:member(FT,FTSet) :-
  c-inst(ET,elem-t) &
  val(ET,struct(tagname=_,attrtypes=ATSet)) &
  val(ATSet,Set) & member(AT,Set) & ID(ET,S) &
  ID(AT,FT) & dest:val(S,FTSet).
% create an empty scheme for each sub-element type
add:val(S,set()) :-
  c-inst(ST,subelem-t) & ID(ET,S).
% create two field types for sub element type scheme
add:val(FT1,struct(name=PN,dom=stringtype)) &
add:val(FT2,struct(name=CN,dom=stringtype)) :-
  c-inst(ST,subelem-t) &
  val(ST,struct(parrenttype→P,childtype→C) &
  val(P,struct(tagname→PN,attrtypes→_) &
  val(C,struct(tagname→CN,attrtypes→_) &
  ID(P,FT1) & ID(C,FT2).
% create a table element type
add:c-inst(R,tuples) & add:val(R,bag()) &
add:val(T,struct(name=N, rows=R)) :-
  c-inst(ET,elem-t) &
  val(ET,struct(tagname→N,attrtypes→_) &
  ID(ET,S) & ID(S,T) & ID(T,R) .
% create a field for each attribute
add:val(F,struct(name→N,val→V)) :-
  c-inst(A,attr) & val(A, struct(name→N,val→C) &
  ID(A,F) & ID(C,V).
% create a field for each element as an id
add:val(F,struct(name→'id',val→V)) :-
  c-inst(E,elem) & ID(E+'id',F) & tostring(F, V).
...
```

Fig. 10. Structural rules for a model-to-model transformation.

4.2 An Example Model-to-Model Mapping

Figures 9 and 10 show a portion of a simple model-to-model mapping between XML and the Relational model using their descriptions from Section 3. Figure 9 shows instance and conformance rules and Figure 10 shows basic structure conversions. This mapping converts XML element types to Relational schemes and converts the rest of the data accordingly. Specifically, tables contain elements with their associated attribute values and a new table is created for each subelement-type relationship, which is where nested elements are stored. Figure 11 shows the tables that would result from example XML data. Note that we choose to skolemize each identifier to assure uniqueness in the destination (*e.g.*, the destination may contain other values prior to the transformation).

While model-to-model transformations are useful, there are a number of other transformations that can be supported in this framework, including schema-to-schema, model-to-schema (also call "modeling the model" [3]), and various mixtures of each [4,5].

4.3 Higher-Level Transformations

While mapping rules are expressed using a declarative language, they specify the low-level details of transformations. Such an approach is attractive since it enables extremely flexible transformations (*e.g.*, model-to-model, schema-to-schema, and arbitrary combinations), however, specifying mapping rules is not trivial. In the remainder of this section, we describe two approaches for higher-level rule specification (each of which builds on the current mapping rule approach), which we believe can make writing rules simpler. We briefly describe *transformation patterns* for capturing and re-using common transformation conventions and discuss semi-automatic *rule derivation*, which is enabled through our generic framework.

We view a transformation pattern as a mapping rule abstraction, *i.e.*, a transformation pattern is a parameterized specification taking mapping rules as input to create a more complex transformation. In this way, patterns are analogous to higher-order functions like `map` and `fold` in functional programming languages. In general, a pattern can be used as a runtime operation where a set of arguments is provided to perform the concrete transformation, or, as a mapping rule generator, *i.e.*, given a set of arguments, the pattern is used to generate (one or more) concrete mapping rules.

```
<!ELEMENT professor (person)>
<!ATTLIST professor dept CDATA #REQUIRED>
<!ELEMENT person EMPTY>
<!ATTLIST person name CDATA #REQUIRED>

<professor dept=''CSE''>
   <person name=''Lois''/>
</professor>
```

Professor Table

id	*dept*
e1	"CSE"

Person Table

id	*name*
e2	"Lois"

Professor-Person Table

professor	*person*
e1	e2

Fig. 11. Results from the XML to Relational transformation.

For example, there are often many ways to transform between models allowing multiple levels of conformance to models permitting only a single level. One such convention is to flatten multiple levels of conformance, making the conformance implicit in the destination. A different convention is to represent source instance-of relationships as hierarchical relationships in the destination (again, conformance becomes implicit). Examples are shown in Figure 12 where the source is a Topic Map and the target a Relational database (for the flattening approach) and an XML document (for the hierarchy approach). The source has three levels of conformance: topic "Lois" is an instance of topic "Professor"; topic "Professor" is an instance of topic "Profession"; and topic "Profession" is an instance of topic "Activity."[3] In the case of XML, the hierarchical relationship is modeled as nested elements, and for the Relational example, each table represents a path of the original conformance relationship.

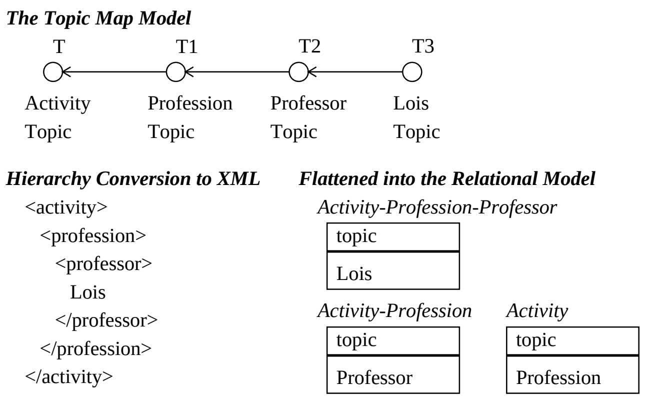


Fig. 12. An example of flattening and hierarchy conversion of the Topic Map model.

Figure 13 shows an example transformation pattern called `hierarchy-convert` for generically applying the hierarchy conversion. (The language shown for describing patterns is for exposition only; we are currently developing a pattern language for transformations.) The pattern takes five arguments: the construct instance `T` serving as the top item of the multi-level conformance relationship (*e.g.*, "Activity" in Figure 12); a mapping rule $\mathtt{M}_1$ to map source items to target items (*e.g.*, topics to XML elements); a special mapping rule $\mathtt{M}_2$ for converting items at the leaf level (*e.g.*, instead of mapping topics to elements, we would map topics to XML element content); and mapping rules $\mathtt{A}_1$ and $\mathtt{A}_2$ for attaching the hierarchical relationships in the destination.

The `hierarchy-convert` pattern is broken into two sub-rules. The first rule identifies items above the leaf conformance items (*e.g.*, the "Activity" and "Profession" topics) and converts them to target items (using $\mathtt{M}_1$). The second rule is similar, but for leaf nodes. In our example, the second rule would convert the topic "Lois" into XML content (using $\mathtt{M}_2$), and attach it to the "Professor" element (using $\mathtt{A}_2$).

We see a number of potential patterns as useful, *e.g.*, a simple class of patterns are those that convert between construct

```
pattern:hierarchy-convert(T,M₁,M₂,A₁,A₂) =
  % convert inter-nodes T and T1
  map T to S and T1 to S1 using rule M₁ &
  attach S1 to S using rule A₁ :-
    d-inst(T3,T2) & d-inst(T1,T) &
    c-inst(T,C) & c-inst(T1,C) & c-inst(T2,C).
  % convert last node
  map T2 to S2 using rule M₁ &
  map T3 to S3 using rule M₂ &
  attach S3 to S2 using rule A₂ :-
    d-inst(T3,T2) & not d-inst(T4,T3) &
    c-inst(T2,C) & c-inst(T3,C) & c-inst(T4,C).
```

Fig. 13. Example of the conformance flattening pattern.

types. That is, patterns for converting lists to sets and bags, flattening nested sets, merging sets, lists, bags, and structs, and converting a group of structs into sets, and so on.

Patterns help make it easier to write rules by enabling the reuse of commonly used transformations. Another approach for simplifying rule writing is semi-automatic rule derivation. The goal is to generate the structural mapping rules (like those in Figure 10) whenever possible from the simpler construct level rules (such as those in Figure 9) by leveraging the composition constraints of constructs.

For example, consider the excerpt of XML and Relational models shown in Figure 14. The following mapping rule specifies a construct transformation between XML element types and Relational schemes.

c-inst(`ET,scheme`) :- *c-inst*(`ET,elem-t`).

Based on the description and constraints of the source model, we can conclude the following: (1) every `scheme` must have exactly one `table` (and vice versa), (2) every `table` is a bag of `tuple`'s, (3) every `tuple` must conform to its `tables` conforming `scheme`, and (4) more than one `elem` can conform to an `elem-t`.

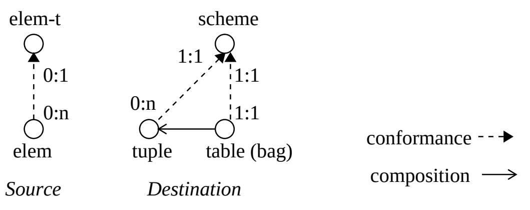


Fig. 14. Source and destination constructs for rule derivation.

From the above mapping rule requesting each `elem-t` be converted to a `scheme` and the above constraints, it is reasonable to consider the following mappings.

1. Each `elem` should be mapped to a `tuple` since many `elem`'s can be associated with each `elem-t`, and it is only through `tuple`'s that multiple items can be associated with a given `table`.
2. A `table` should be created for each converted `elem-t` (*i.e.*, each new `scheme`).
3. The corresponding `table` (for the new `scheme`) should contain the converted `tuple` items.

Note that to generate these rules, we can employ structural transformation patterns, *e.g.*, to convert the source *d-inst*

[3] To see this is correctly modeled, notice that "Lois" is an instance of a "Professor," but not an instance of a "Profession."

relations into a bag (`table`) of the appropriate `tuples`, and so on. Although a simple example, the derived rules are equivalent to those used to map element types to schemes in the transformation of Section 4.2. Thus, combining patterns and constraint-based rule derivations can ease the task of rule specification, both of which are enabled by the uniform framework. We intend to investigate both patterns and rule derivation further.

5 Related Work

In this paper, we extent our earlier work [4,5], which introduced the use of a uniform, RDF-based representation scheme allowing model, schema, and instance information to be represented in triples. First, the metamodel is significantly more complex. It also has a relatively complete set of structural primitives, including most database modeling constructs: *struct* (much like an object in object-oriented models) and explicit collection primitives *set*, *bag*, and *list*. More importantly, these basic structures can be freely composed as appropriate for the data model being described using the constraint facility. We also contribute a clear separation of basic structures (*i.e.*, construct types) from instance-of and conformance relationships—this separation differs from RDF and Topic Map models each having only a single version. Thus, the choice of construct types (*i.e.*, metamodel) is extensible and the architecture can be exploited to provide high-level transformation rules (*i.e.*, patterns and constraint-based rule derivation).

The metamodel architecture we propose is the only approach we know of that explicitly models conformance. Existing metamodel approaches [6,7,8,9,10,11] primarily focus on solving database interoperability issues and assume (and exploit) schema-first data models. Typically, transformation is defined at the data model level or at the schema level (but not both). The transformation approach we propose enables mappings at differing levels of abstraction, *e.g.*, model-to-model, model-to-schema, and combinations of these. In addition, we permit partial mappings, *i.e.*, we do not require complete (or one-to-one) transformations.

The use of logic-based languages for transformation has been successfully applied [10,12], *e.g.*, WOL [12] is a schema transformation language that uses first-order logic rules for both schema transformations and specifying (schema) constraints. Similar to WOL, we use a logic-based approach for specifying constraints.

Finally, a number of recent model-to-model and model-to-schema mappings (without a formal description of the transformation) have been defined between XML and Relational models [13] and between Topic Maps and RDF [3,14]. We believe using a formal tranformation language can benefit both defining such mappings, and implementing them.

6 Summary and Future Work

The three-level architecture we propose has the following advantages. It allows the data model designer to explicitly define when and how one construct conforms to another (through conformance links). It can be used to describe models that require schema, don't require schema, permit multiple schemas, allow multiple levels of schema-instance connections, and include any or all of these possibilities. It introduces four explicit relationships, *ct-inst*, *conformance*, *c-inst*, and *d-inst*, which can be easily exploited in a transformation language as well as by other tools such as a query language or a browser. Finally, it provides orthogonal, complementary specification of conformance and instance-of relationships versus composition and constraints for describing actual data structures that exist in the information representation.

We believe patterns and constraint-based transformation offer powerful mechanisms for specifying mapping rules and intend to further investigate their use. We are also exploring "in-place transformation" for performance optimization, *i.e.*, transforming information without first converting it to an intermediate representation. Thus, mappings are implemented by calls to the associated source and destination native technology. Such native rule generation is made possible by the specification of the data model using the framework we have presented.

References

1. Marshall, C., III, F.S., Coombs, J.: Viki: Spatial hypertext supporting emergent structure. In: European Conf. on Hypertext Technology (ECHT '94), ACM (1994) 13–23
2. Nanard, J., Nanard, M.: Should anchors be typed too? An experiment with macweb. In: Proc. of Hypertext. (1993) 51–62
3. Moore, G.: RDF and TopicMaps: An exercise in convergence. In: XML Europe. (2001)
4. Bowers, S., Delcambre, L.: Representing and transforming model-based information. In: Proc. of the First Inter. Workshop on the Semantic Web. (2000)
5. Bowers, S., Delcambre, L.: A generic representation for exploiting model-based information. In: ETAI Journal. Volume 6. (2001)
6. Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: EDBT. Volume 1057 of Lecture Notes in Computer Science. (1996) 79–95
7. Atzeni, P., Torlone, R.: A unified framework for data translation over the web. In: Proc. of WISE. IEEE Computer Society Press (2001)
8. Barsalou, T., Gangopadhyay, D.: M(DM): An open framework for interoperation of multimodel multidatabase systems. In: Proc. of ICDE. (1992) 218–227
9. Christophides, V., Cluet, S., Siméon, J.: On wrapping query languages and efficient XML integration. In: Proc. of SIGMOD. (2000) 141–152
10. McBrien, P., Poulovassilis, A.: A uniform approach to intermodel transformations. In: Proc. of CAiSE'99. Volume 1626 of Lecture Notes in Computer Science. (1999) 333–348
11. Papazoglou, M., Russell, N.: A semantic meta-modeling approach to schema transformation. In: Proc. of CIKM, ACM (1995) 113–121
12. Davidson, S., Kosky, A.: WOL: A language for database transformations and constraints. In: Proc. of ICDE. (1997) 55–65
13. Bohannon, S., Freire, J., Roy, P., Siméon, J.: From xml schema to relations: A cost-based approach to xml storage. In: ICDE. (2002)
14. Lacker, M., Decker, S.: On the integration of Topic Maps and RDF. In: Proc. of the 1st International Semantic Web Working Symposium (SWWS '01). (2001)

Tracking Changes in RDF(S) Repositories

Atanas Kiryakov and Damyan Ognyanov

OntoText Lab, Sirma AI EOOD,
38A Chr. Botev blvd, 1000 Sofia, Bulgaria {naso, damyan}@sirma.bg

Abstract. The real-world knowledge management applications require administrative features such as versioning, fine-grained access control, and meta-information to be supported by the back-end infrastructure. Those features together with the needs of the ontology maintenance and development process raise the issue of tracking changes in knowledge bases. Part of the research presented is as basic as defining the rules of the game, the proper formal models to build upon – what to count as a change and what to ignore, how to represent and manage the tracking information. A number of more 'technical' issues such as tracking changes in imported and inferred knowledge are also discussed. The implementation is a part of the ontology middleware module developed under the On-To-Knowledge project where it is implemented as extension of the Sesame RDF(S) repository.

1 Introduction

The ontology middleware can be seen as an 'administrative' software infrastructure that makes the rest of the modules in a KM toolset easier for integration in real-world applications. The central issue is to make the tools available to the society in a shape that allows easier development, management, maintenance, and use of middle-size and big knowledge bases[1]. The following basic features are considered:

– Versioning (tracking changes) of knowledge bases;
– Access control (security) system;
– Meta-information for knowledge bases.

These three aspects are tightly interrelated among each other as depicted on the following scheme.

The composition of the three functions above represents a Knowledge Control System (KCS) that provides the knowledge engineers with the same level of control and manageability of the ontology in the process of its development and maintenance as the source control systems (such as CVS) provide for the software. However, KCS is not only limited to support the knowledge engineers or developers – from the perspective of the end-user applications, KCS can be seen as equivalent to the database security, change tracking (often called cataloguing) and auditing systems. A KCS is carefully designed to support these two distinct use cases.

Further, an ontology middleware system should serve as a flexible and extendable platform for knowledge management solutions. It should provide infrastructure with the following features:

– A repository providing the basic storage services in a scalable and reliable fashion. Such example is the Sesame RDF(S) repository, [2];
– Multi-protocol client access allowing different users and applications to use the system via the most efficient transportation media;
– Knowledge control – the KCS introduced above;
– Support for plugable reasoning modules suitable for various domains and applications. This ensures that within a single enterprise or computing environment one and the same system may be used for various purposes (that require different reasoning services) so providing easy integration, interoperability between applications, knowledge maintenance and reuse.

[1] See the next sub-section for discussion on ontology vs. instance data vs. knowledge base.

In the rest of this introductory section we define better the scope of our research and the terminology used. Section 2 is dedicated to an overview on tracking changes in RDF(S) repositories – related work and principles. The design and implementation approach of the change tracking and versioning is presented in Section 3 and Section 4 covers the meta-information and it's tracking followed by an implementation approach and formal representation respectively in Sections 5 and 6. Future work and conclusion are presented in the last section.

The work presented here was carried as part of the On-To-Knowledge project. The design and implementation of our ontology middleware implementation is just an extension of the Sesame architecture (see [2]) that already covers many of the desired features. Earlier stage of the research is presented in bigger details in [7] where the reader can find more about the Access control system (security), which is out of the scope of this paper.

1.1 Scope, Ontologies vs. Knowledge Bases

A number of justifications in the terminology are necessary. An almost trivial but still relevant question is 'What the KM tools support: ontologies, data, knowledge, or knowledge bases?' Due to the lack of space we are no going in to comment this basic notions here. A simple and correct answer is 'All of this'. The ontology middleware module extends the Sesame RDF(S) repository that affects the management of both ontologies and instance data in a pretty much unified fashion.

For the purpose of compliance with Sesame, here the term repository will be used to denote a compact body of knowledge that could be used, manipulated, and referred as a whole. Such may contain (or host) both ontological assertions and instance data.

2 Overview

The problem for tracking changes within a knowledge base is addressed in this section. It is important to clarify that higher-level evaluation or classification of the updates (considering, for instance, different sorts of compatibility between two

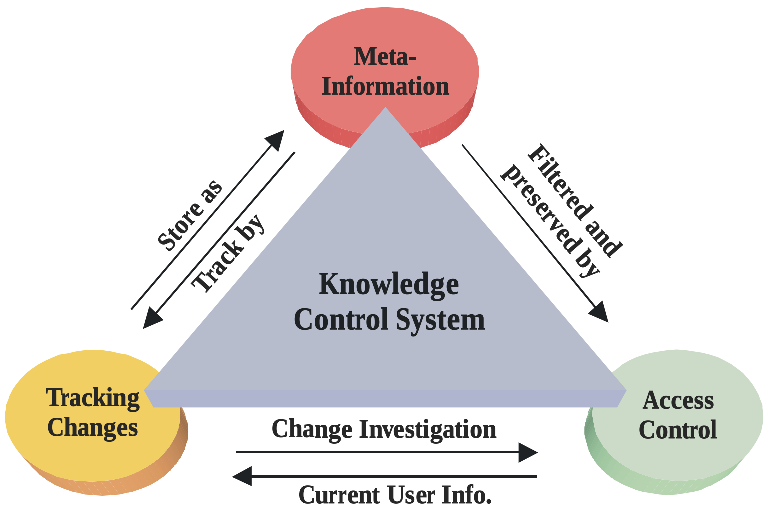


Fig. 1. Knowledge Control System

states or between a new ontology and old instance data) is beyond the scope of this work. Those are studied and discussed in [3], sub-section 2.2. The tracking of the changes in the knowledge (as discussed here) provides the necessary basis for further analysis. For instance, in order to judge the compatibility between two states of an ontology a system should be able to at least retrieve the two states and/or the differences between them.

In summary, the approach taken can be shortly characterized as 'versioning of RDF on a structural level in the spirit of the software source control systems'.

2.1 Related Work

Here we will shortly review similar work, namely, several other studies related to the management of different versions of a complex objects. In general, although some of the sources discuss closely related problems and solutions, there is not one addressing ontology evolution and version management in a fashion that allows granularity down to the level of statements (or similar constructs) and capturing interactive changes in knowledge repositories such as asserting or retracting statements.

One of the studies that provide a methodological framework pretty close to the one need here is [8]. The authors model a framework, which is designed to handle the identification, control, recording, and tracking of the evolution of software products, objects, structures, and their interrelationships. The paper investigates the different models and versioning strategies for large scale software projects and present a way to express the meta-information and the impact of a single change over the various components of the project in RDF(S) – in this case used just for representation of the related meta-information, the objects being tracked are from the software domain.

Database schema evolution and the tasks related to keeping schema and data consistent to each other can be recognized as very similar to ours. A detailed and pretty formal study on this problem can be found in [4] – it presents an approach allowing the different sorts of modifications of the schema to be expressed within suitable description logic. More detailed information about the reasoning and other related tasks could be found in [5].

Another study dealing with the design of a framework handling database schema versioning is presented in [1]. It is similar to [4] and [5] and can be seen as a different approach of handling the changes of the evolving object and the process of class evolution.

Probably the most relevant work was done under the On-To-Knowledge project – among the reports concerning various aspects of the knowledge management, most relevant is [3], mentioned earlier in this section.

3 Versioning Model for RDF(S) Repositories

A model for tracking of changes, versioning, and meta-information for RDF(S) repositories is proposed. To make it more explicit (i) the knowledge representation paradigm supported is RDF(S) and (ii) the subject of tracking are RDF(S) repositories – independently from the fact if they contain ontologies, instance data, or both. The most important principles are presented in the next paragraphs.

VPR1: The RDF statement is the smallest directly manageable piece of knowledge.

Each repository, formally speaking, is a set of RDF statements (i.e. triples) – these are the smallest separately manageable pieces of knowledge. There exist arguments that the resources and the literals are the smallest entities – it is true, however they cannot be manipulated independently. It is the case that none of them can independently 'live' in a repository because they always appear as a part of a triple and never independently. The moment when a resource was added to the repository may only be defined indirectly as the same as 'the moment when the first triple including the resource was added'. Analogously a resource may be considered removed from a repository when the last statement containing it gets out. To summarize, there is no way to add, remove, or update (the description of) a resource without also changing some statements while the opposite does not hold. So, the resources and the literals from a representational and structural point of view are dependent on the statements.

VPR2: An RDF statement cannot be changed – it can only be added and removed.

As far as the statements are nothing more than triples, changing one of the constituents, just converts it into another triple. It is because there is nothing else but the constituents to determine the identity of the triple, which is an abstract entity being fully defined by them. Let us take for instance the statement `ST1=<A, PR1, B>` and suppose B is a resource, i.e. an URI of resource. Then `ST1` is nothing more but a triple of the URIs of A, PR1, and B – if one of those get changed it will be already pointing to a different resource that may or may not have something in common with the first one. For example, if the URI of A was `http://x.y.z/o1#A` and it get changed to `http://x.y.z/o1#C` then the statement `ST2=<C,PR1,B>` will be a completely different statement.

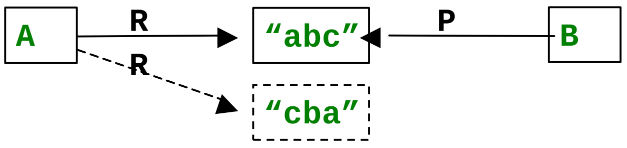
Further, if the resource pointed by an URI gets changed two cases could be distinguished:

– The resource is changed but its meta-description in RDF is not. Such changes are outside the scope of the problem for tracking changes in formally represented knowledge, and particularly in RDF(S) repositories.
– The description of the resource is changed – this can happen iff a statement including this resource get changed, i.e. added or removed. In such case, there is another statement affected, but the one that just bears the URI of the same resource is not.

There could be an argument, that when the object of a triple is a literal and it gets changed, this is still the same triple. However, if there is for instance statement `<A,R,"abc">` and it get changed to `<A,R,"cba">`, the graph representation shows that it is just a different arc because the new literal is a new node and there could be other statements (say, `<B,P,"abc">`) still connected to the note of the old literal. As a consequence here comes the next princple:

VPR3: The two basic types of updates in a repository are addition and removal of a statement

In other words, those are the events that necessarily have to be tracked by a tracking system. It is obvious that more event types such as replacement or simultaneous addition of a number of statements may also be considered as relevant for an RDF(S) repository change tracking system. However, those can all be seen as composite events that can be modeled via sequences of additions and removals. As far as there is no doubt that the solution proposed should allow for tracking of composite events (say, via post-processing of sequence of simple ones), we are not going to enumerate or specify them here.



VPR4: Each update turns the repository into a new state

Formally, a state of the repository is determined by the set of statements that are explicitly asserted. As far as each update is changing the set of statements, it is also turning the repository into another state. A tracking system should be able to address and manage all the states of a repository.

3.1 History and Versions

Some notes and definitions that complement the above stated principles are presented below.

History, Passing through Equivalent States. The history of changes in the repository could be defined as sequence of states, as well, as a sequence of updates, because there is always an update that turned repository from one state to the next one. It has to be mentioned that in the history, there could be a number of equivalent states. It is just a question of perspective do we consider those as one and the same state or as equivalent ones. Both perspectives bear some advantages for some applications. We accepted that there could be equivalent states in the history of a repository, but they are still managed as distinct entities. Although it is hard to provide formal justification for this decision the following arguments can be presented:

– For most of the applications it is not typical a repository to pass through equivalent states often. Although possible, accounting for this phenomenon does not obviously worth taking into account that finding (or matching) equivalent states could be a computationally very heavy task.
– It is the case that, if necessary, equivalent states could be identified and matched or combined via post-processing of a history of a repository.

Versions are labeled states of the repository. Some of the states of the repository could be pointed out as versions. Such could be any state, without any formal criteria and requirements – it completely depends on the user's or application's needs and desires. Once defined to be a version, the state becomes a first class entity for which additional knowledge could be supported as a meta-information (in the fashion described below.)

4 Meta-Information

Meta-information is supported for the following entities: resources, statements, and versions. As far as DAML+OIL ontologies are also formally encoded as resources (of type daml:Ontology) meta-information can be attached to them as well.

4.1 Model and Representation of the Meta-Information

We propose the meta-information to be modeled itself in RDF – something completely possible taking into account the unrestricted meta-modeling approach behind RDF(S). A number of objections against such approach can be given:

– It increases the number of meta-layers and so it makes the representation more abstract and hard to understand. However, adding meta-information always requires one more layer in the representation, so, making it via extension of the same primitives used for the ‘real data’ (instead of defining some new formalization) can even be considered as a simplification.
– It makes possible confusion and may introduce technical difficulties, say, because of intensive use of heavy expressive means such as reification.

The schema proposed below handles in some degree these problems and provides number advantages:

– It is probably the most typical role of RDF to be used for encoding of meta-information
– One and the same technology can be used for viewing, editing, and management of both knowledge and meta-information. Any RDF(S) reasoners and editors will be able to handle meta-information without special support for it.
– Queries including both knowledge and meta-information will be pretty straightforward. So, lookup of knowledge according to conditions involving both meta-information and ‘real’ knowledge is possible. Imagine a situation, when a complex ontology is being developed and there is meta-information supporting this process, say, a meta-property ‘Status’ (with possible values ‘New’, ‘Verified against the ontology’, ‘Verified against the sample data’, ‘Done’) being attached to each class. Then a lookup of all classes that are subclasses of C and have status ‘New’ will be just a typical query against the RDF(S) repository.
– Representing the meta-information as RDF could be done in a flexible way that allows it to be customized for the specific needs of the use case.

4.2 Tracking Changes in the Meta-Information

An important decision to be taken is whether changes in the meta-information should be tracked. The resolution proposed here is: Changes in the meta-information should be considered as regular changes of the repository, so, they turn it from one state to another. Here are few arguments backing this position:

– There are number of cases when the only result of a serious work over an ontology is just a single change in the meta-information. Let is use again the example with the ‘Status’ meta-property for classes (described above.) The result of a complex analysis of the coherence of a class definition may result just in changing the status from ‘New’ to one of the other values. In such case, although there is no formal change in the ‘real’ data, something important get changed. From an ontology development and maintenance point of view it is extremely important tracking of such changes to be possible.
– If necessary, it is possible appropriate analysis to be made so that changes that affect only meta-information to be ignored. This way both behaviors can be achieved. In case of the opposite decision (not to track changes in meta-information), no kind of analysis can reconstruct the missing information.
– An analogy with the software source control systems may also provide additional intuition about this issue. If we consider the comments in the software code as a meta-information, it becomes clear that the source control systems definitely account the changes in the meta-information as equal to the ‘real’ changes in the code.

5 Implementation Approach

Let us first propose the technical schema for tracking changes in a repository. For each repository, there is an *update counter (UC)* – an integer variable that increases each time when the repository is updated, that in the basic case means when a statement get added to or deleted from the repository. Let us call each separate value of the UC *update identifier, UID*. Then for each statement in the repository the UIDs when it was added and removed will be known – these values determine the ‘lifetime’ of the statement. It is also the case that each state of the repository is identified by the corresponding UID.

The UIDs that determine the ‘lifetime’ of each statement are kept, so, for each state it is straightforward to find the set of statements that determine it – those that were ‘alive’ at the UID of the state being examined. As far as versions are nothing more than labeled states, for each one there will be also UID that uniquely determines the version.

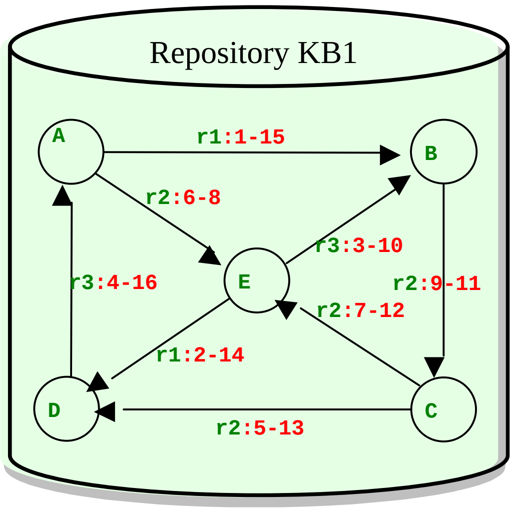
The approach could be demonstrated with the sample repository KB1 and its ‘history’. The repository is represented as a graph – each edge is an RDF statement which lifetime is given separated with semicolons after the property name. The history is presented as a sequence of events in format

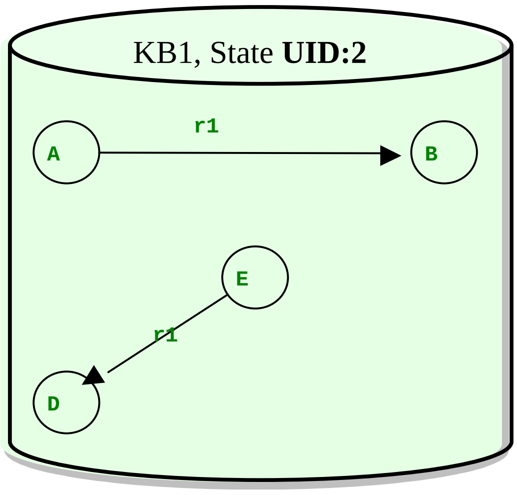
```
UID:nn {add|remove} <subj, pred, obj>
```

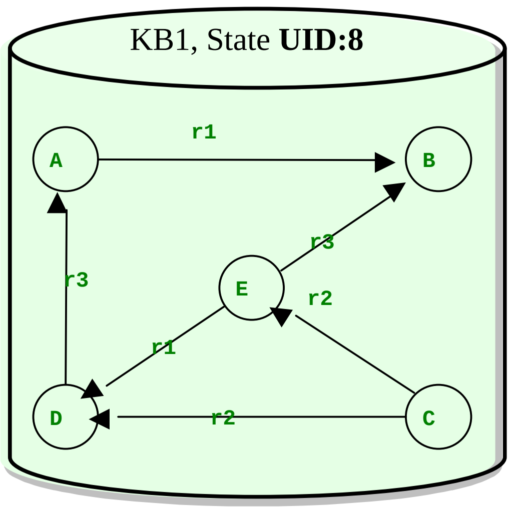
```
History:
UID:1 add <A, r1, B>
UID:2 add <E, r1, D>
UID:3 add <E, r3, B>
UID:4 add <D, r3, A>
UID:5 add <C, r2, D>
UID:6 add <A, r2, E>
UID:7 add <C, r2, E>
UID:8 remove <A, r2, E>
UID:9 add <B, r2, C>
UID:10 remove <E, r3, B>
UID:11 remove <B, r2, C>
UID:12 remove <C, r2, E>
UID:13 remove <C, r2, D>
UID:14 remove <E, r1, D>
UID:15 remove <A, r1, B>
UID:16 remove <D, r3, A>
```

Here follow two ‘snapshots’ of the states of the repository respectively for UIDs [2] and [8].

It is an interesting question how we handle in the above model, multiple additions and removals of one and the same statement, which in a sense periodically appears and disappears form the repository. We undertake the approach to







consider the appearance of such statement as separate statements, because of reasons similar to those presented for the support of distinguishable equivalent states of the repository.

5.1 Batch Updates

We call *batch update* the possibility the update counter of the repository to be stopped, so not to increment its value for a number of consequent updates. This feature is important for cases when it does not make sense the individual updates to be tracked one by one. Such example could be an assertion of a DAML+OIL element that is represented via set of RDF statements none of which can be interpreted separately.

Another example for a reasonable batch update would be an application that works with the repository in a transactional fashion – series of updates are bundled together, because according to the logic of the application they are closely related. Finally, batch updates can also be used for file imports (see subsection 5.4.).

5.2 Versioning and Meta-information for Imported Statements

New statements can appear in the repository when an external ontology is imported in the repository either by `xmlns:prefix="uri"` attribute of an XML tag in the serialized form of the ontology either by `daml:imports` statement found in the header of a DAML+OIL ontology. In each of those cases the statements imported in the repository are treated as read-only and thus the users cannot change them. All these statements will be added and removed to/from the repository simultaneously with the statement that causes their inference or import. An additional note about the imported statements related to the security: these statements should be recognized as external, and not belonging to the repository and thus we can avoid the application of the security policies to them. Meta-information may not be attached to such statements.

5.3 Versioning and Meta-information for Inferred Statements

There are cases when addition of a single statement in the repository leads to the appearance of several more statements in it. For example, the addition of the statement `ST1=<B,rdfs:subClassOf, C>` leads to the addition of two new statements `ST2=<B, rdf:type, rdfs:Class>` and `ST3=<C,rdf:type, rdfs:Class>`. This is a kind of simple inference necessary to ‘uncover’ knowledge that is implicit but important for the consistency of the repository. There are number of such inferences implemented in Sesame down the lines of [10].

The question about the lifetime of such inferred statements is far not trivial. Obviously, they get born when inferred. In the simplest case, they should die (get removed) together with the statement that caused them to be inferred. However, imagine that after the addition of ST1 in the repository, there was another statement added, namely `ST4=<B, rdfs:subClassOf, D>`. As far, as `ST2` is already in the repository only

`ST5=<D,rdf:type, rdfs:Class>` will be inferred and added. Now, imagine `ST1` is deleted next while ST4 remains untouched. Should we delete `ST2`? It was added together with `ST1` on one hand, but on the other it is also 'supported by' `ST4`. One approach for resolving such problems is the so-called 'truth maintenance systems' (TMS) – basically, for each statement information is being kept about the statements or expressions that 'support' it, i.e. such that (directly) lead to its inference. Sesame currently implements such a TMS.

Suppose, there is a TMS working in Sesame (because it could be 'switched off' for performance reasons), the tracking of the inferred statements is relatively easy. When the TMS 'decides' that an inferred statement is not supported anymore, it will be deleted – this is the natural end of its lifetime. It will be considered as deleted during the last update in the repository, which automatically becomes a sort of batch update (if it is not already.)

This is the place to mention that the pragmatics of the remove operation in an RDF(S) repository is a bit not obvious. When a statement is removed this only means that it is not explicit any more. However, if it follows from other statements, it effectively remains in the repository and there is no way to remove it, without removing all of its 'supporters'. Imagine a situation when one statements was inferred on update U1, next explicitly asserted on updated U4, next deleted at U6, but still supported by other sentences until U9. Than the life time of the statement is U1-U9.

For many applications, the 'explicitness' of the statements bear significant importance, so, in our implementation we keep track for this – for each state it is not only possible to uncover the statements that were 'alive', but also which of them were explicit.

As with the imported statements, meta-information may not be attached to inferred statements. The security restrictions towards inferred statements can be summarized as follows:

- Inferred statements may not be directly removed;
- A user can read an inferred statement iff s/he can read one of the statements that support it.
- The rights for adding statements are irrelevant – a user may or may not be allowed to add a statement independently from the fact is it already inferred or not.

5.4 Versioning of Knowledge Represented in Files

The issues concerning the work with knowledge represented in files and its versioning can be discussed in two main topics – each of them presenting a different involvement of the content of the files.

The first one is the case when the Sesame uses a specific storage and inference layer (SAIL) to access knowledge directly from the underlying file – so files used for persistency instead of, say, relational database. In such case we cannot control the appearance and disappearance of distinct statements, which easily can happen independently from Sesame. The knowledge control system (KCS) presented here is not applicable for such SAILs.

The second case is to import knowledge into the Sesame from files - one of the typical usage scenarios. The first step than is to convert the file F into a set of statements FS, which also includes the inferred ones. Next, the appropriate changes are made in the repository within a single batch update (see subsection 5.1.) Three different modes for populating repository from files are supported:

Re-initializing the existing content of the repository is cleared and the set of statement FS is added. No kind of tracking or meta-information is preserved for the statements that were in the repository before the update. This is equivalent to Clear followed by Accumulative import;

Accumulative FS is added to the repository, it actually means that the statements from FS that are already in the repository are ignored (any tracking and meta-information for them remains unchanged) and the rest of the statements are added. This type of import has to be used carefully, because it may lead to inconsistency of the repository even if its previous state and the file were consistent on their own;

Updating after the import the repository contains only the statements form the file, the set FS (as in the re-initializing mode). The difference is that the statements from the repository that were not in FS are deleted but not cleared, i.e. after the update, they are still kept together with their tracking and meta-information. The statements from FS that were not already in the repository are added.

The Updating import mode is the most comprehensive one and allows the repository to be used to track changes in a file that is being edited externally and 'checked-in' periodically. This can also be used for outlining differences between versions or different ontologies represented in files.

5.5 Branching Repositories

Branching of states of repositories is possible. In this case a new repository is created and populated with a certain state of an existing one – the state we want to make branch of. When a state is getting branched, it will automatically be labeled as a version first. The appropriate meta-information indicating that this version was being used to create a separate branch of the repository into a new one will be stored.

As it can be expected, no kind of operations with the branch of the repository will affect the original one. Branches have to be used, for instance, in cases when the development of an ontology have to be split into two separate processes or threads. A typical situation when this is necessary is when an old version has to be supported (which includes making small maintenance fixes) during the development of a new version. The state, which is used in production, can be branched and the development of the new one can take place in the branch while at the same time, the old version can still be supported and updated.

5.6 Controlling the History

The possibility for tracking changes in a repository and gathering history has an important consequence: the history has to be controlled! The most important reasons for this and the appropriate mechanisms are discussed here.

Reason 1: The volume of the data monotonously grows.

As far as nothing is really removed from the repository (the statements are only marked as 'dead') all the statements that were ever asserted in the repository together with their tracking and meta-information can be expected to be preserved forever. This way the volume of the data monotonously grows with each update.

Reason 2: The history may need refinement. The automatic tracking of the changes allows a great level of control, however the fine-grained information may also be obsolete or confusing. For instance, after the end of a phase of development of an ontology, the particular updates made in the process may become unimportant – often it is the case that finally the differences with a specific previous state (say, a version) are those that count.

Therefore the following methods control over the tracking information are implemented:

Clear the history before certain state all the statements died before this state (say S1), together with any kind of tracking and meta-information about them will be permanently removed from the repository. The same applies for the labeled versions before this state. All values of the update counter form a kind of 'calendar' and all changes are logged against it. There will be two options for managing the tracking information for statements that were 'born' before S1 and died after it. Under the first option, they will be stated to be born at a special moment 'before the Calendar' and all calendar records before S1 will also be deleted. Under the second option, the 'calendar' will be preserved, so no changes will be introduced to the tracking information for those statements that remain in the repository and the Calendar will be cleared from all the records that are not related to such statements.

Aggregate updates a number of sequential updates (say, between UID1 and UID2) to be merged and made equivalent to specified one, say UID3. In this case, all references to UIDs between UID1 and UID2 will be replaced with UID3, which may or may not be equal to UID1 or UID2.

6 Formal Representation of the Meta-Information

All the Knowledge Control System (KCS) related information would be represented in RDF according to a schema that could be found at: `http://www.ontotext.com/otk/2002/03/kcs.rdfs`. That includes tracking, versioning, and security information as well as user-defined meta-information. It is important to acknowledge that although this schema provides a well-structured conceptual view to the meta-information, its support by a repository is not be implemented directly after this schema because of obvious performance problems. So, the schema presents the way this information can be imported, exported, and accessed via RQL queries. It also facilitates good formal understanding of the supported model.

The basic idea is that all the meta-information is encoded via kind of special, easily distinguishable, properties – namely such defined as sub-properties of a kcs:metaInfo. Also, all the related classes are defined as sub-classes of kcs:KCSClass. Here follows the set of pre-defined meta-properties, mostly related to the tracking information The hierarchy of the properties is presented together with the domain and range restrictions for each property:

```
metaInfo
 trackingInfo (domain=rdfs:Statement
     range=)
  bornAt (domain=rdfs:Statement
     range=Update)
  diedAt (domain=rdfs:Statement
     range=Update)
 securityInfo
  lockedBy (domain=rdfs:Statement
     range=User)
```

The bornAt and diedAt properties define the lifetime of the statement via references to the specific updates. In similar manner we express the information associated with each particular update – the user who made it, the actual time and, etc.

Obvious extensions of the above schema are the Dublin Core primitives – there is no problem those to be declared to be sub-properties of metaInfo. The above-proposed model has number of advantages:

– Flexibility. Various types of meta-information could be defined – the appropriate schema has to be created with the only requirement the properties there to be defined as sub-properties of metaInfo and the classes as sub-classes of MetaInfoClass.
– The different meta-properties may have their appropriate domain and range restrictions.
– It is easy to preserve the meta-info, just ignoring the statements which predicates are sub-properties of metaInfo and the resources of class MetaInfoClass.

6.1 Meta-Information for Statements

Assigning meta-information to statements is trickier than to resources at least because there is no URIs to identify each statement. For each statement that we like to attach a meta-information we need to apply a sort of explicit- or pseudo-reification in order to associate the required meta-properties with the appropriate instance of rdf:Statement. A special class is defined in the KCS schema for this purpose:

```
<rdfs:Class
  rdf:about="&kcs;StatementMetaInfo"
 rdfs:label="StatementMetaInfo">
 <rdfs:comment>
   A common super-class for all the
   meta-information about statements
 </rdfs:comment>
 <rdfs:subClassOf
   rdf:resource="&rdf;Statement"/>
 <rdfs:subClassOf
   rdf:resource="&kcs;MetaInfoClass"/>
</rdfs:Class>
```

When we need to associate some meta-information to a statement we can instantiate kcs:StatementMetaInfo for that statement directly referring to its subject, predicate and object. Here is an example of such instantiation:

```
<kcs:StatementMetaInfo
     rdf:about="&mex;status1">
  <rdf:subject rdf:resource="&mex;Jim"/>
  <rdf:predicate
     rdf:resource="&mex;childOf"/>
  <rdf:object rdf:resource="&mex;John"/>
  <rdfs:comment>Comment on statement
      (Jim,childOf,John)</rdfs:comment>
   <mex:customMetaProp>customMetaProp on
   statement (Jim,childOf,John)
   </mex:customMetaProp>
</kcs:StatementMetaInfo>
```

In this case, two pieces of meta-information are attached to the `<Jim,childOf,John>` statement. The first one is just a comment encoded using the standard rdfs:comment property. The second one is a sample for custom meta-property, which was defined by the user.

We can easily extract all the meta-information about specific statement using an RQL query. To do that we need the subject, predicate and object of the statement – it is sad but true, there is no other standard way to refer to or specify a triple. A sample query retrieving the meta-properties and comments about the statement `<Jim, childOf, Jonh>` from the above example may looks like:

```
select
    @metaProp, result
from
    {X : $CX } &rdfs;subject {A},
    {X : $CX } &rdfs;predicate {B},
    {X : $CX } &rdfs;object {C},
    {X : $CX } @metaProp {result}
where
    A=&mex;Jim and B=&mex;childOf
       and C=&mex;John
  and
    ( @metaProp=&rdfs;comment or
      @metaProp < &kcs;metaInfo )
  and   $CX > &rdf;Statement
```

6.2 Low-level Representation of Meta-Information for Statements

Although conceptually clear, the above model for keeping meta-information (including tracking data) for statements has at least the following problems:

– Technically speaking, it requires reification, which is the most criticized RDF(S) feature, also not supported by many tools;
– For each statement of 'real' data, there should be five statements tracking it (one for each of the sub-properties of kcs:trackingInfo and three more that define the appropriate updates), i.e. the volume of the tracking data is five times (!) bigger than the volume of the repository without it.

In order to resolve this issues, the KCS meta-information is actually stored and queried internally using more appropriate encoding -x- the RDF(S) engine takes care to support some level of mimicry to preserve the abstraction of the above presented schema for the applications.

A simple database schema is presented next to provide a general idea for possible implementation of a KCS – the real implementation uses a bit more complex schema which still follows the same ideas. Suppose, an RDF(S) repository works on top of an RDBMS (which is the case with SESAME) and there is a table Statements with columns Subject, Predicate and Object each row of which represents a single statement. Few more columns could be added to this table with references to the tables with security and tracking information. This way the problems with the volume and performance will be resolved at least with respect to the most critical use cases.

The Updates table can keep the information relevant to each update: the time when it happened and the user who performed the update (this information can easily be extended on demand.) In the Statements table, for each statement, the UID when it appeared and disappeared is kept as a reference to the Updates table.

With respect to the tracking of the changes, design as the one proposed above has a number of advantages compared to a variant where the update information is kept directly in the Statements table:

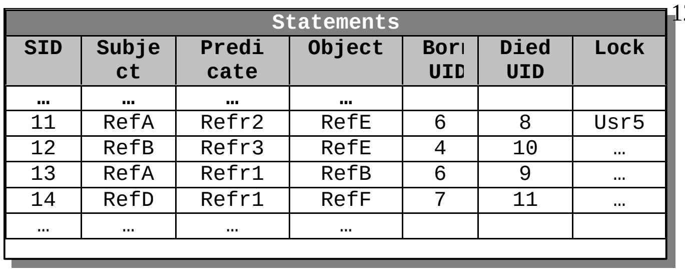
– All the necessary information is kept;
– The tracking information is not messing with the 'real' information about the statements, however when necessary the two tables can be easily joined;
– The most basic operations (that are expected to be performed most frequently) can be done over the Statements table without need for joining with the Updates table. Such operation is to take all the statements that were 'alive' at certain state identified by UID;
– There is significant reduction of the volume of the tracking information in cases of batch updates when multiple statements are getting added or removed at once and thus refer to one and the same UID.

7 Conclusion and future work

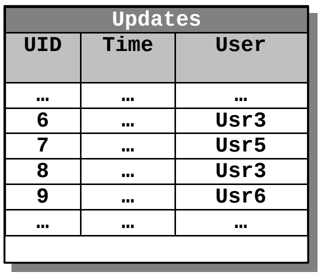
The ontology middleware, part of which is tracking changes module presented still have to prove itself in real-world applications. At this stage it is work in progress inspired by the methodology, tools, and case studies developed under the On-To-Knowledge project.

We see two interesting areas for development. First, the tracking changes mechanism will be used under the OntoView (see [12]) project to serve as a basis for development of the higher-level ontology versioning services. Second, the whole ontology middleware will be applied in language engineering domain for management of linguistic and world knowledge for the purposes of information extraction and ontology extraction application.

To achieve the first goal, we are already working for implementation of the OntoView – ambitious ontology versioning portal. On the linguistic front, the Ontology Middleware module will be integrated with GATE (see [11] and `http://gate.ac.uk`) – one of the most mature platforms for language engineering that already provides substantial support of ontology-based annotations and integration of lexical knowledge bases such as WordNet.



Statements						
SID	Subject	Predicate	Object	Bor UID	Died UID	Lock
…	…	…	…			
11	RefA	Refr2	RefE	6	8	Usr5
12	RefB	Refr3	RefE	4	10	…
13	RefA	Refr1	RefB	6	9	…
14	RefD	Refr1	RefF	7	11	…
…	…	…	…			



Updates		
UID	Time	User
…	…	…
6	…	Usr3
7	…	Usr5
8	…	Usr3
9	…	Usr6
…	…	…

References

1. B. Benatallah, Z.Tari. Dealing with Version Pertinence to Design an Efficient Schema Evolution Framework. In: Proceedings of a ‘International Database Engineering and Application Symposium (IDEAS’98)’, pp.24-33, Cardiff, Wales, U.K. July 8-10, 1998
2. J. Broekstra, A. Kampman. Sesame: A generic Architecture for Storing and Querying RDF and RDF Schema.Deliverable 9, On-To-Knowledge project, October 2001. http://www.ontoknowledge.org/downl/del10.pdf
3. Y. Ding, D. Fensel, M. Klein, B. Omelayenko. Ontology management: survey, requirements and directions. Deliverable 4, On-To-Knowledge project, June 2001. http://www.ontoknowledge.org/downl/del4.pdf
4. E. Franconi, F. Grandi, F. Mandreoli. Schema Evolution and Versioning: a Logical and Computational Characterization. In ‘Database schema evolution and meta-modeling’ - Ninth International Workshop on Foundations of Models and Languages for Data and Objects, Schloss Dagstuhl, Germany, September 18-21, 2000. LNCS No. 2065, pp. 85-99
5. E. Franconi, F. Grandi, F. Mandreoli. A Semantic Approach for Schema Evolution and Versioning of OODB. Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, August 17 - 19, 2000, pp. 99-112
6. A. Kiryakov, K. Simov, M. Dimitrov. OntoMap – the Guide to the Upper-Level.In: Proceedings of the International Semantic Web Working Symposium (SWWS), July 30 - August 1, 2001, Stanford University, California, USA.
7. A. Kiryakov, K. Simov, D. Ognyanov. Ontology Middleware: Analysis and Design. Deliverable 38, On-To-Knowledge project, March 2002.
8. S. Kitcharoensakkul and V. Wuwongse. Towards a Unified Version Model using the Resource Description Framework (RDF). International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 11, No. 6, December 2001.
9. W3C; O. Lassila, R. Swick, eds. Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
10. P. Hayes. RDF Model Theory. W3C Working Draft, 2001 http://www.w3.org/TR/rdf-mt/
11. H. Cunningham. GATE, a General Architecture for Text Engineering. Computing and the Humanities, 2002. (to appear).
12. M. Klein, A. Kiryakov, D. Fensel, D. Ognyanov. Finding and characterizing changes in ontologies. Proceedings of the 21st International Conference on Conceptual Modeling – ER 2002, October 7-11, 2002, Tampere, Finland. (to appear)

Tracing Data Lineage Using Schema Transformation Pathways

Hao Fan and Alexandra Poulovassilis

School of Computer Science and Information Systems, Birkbeck College,
University of London, Malet Street, London WC1E 7HX, {hao,ap}@dcs.bbk.ac.uk

Abstract. With the increasing amount and diversity of information available on the Internet, there has been a huge growth in information systems that need to integrate data from distributed, heterogeneous data sources. Tracing the lineage of the integrated data is one of the current problems being addressed in data warehouse research. In this paper, we propose a new approach for tracing data linage based on schema transformation pathways. We show how the individual transformation steps in a transformation pathway can be used to trace the derivation of the integrated data in a step-wise fashion. Although developed for a graph-based data model and a functional query language, our approach is not limited to these and would be useful in any data transformation/integration context.

1 Introduction

A *data warehouse* consists of a set of materialized views defined over a number of data sources. It collects copies of data from remote, distributed, autonomous and heterogeneous data sources into a central repository to enable analysis and mining of the integrated information. Data warehousing is popularly used for on-line analytical processing (OLAP), decision support systems, on-line information publishing and retrieving, and digital libraries. However, sometimes what we need is not only to analyse the data in the warehouse, but also to investigate how certain warehouse information was derived from the data sources. Given a tuple t in the warehouse, finding the set of source data items from which t was derived is termed the *data lineage* problem [8]. Supporting lineage tracing in data warehousing environments brings several benefits and applications, including in-depth data analysis, on-line analysis mining (OLAM), scientific databases, authorization management, and materialized view schema evolution [2,20,6,8,10,9].

Automed (`http://www.ic.ac.uk/Automed`) is a data transformation and integration system, supporting both virtual and materialized integration of schemas expressed in a variety of modelling languages. This system is being developed in a collaborative EPSRC-funded project between Birkbeck and Imperial Colleges, London

Common to many methods for integrating heterogeneous data sources is the requirement for logical integration [11] of the data, due to variations in the design of data models for the same universe of discourse. A common approach is to define a single integrated schema expressed using a *common data model* (CDM). In previous work within the Automed project [18,12], a general framework has been developed to support schema transformation and integration. This framework consists of a low-level **hypergraph based data model (HDM)** and a set of primitive schema transformations defined in terms of this lower-level model.

[13] gives the definitions of equivalent HDM representations for ER, relational and UML schemas, and discusses how inter-model transformations can be supported via this underlying common data model. The approach is extended to also encompass XML data sources in [14], and current work in Automed is also extending its scope to formatted data files and plain text files.

Using a higher-level CDM such as an ER model or the relational model can be complicated because the original and transformed schemas may be represented in different high-level modelling languages and there may not be a simple semantic correspondence between their modelling constructs. In contrast, HDM schemas contain *Nodes*, *Edges* and *Constraints* as their constructs, which can be used as the underlying representation for higher-level modelling constructs. Thus, inter-model transformations can be performed by transforming the HDM representations of higher-level modelling constructs. We term the sequence of primitive transformations defined for transforming a schema S_1 to a schema S_2 a *transformation pathway* from S_1 to S_2. That is, a transformation pathway consists of a sequence of primitive schema transformations.

[12] discusses how Automed transformation pathways are automatically reversible, thus allowing automatic translation of data and queries between schemas. In this paper we show how Automed's transformation pathways can also be used to trace the lineage of data in a data warehouse which integrates data from several sources. We assume that both the source schemas and the integrated schema are expressed in the HDM data model since, as discussed in [13], higher-level schemas and transformations between them can be automatically translated into an equivalent HDM representation. We use a functional *intermediate query language* (IQL) for expressing the semantic relationships between schema constructs in transformations.

The remainder of this paper is as follows. Section 2 discusses related work and existing methods of tracing data lineage. Section 3 reviews the Automed framework, including the HDM data model, IQL syntax, and transformation pathways. Section 4 gives our definitions of data lineage and describes the methods of tracing data lineage we have adopted in Automed. Section 5 gives our conclusions and directions of future work.

2 Related work

[20] proposes a general framework for computing *fine-grained* data lineage using a limited amount of information about the processing steps. The notion of *weak inversion* is introduced. Based on a weak inverse function, which must be specified by the transformation definer, the paper defines and traces data lineage for each transformation step in a visualization database environment.

[8] provides some fundamental definitions relating to the data lineage problem, including tuple derivation for an operator, and tuple derivation for a view. It has addressed the derivation tracing problem using bag semantics and has provided the concept of *derivation set* and *derivation pool* for tracing data lineage with duplicate elements. We use these ideas in our approach and define the notions of *affect-pool* and *origin-pool* in Automed.

Another fundamental concept is addressed in [4,5], namely the difference between "why" provenance and "where" provenance. Why-provenance refers to the source data that had some influence on the existence of the integrated data. Where-provenance refers to the actual data in the sources from which the integrated data was extracted. The problem of why-provenance has been studied for relational databases in [8,20,6,7]. Here, we introduce the notions of *affect* and *origin* provenance, give definitions for data lineage in Automed, and discuss the lineage tracing algorithms for these the two kinds of provenance.

There are also other previous works related to data lineage tracing [2,9,10]. Most of these consider *coarse-grained* lineage based on annotations on each data transformation step, which provides estimated lineage information not the exact tuples in the data sources. Using our approach, *fine-grained* lineage, i.e. a specific derivation in the data sources, can be computed given the source schemas, integrated schema, and transformation pathways between them. All of our algorithms are based on bag semantics using the HDM data model and the IQL query language.

3 The Automed Framework

This section gives a short review of the Automed schema transformation framework, including the HDM data model, IQL language, and transformation pathways. More details of this material can be found in [18,12,13,17].

A **schema** in the **Hypergraph Data Model (HDM)** is a triple (*Nodes, Edges, Constraints*) containing a set of nodes, a set of edges, and a set of constraints. A **query** q over a schema S is an expression whose variables are members of *Nodes* and *Edges*. *Nodes* and *Edges* define a labelled, directed, nested hypergraph. It is nested in the sense that edges can link any number of both nodes and other edges. *Constraints* is a set of boolean-valued queries over S. The nodes and edges of a schema are identified by their *scheme*. For a node this is just its name and for an edge it is of the form $\langle\langle edgeName, scheme_1, scheme_2, \ldots, scheme_n\rangle\rangle$, where $scheme_1, \ldots, scheme_n$ are the schemes of the constructs connected by the edge. Edge names are optional and the absence of a name is denoted by "_".

An **instance** I of a schema S = (*Nodes, Edges, Constraints*) is a set of sets satisfying the following:

(i) each construct $c \in Nodes \cup Edges$ has an extent, denoted by $Ext_{S,I}(c)$, that can be derived from I ;
(ii) conversely, each set in I can be derived from the set of extents $\{Ext_{S,I}(c) | c \in Nodes \cup Edges\}$
(iii) for each $e \in Edges$, $Ext_{S,I}(e)$ contains only values that appear within the extents of the constructs linked by e;
(iv) the value of every constraint $c \in Constraints$ is true, the **value** of a query q being given by $q[c_1/Ext_{S,I}(c_1), \ldots, c_n/Ext_{S,I}(c_n)]$ where $c_1, \ldots, c_n$ are the constructs in $Nodes \cup Edges$.

The function $Ext_{S,I}$ is called an **extension mapping**. A HDM **model** is a triple $(S, I, Ext_{S,I})$. The primitive transformations on HDM models are as follows, each transformation being a function that when applied to a model returns a new model. Note that only the schema and extension mapping are affected by these transformations, not the instance i.e. not the data:

- $renameNode(fromName, toName)$ renames a node.
- $renameEdge(\langle\langle fromName, c_1, \ldots, c_n\rangle\rangle, toName)$ renames an edge.
- $addConstraint\ c$ adds a new constraint c.
- $delConstraint\ c$ deletes a constraint.
- $addNode(name, q)$ adds a node named name whose extent is given by the value of the query q over the existing schema constructs.
- $delNode(name, q)$ deletes a node. Here, q is a query that states how the extent of the deleted node could be recovered from the extents of the remaining schema constructs (thus, not violating property (ii) of an instance).
- $addEdge(\langle\langle name, c_1, \ldots, c_n\rangle\rangle, q)$ adds a new edge between a sequence of existing schema constructs $c_1, \ldots, c_n$. The extent of the edge is given by the value of the query q over the existing schema constructs.
- $delEdge(\langle\langle name, c_1, \ldots, c_n\rangle\rangle, q)$ deletes an edge. q states how the extent of the deleted edge could be recovered from the extents of the remaining schema constructs.

A **composite transformation** is a sequence of $n \geq 1$ primitive transformations. We term the composite transformation defined for transforming schema S_1 to schema S_2 a *transformation pathway* from S_1 to S_2.

The query, q, in each transformation is expressed in a functional *intermediate query language*, IQL [18]. This supports a number of primitive types such as booleans, strings and numbers, as well as product, function and bag types. The set of *simple* IQL queries are as follows, where D, $D_1 \ldots$, D_r denote a bag of the appropriate type, $++$ is bag union, $--$ is bag *monus* [1], $group$ groups a bag of pairs on their first component, $sortDistinct$ sorts a bag and removes duplicates, $aggFun$ is an aggregation function (max, min, $count$, sum, avg), and gc groups a bag of pairs on their first component and applies an aggregation function to the second component:

$$
\begin{aligned}
&q = D_1 ++ D_2 ++ \ldots ++ D_r\\
&q = D_1 -- D_2\\
&q = group\ D\\
&q = sort\ D\\
&q = sortDistinct\ D\\
&q = aggFun\ D\\
&q = gc\ aggFun\ D\\
&q = [p | p \leftarrow D_1;\ member\ D_2\ p]\\
&q = [p | p \leftarrow D_1;\ not\ (member\ D_2\ p)]\\
&q = [p | p_1 \leftarrow D_1; \ldots; p_r \leftarrow D_r; c_1; \ldots; c_k]
\end{aligned}
$$

General IQL queries are formed by arbitrary nesting of the above simple query constructs.

The last three constructs above are *comprehensions* [19]. These have the general syntax $[e | Q_1; \ldots; Q_n]$, where Q_1 to

Q_n are qualifiers, each qualifier being either a filter or a generator. A filter is a boolean-valued expression (the c_i above are filters). A generator has syntax $p \leftarrow q$ where p is a pattern and q is a collection-valued expression. A pattern is either a variable or a tuple of patterns. In IQL, the head expression e of a comprehension is also constrained to be a pattern.

IQL can represent common database query operations, such as select-project-join (SPJ) operations and SPJ operations with aggregation (ASPJ). For example, to get the maximum daily sales total for each store in the relation `StoreSales(store_id,daily_total,date)`, in SQL we use:

```
SELECT   store_id, max(daily_total)
FROM     StoreSales
GROUP BY store_id
```

In IQL this query is expressed by

$$gc\ max\ [(s,t) \mid (s,t,d) \leftarrow StoreSales]$$

Example: Transforming between HDM schemas

Consider two HDM schemas $S_1 = (N_1, E_1, C_1)$ and $S_2 = (N_2, E_2, C_2)$, where

$N_1 = \{mathematician, compScientist, salary\}$,
$C_1 = \{\}$,
$E_1 = \{\langle\langle \_, mathematician, salary\rangle\rangle,$
$\langle\langle \_, compScientist, salary\rangle\rangle\}$,
$N_2 = \{dept, person, salary, avgDeptSalary\}$,
$C_2 = \{\}$,
$E_2 = \{\langle\langle \_, dept, person\rangle\rangle, \langle\langle \_, person, salary\rangle\rangle,$
$\langle\langle \_, dept, avgDeptSalary\rangle\rangle\}$.

Figure 1 illustrates these two schemas, with S_1 on the left and S_2 on the right of the figure:

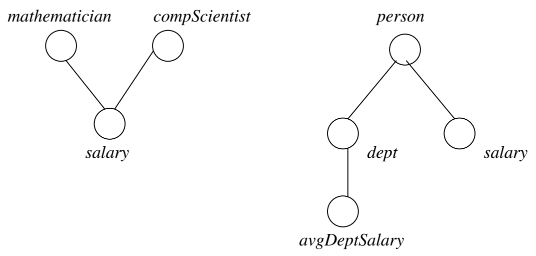


Fig. 1. Transforming Schema S_1 to Schema S_2

S_1 can be transformed to S_2 by the sequence of primitive schema transformations given below. The first 6 transformation steps create the constructs of S_2 which do not exist in S_1. The query in each step gives the extension of the new schema construct in terms of the existing schema constructs. The last 4 steps then delete the redundant constructs of S_1. The query in each of these steps shows how the extension of each deleted construct can be reconstructed from the remaining schema constructs:

addNode $(dept, \{\text{"Maths"}, \text{"CompSci"}\})$;
addNode $(person, [x|x \leftarrow mathematician] ++$
$[x|x \leftarrow compScientist])$;
addNode $(avgDeptSalary,$
$\{avg\ [s\ |(m,s) \leftarrow \langle\langle \_, mathematician, salary\rangle\rangle]\}$
$++$
$\{avg\ [s|(c,s) \leftarrow \langle\langle \_, compScientist, salary\rangle\rangle]\})$;
addEdge $(\langle\langle \_, dept, person\rangle\rangle,$
$[(\text{"Maths"}, x)|x \leftarrow mathematician] ++$
$[(\text{"CompSci"}, x)|x \leftarrow compScientist])$;
addEdge $(\langle\langle \_, person, salary\rangle\rangle, \langle\langle \_, mathematician, salary\rangle\rangle$
$++ \langle\langle \_, compScientist, salary\rangle\rangle)$;
addEdge $(\langle\langle \_, dept, avgDeptSalary\rangle\rangle,$
$\{(\text{"Maths"},$
$avg\ [s|(m,s) \leftarrow \langle\langle \_, mathematician, salary\rangle\rangle]),$
$(\text{"CompSci"},$
$avg\ [s|(c,s) \leftarrow \langle\langle \_, compScientist, salary\rangle\rangle])\})$;
delEdge $(\langle\langle \_, mathematician, salary\rangle\rangle,$
$[(p,s)|(d,p) \leftarrow \langle\langle \_, dept, person\rangle\rangle;$
$(p', s) \leftarrow \langle\langle \_, person, salary\rangle\rangle;$
$d = \text{"Maths"}; p = p'])$;
delEdge $(\langle\langle \_, compScientist, salary\rangle\rangle,$
$[(p,s)|(d,p) \leftarrow \langle\langle \_, dept, person\rangle\rangle;$
$(p', s) \leftarrow \langle\langle \_, person, salary\rangle\rangle;$
$d = \text{"CompSci"}; p = p'])$;
delNode $(mathematician,$
$[p|(d,p) \leftarrow \langle\langle \_, dept, person\rangle\rangle; d = \text{"Maths"}])$;
delNode $(compScientist,$
$[p|(d,p) \leftarrow \langle\langle \_, dept, person\rangle\rangle; d = \text{"CompSci"}])$;

4 Tracing data lineage in Automed

The fundamental definitions regarding data lineage are given in [8], including tuple derivation for an operator, tuple derivation for a view, and methods of derivation tracing with both *set* and *bag* semantics. However, these definitions and methods are limited to *why-provenance*[5] and what they consider is a class of views defined over base relations using the relational algebra operators: *selection* (σ), *projection*(π), *join* ($\bowtie$), *aggregation* (α), *set union* ($\bigcup$), and *set difference* ($-$). The query language used in Automed is IQL based on *bag* semantics allowing duplicate elements in a data source schema or the integrated data, and also within the collections that are derived during lineage tracing. Also, we consider both *affect-provenance* and *origin-provenance* in our treatment of the data lineage problem.

What we regard as affect-provenance includes all of the source data that had some influence on the result data. Origin-provenance is simpler because here we are only interested in the specific data in the sources from which the resulting data is extracted.

4.1 Data lineage with set semantics in IQL

The definition of *tuple derivation for an operation* was given in [8] considering only the aspect of affect-provenance. We use the notions of *maximal witness*and *minimal witness* from [5] to classify data lineage into two aspects: *affect-set* and *origin-set*. For set semantics and simple IQL queries, the definitions of affect-set and origin-set for a tuple in the integrated database are as follows. The q in these definitions is any IQL simple query.

Definition 1 (Affect-set for a simple query in IQL). Let q be any simple query over sets $T_1, \ldots, T_m$, and let $V = q(T_1, \ldots, T_m)$ be the set that results from applying q to $T_1, \ldots, T_m$. Given a tuple $t \in V$, we define t's *affect-set in* $T_1, \ldots, T_m$ *according to* q to be the sequence of sets $q^A_{\langle T1,\ldots,Tm\rangle}(t) = \langle T^*_1, \ldots, T^*_m\rangle$, where $T^*_1, \ldots, T^*_m$ are **maximal** subsets of $T_1, \ldots, T_m$ such that:

(a) $q(T^*_1, \ldots, T^*_m) = \{t\}$
(b) $\forall T_i$': $q(T^*_1, \ldots, T_i', \ldots, T^*_m) = \{t\} \Rightarrow T_i' \subseteq T^*_i$
(c) $\forall T^*_i$: $\forall t^* \in T^*_i$: $q(T^*_1, \ldots, \{t^*\}, \ldots, T^*_m) \neq \emptyset$

Also, we say that $q^A_{Ti}(t) = T^*_i$ is *t's affect-set in* T_i.

Definition 2 (Origin-set for a simple query in IQL). Let $q, T_1, \ldots, T_m, V$ and t be as above. We define t's *origin-set in* $T_1, \ldots, T_m$ *according to* q to be the sequence of sets $q^O_{\langle T1,\ldots,Tm\rangle}(t) = \langle T^*_1, \ldots, T^*_m\rangle$, where $T^*_1, \ldots, T^*_m$ are **minimal** subsets of $T_1, \ldots, T_m$ such that:

(a) $q(T^*_1, \ldots, T^*_m) = \{t\}$
(b) $\forall T_i$':$T_i' \subset T^*_i$: $q(T^*_1, \ldots, T_i', \ldots, T^*_m) \neq \{t\}$
(c) $\forall T^*_i$: $\forall t^* \in T^*_i$: $q(T^*_1, \ldots, \{t^*\}, \ldots, T^*_m) \neq \emptyset$

Also, we say that $q^O_{Ti}(t) = T^*_i$ is *t's origin-set in* T_i.

In the above definitions, condition (a) states that the result of applying query q to the lineage must be the tracing tuple t; condition (b) is used to enforce the maximizing and minimizing properties respectively; and condition (c) removes the redundant elements in the computed derivation of tuple t (see [8]).

Proposition 1. The origin-set of a tuple t is a subset of the affect-set of t.

4.2 Data lineage with bag semantics in IQL

As mentioned above, our approach for tracing data lineage is based on bag semantics which allow duplicate elements to exist in the source schemas, the integrated schema and the computed lineage collections. We use the notions of *affect-pool* and *origin-pool* to describe the data lineage problem with bag semantics:

Definition 3 (Affect-pool for a simple query in IQL). Let q be any simple query over bags $T_1, \ldots, T_m$, and let $V = q(T_1, \ldots, T_m)$ be the bag that results from applying q to $T_1, \ldots, T_m$. Given a tuple $t \in V$, we define t's *affect-pool in* $T_1, \ldots, T_m$ *according to* q to be the sequence of bags $q^{AP}_{\langle T1,\ldots,Tm\rangle}(t) = \langle T^*_1, \ldots, T^*_m\rangle$, where $T^*_1, \ldots, T^*_m$ are **maximal** sub-bags of $T_1, \ldots, T_m$ such that:

(a) $q(T^*_1, \ldots, T^*_m) = \{x|x \leftarrow T; x = t\}$
(b) $\forall T_i$': $q(T^*_1, \ldots, T_i', \ldots, T^*_m) = \{x|x \leftarrow T; x = t\} \Rightarrow T'_i \subseteq T^*_i$
(c) $\forall T^*_i$: $\forall t^* \in T^*_i$: $q(T^*_1, \ldots, \{t^*\}, \ldots, T^*_m) \neq \emptyset$

Also, we say that $q^{AP}_{Ti}(t) = T^*_i$ is *t's affect-pool in* T_i.

Definition 4 (Origin-pool for a simple query in IQL). Let $q, T_1, \ldots, T_m, V$ and q be as above. We define t's *origin-pool in* $T_1, \ldots, T_m$ *according to* q to be the sequence of bags $q^{OP}_{\langle T1,\ldots,Tm\rangle}(t) = \langle T^*_1, \ldots, T^*_m\rangle$, where $T^*_1, \ldots, T^*_m$ are **minimal** sub-bags of $T_1, \ldots, T_m$ such that:

(a) $q(T^*_1, \ldots, T^*_m) = \{x|x \leftarrow T; x = t\}$
(b) $\forall T^*_i$: $\neg\exists t^*$: $t^* \in T^*_i$, $t^* \in (T_i - T^*_i)$
(c) $\forall T^*_i$: $\forall t^* \in T^*_i$: $q(T^*_1, \ldots, \{x|x \leftarrow T^*_i; x \neq t^*\}, \ldots, T^*_m) \neq \{x|x \leftarrow T; x = t\}$
(d) $\forall T^*_i$: $\forall t^* \in T^*_i$: $q(T^*_1, \ldots, \{t^*\}, \ldots, T^*_m) \neq \emptyset$

Also, we say that $q^{OP}_{Ti}(t) = T^*_i$ is *t's origin-pool in* T_i.

Note that the condition (b) in Definition 4 ensures that if the origin-pool of a tuple t is T^*_i in the source bag T_i, then for any tuple in T_i, either all of the copies of the tuple are in T^*_i or none of them are in T^*_i.

Proposition 2. The origin-pool of a tuple t is a sub-bag of the affect-pool of t.

¿From above definitions and the definition of simple IQL queries in Section 3, we now specify the affect-pool and origin-pool for IQL simple queries. As in [8], we use *derivation tracing queries* to evaluate the lineage of a tuple t with respect to a sequence of bags D. That is, we apply a query to t and the result is the derivation of t in D. We call such a query the *tracing query for t on D*, denoted as $TQ_D(t)$.

Theorem 1 (*Affect-pool* and *Origin-pool* for a tuple with IQL simple queries.) Let $V = q(D)$ be the bag that results from applying a simple IQL query q to a sequence of bags D. Then, for any tuple $t \in V$, the tracing queries$TQ^{AP}_D(t)$ below give the affect-pool of t in D, and the tracing queries $TQ^{OP}_D(t)$ give the origin-pool of t in D:

$$\begin{aligned} q &= D_1 ++ \ldots ++ D_r \quad (D = \langle D_1, \ldots, D_r\rangle) \\ TQ^{AP}_D(t) &= TQ^{OP}_D(t) = \\ &\langle [x|x \leftarrow D_1; x = t], \ldots, [x|x \leftarrow D_r; x = t]\rangle \end{aligned}$$

$$\begin{aligned} q &= D_1 -- D_2 \quad (D = \langle D_1, D_2\rangle) \\ TQ^{AP}_D(t) &= \langle [x|x \leftarrow D_1; x = t], D_2\rangle \\ TQ^{OP}_D(t) &= \langle [x|x \leftarrow D_1; x = t], [x|x \leftarrow D_2; x = t]\rangle \end{aligned}$$

$$\begin{aligned} q &= group\ D \\ TQ^{AP}_D(t) &= TQ^{OP}_D(t) = \\ &[x|x \leftarrow D; first\ x = first\ t] \end{aligned}$$

$$\begin{aligned} q &= sort\ D\ /\ sortDistinct\ D \\ TQ^{AP}_D(t) &= TQ^{OP}_D(t) = \\ &[x|x \leftarrow D; x = t] \end{aligned}$$

$$\begin{aligned} q &= max\ D\ /\ min\ D \\ TQ^{AP}_D(t) &= D \\ TQ^{OP}_D(t) &= [x|x \leftarrow D; x = t] \end{aligned}$$

$$\begin{aligned} q &= count\ D\ /\ sum\ D\ /\ avg\ D \\ TQ^{AP}_D(t) &= TQ^{OP}_D(t) = D \end{aligned}$$

$$\begin{aligned} q &= gc\ max\ D\ /\ gc\ min\ D \\ TQ_D^{AP}(t) &= [x|x \leftarrow D; first\ x = first\ t] \\ TQ_D^{OP}(t) &= [x|x \leftarrow D; x = t] \end{aligned}$$

$$\begin{aligned} q &= gc\ count\ D\ /\ gc\ sum\ D\ /\ gc\ avg\ D \\ TQ_D^{AP}(t) &= TQ_D^{PP}(t)\ = \\ & [x|x \leftarrow D; first\ x = first\ t] \end{aligned}$$

$$\begin{aligned} q &= [x|x \leftarrow D_1; member\ D_2\ x] \\ & (D = \langle D_1, D_2 \rangle) \\ TQ_D^{AP}(t) &= TQ_D^{OP}(t)\ = \\ & \langle [x|x \leftarrow D_1; x = t], [x|x \leftarrow D_2; x = t] \rangle \end{aligned}$$

$$\begin{aligned} q &= [x|x \leftarrow D_1; not\ (member\ D_2\ x)] \\ & (D = \langle D_1, D_2 \rangle) \\ TQ_D^{AP}(t) &= \langle [x|x \leftarrow D_1; x = t], D_2 \rangle \\ TQ_D^{OP}(t) &= [x|x \leftarrow D_1; x = t] \end{aligned}$$

$$\begin{aligned} q &= [p|p_1 \leftarrow D_1; \ldots; p_r \leftarrow D_r; c_1; \ldots; c_k] \\ & (D = \langle D_1, \ldots, D_r \rangle) \\ TQ_D^{AP}(t) &= TQ_D^{OP}(t)\ = \\ & \langle [p_1|p_1 \leftarrow D_1; p_1 = t_1; \ldots; \\ & \quad p_r \leftarrow D_r; p_r = t_r; c_1; \ldots; c_k], \ldots, \\ & \ [p_r|p_1 \leftarrow D_1; p_1 = t_1; \ldots; \\ & \quad p_r \leftarrow D_r; p_r = t_r; c_1; \ldots; c_k] \rangle \end{aligned}$$

In the last query form above, each pattern p_i is a subpattern of p and all tuples $t \in V$ match p; for any $t \in V$, t_i is the tuple derived by projecting the components of $p_i$from t.

It is simple to show that the results of queries $TQ_D^{AP}(t)$ and $TQ_D^{OP}(t)$ satisfy Definition 3 and 4 respectively. For more complex IQL queries, the above formulae can be successively applied to the syntactic structure of an IQL query. An alternative approach would be to decompose a transformation step containing a complex IQL query into a sequence of transformation steps each containing a simple IQL query.

4.3 Tracing data lineage through transformation pathways

For simplicity of exposition, henceforth we assume that all of the source schemas have first been integrated into a single schema S consisting of the union of the constructs of the individual source schemas, with appropriate renaming of schema constructs to avoid duplicate names.

Suppose an integrated schema *GS* has been derived from this source schema S though a transformation pathway $TP = tp_1, \ldots, tp_r$. Treating each transformation step as a function applied to *S*, *GS* can be obtained as *GS* = $tp_1 \circ tp_2 \circ \ldots \circ tp_r(S) = tp_r(\ldots(tp_2(tp_1(S)))\ldots)$. Thus, tracing the lineage of data in *GS* requires tracing data lineage via a query-sequence, defined as follows:

Definition 5 (Affect-pool for a query-sequence)
Let $Q = q_1, q_2, \ldots, q_r$ be a query-sequence over a sequence of bags D, and let $V = Q(D) = q_1 \circ q_2 \circ \ldots \circ q_r(D)$ be the bag that results from applying Q to D. Given a tuple $t \in V$, we define t's *affect-pool in D according to Q* to be $Q_D^{AP}(t) = D^*$, where $D_i^* = q_i^{AP}(D_{i+1}^*)$ $(1 \le i \le r)$, $D_{i+1}^* = \{t\}$ and $D^* = D_1^*$.

Definition 6 (Origin-pool for query-sequence).
Let Q, D, V and t be as above. We define t's *origin-pool in D according to Q* to be $Q_D^{OP}(t) = D^*$, where $D_i^* = q_i^{OP}(D_{i+1}^*)$ $(1 \le i \le r)$, $D_{i+1}^* = \{t\}$ and $D^* = D_1^*$.

Definitions 5 and 6 show that the derivations of data in an integrated schema can be derived by examining the transformation pathways in reverse, step by step.

An Automed transformation pathway is a composite transformation consisting of a sequence of primitive transformations which generate the integrated schema from the given source schemas. The constructs of an HDM schema are *Nodes*, *Edges*, and *Constraints*. When considering data lineage tracing, we treat *Nodes* and *Edges* similarly since both of these kinds of constructs have an extent, i.e. contain data. We ignore the *Constraints* part of a schema because a constraint is just a query over the nodes and edges of a schema and does not contain any data.

Thus, for data lineage tracing, we consider the primitive transformations *addNode* and *addEdge* as a single *addConstruct* transformation, *delNode* and *delEdge* as *delConstruct*, *renameNode* and *renameEdge* as *renameConstruct*, and we ignore *addConstraint* and *delConstraint* transformations.

We thus summarize the problem of data lineage for each kind transformation step as follows:

(a) For an *addConstruct*(O, q) transformation, the lineage of data in the extent of schema construct O is located in the extents of the schema constructs appearing in q.
(b) For a *renameConstruct*(O', O) transformation, the lineage of data in the extent of schema construct O is located in the extent of schema construct O'.
(c) All *delConstruct*(O, q) transformations can be ignored since they create no schema constructs.

4.4 Algorithms for tracing data lineage

In our algorithms below, we assume that each schema construct, O, has two attributes: *relateTP* is the transformation step that created O, and *extent* is the current extent of O. If a schema construct remains in the global schema directly from one of the source schemas, its *relateTP* value is Ø.

In our algorithms, each transformation step *tp* has four attributes:

– *transfType*, which is *"add"*, *"ren"* or *"del"*;
– *query*, which is the query used in this transformation step (if any);
– *source*, which for a *renameConstruct*(O', O) returns just O', and for an *addConstruct*(O, q) returns a sequence of all the schema constructs appearing in q; and
– *result* which is O for both *renameConstruct*(O', O) and *addConstruct*(O, q).

It is simple to trace data lineage in case (b) discussed above. If *B* is a tuple bag (i.e. bag of tuples) contained in the extent of O, *B*'s data lineage in O' is just *B* itself, and we define this to be both the affect-pool and the origin-pool of *B* in O'.

In case (a), where the construct O was created by a transformation step *addConstruct*(O, q), the key point is how to trace the lineage using the query q. We can use the formulae given in Theorem 1 to obtain the lineage of data created in this case. The procedures *affectPoolOfTuple*(t, O) and *originPoolOfTuple*(t, O) below can be applied to trace the affect pool and origin pool of a tuple t in the extent of schema construct O. The result of these procedures, DL, is a sequence of pairs

$$\langle (D_1, O_1), \ldots, (D_n, O_n) \rangle$$

in which each D_i is a bag which contains t's derivation within the extent of construct O_i. Note that in these procedures, the sequence returned by the tracing queries TQ^{AP} and TQ^{OP} may consist of bags from different schema constructs. For any such bag, B, $B.construct$ denotes the schema construct from whose extent B originates.

```
proc     $affectPoolOfTuple(t, O)$
input :  a tracing tuple $t$ in the extent of construct O
output : $t$'s affect-pool, $DL$
begin
         $D = [(O'.extent, O') \mid O' \leftarrow O.relateTP.source]$
         $D^* = TQ_D^{AP}(t)$;
         $DL = [(B, B.construct) \mid B \leftarrow D^*]$
         $return(DL)$;
end
proc     $originPoolOfTuple(t, O)$
input :  a tracing tuple $t$ in the extent of construct O
output : $t$'s origin-pool, $DL$
begin
         $D = [(O'.extent, O') \mid O' \leftarrow O.relateTP.source]$
         $D^* = TQ_D^{OP}(t)$;
         $DL = [(B, B.construct) \mid B \leftarrow D^*]$
         $return(DL)$;
end
```

Two procedures *affectPoolOfSet*(T, O) and *originPoolOfSet*(T, O) can then be used to compute the derivations of a tuple set (i.e. set of tuples), T. (Because duplicate tuples have an identical derivation, we eliminate any duplicate items and convert the tracing bag to a tracing set first.) We give *affectPoolOfSet* below. *originPoolOfSet*(T, O) is identical, with *originPoolOfTuple* replacing *affectPoolOfTuple*. In these two procedures, we trace the data lineage of each tuple $t \in T$ in turn and incrementally merge each time the result into DL:

```
proc     $affectPoolOfSet(T, O)$
input :  a tracing tuple set $T$ contained in construct $O$
output : $T$'s affect-pool, $DL$
begin
         $DL = \langle\rangle$;  $/*\ the\ empty\ sequence\ */$
         $for\ each\ t \in T\ do$
           $DL = merge(DL, affectPoolOfTuple(t, O))$;
         $return(DL)$;
end
```

Because a tuple t^* can be the lineage of both t_i and t_j $(i \neq j)$, if t^* and all of its copies in a data source have already been added to DL as the lineage of t_i, we do not add them again into DL as the lineage of t_j. This is accomplished by the procedure $merge$ given below, where the operator $-$ removes a element from a sequence and the operator $+$ appends an element to a sequence:

```
proc     $merge(DL, DL^{new})$
input :  data lineage sequence $DL =$
         $\langle (D_1, O_1), \ldots, (D_n, O_n) \rangle$;
         new data lineage sequence$DL^{new}$
output : merged data lineage sequence $DL$
begin
         $for\ each\ (D^{new}, O^{new}) \in DL^{new}\ do$
           $if\ O^{new} = O_i\ for\ some\ O_i\ in\ DL\ then$ {
             $oldData = D_i$;
             $newData = oldData ++$
                $[x \mid x \leftarrow D^{new};\ not\ (member\ oldData\ x)]$;
             $DL = (DL - (oldData, O_i)) +$
                $(newData, O_i)$;
           }
           $else$
             $DL = DL + (D^{new}, O^{new})$;
         $return(DL)$;
end
```

Finally, we give below our algorithm *traceAffectPool*(*B*, *O*) for tracing affect lineage using entire transformation pathways given a integrated schema *GS*, the source schema *S*, and a transformation pathway tp_1, ... , tp_r from S to *GS*. Here, *B* is a tuple bag contained in the extent of schema construct $O \in GS$. We recall that each schema construct has attributes *relateTP* and *extent*, and that each transformation step has attributes *transfType*, *query*, *source* and *result*.

We examine each transformation step from tp_r down to tp_1. If it is a delete step, we ignore it. Otherwise we determine if the $result$ of this step is contained in the current DL. If so, we then trace the data lineage of the current data of O in DL, merge the result into DL, and delete O from DL. At the end of this processing the resulting DL is the lineage of T in the data sources:

```
proc     $traceAffectPool(B, O)$
input :  tracing tuple bag $B$ contained in construct $O$
         transformation pathway $tp_1, \ldots, tp_r$
output : $B$'s affect-pool, $DL$
begin
         $DL = \langle (B, O) \rangle$;
         $for\ j = r\ downto\ 1\ do$
           $case\ tp_j.transfType =$ "del"
             $continue$;
           $case\ tp_j.transfType =$ "ren"
             $if\ tp_j.result = O_i\ for\ some\ O_i\ in\ DL\ then$
               $DL = (DL - (D_i, O_i)) + (D_i, tp_j.source)$;
           $case\ t_j.transfType =$ "add"
             $if\ tp_j.result = O_i\ for\ some\ O_i\ in\ DL\ then$ {
               $DL = DL - (D_i, O_i)$;
               $D_i = sortDistinct\ D_i$;
               $DL = merge(DL, affectPoolOfSet(D_i, O_i))$;
             }
           $end$
         $return(DL)$;
end
```

Procedure $traceOriginPool$ is identical, obtained by replacing *affectPoolOfSet* by *originPoolOfSet*.

5 Conclusions and future work

We have presented definitions for data lineage in Automed based on both why-provenance and where-provenance, which we have termed *affect-pool* and *origin-pool*, respectively. Rather than relying on a high-level common data model such as an ER or relational model, the Automed integration approach is based on a lower-level CDM – the HDM data model. Heterogeneous source schemas can be automatically translated into the equivalent HDM representation, and transformations between them expressed as transformations on their HDM representations. The contribution of the work described in this paper is that we have shown how the individual steps of Automed schema transformation pathways can be used to trace the affect-pool and origin-pool of items of integrated data in a step-wise fashion.

Fundamental to our lineage tracing method is the fact that add and del schema transformations carry a *query* which defines the new or deleted schema construct in terms of the other schema constructs. Thus, our general approach is not limited to the HDM data model and IQL query language and can be applied to schema transformations defined on other data models using different query languages, or indeed to inter-model schema transformation pathways (as used in [14,15] for example).

The data lineage problem and the solutions presented in this paper have led to a number of areas of further work:

– *Combining our approach for tracing data lineage with the problem of incremental view maintenance.* We have already done some preliminary work on using the Automed transformation pathways for incremental view maintenance. We now plan to explore the relationship between our lineage tracing and view maintenance algorithms, to determine if an integrated approach can be adopted for both.
– *Implementing our lineage tracing and view maintenance algorithms*. As a part of the Automed project, we are implementing these algorithms over the Automed repository and API [3].
– *Extending the lineage tracing and view maintenance algorithms to a more expressive transformation language*. [16] extends the Automed transformation language with parametrised procedures and iteration and conditional constructs, and we plan to extend our algorithms to this more expressive transformation language.

References

1. J. Albert. Algebraic properties of bag data types. In *Proc. VLDB'91*, pages 211–219, 1991.
2. P. Bernstein and T. Bergstraesser. Meta-data support for data transformations using microsoft repository. *IEEE Data Engineering Bulletin*, 22(1):9–14, 1999.
3. M. Boyd and N. Tong. The Automed repositories and API. Technical report, Automed Project, 2001.
4. P. Buneman, S. Khanna, and W.C. Tan. Data provenance: some basic issues. In *Proc. FSTTCS 2000*, pages 87–93, 2000.
5. P. Buneman, S. Khanna, and W.C. Tan. Why and Where: a characterization of data provenance. In *Proc. ICDT 2001*, pages 316–33, 2001.
6. Y. Cui. Lineage tracing in data warehouses. phd thesis. Technical report, Computer Science Department, Stanford University, 2001.
7. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Proc. VLDB'01*, pages 471–480, 2001.
8. Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2):179–227, 2000.
9. C. Faloutsos, H.V. Jagadish, and N.D. Sidiropoulos. Recovering information from summary data. In *Proc. VLDB'97*, pages 36–45, 1997.
10. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.A. Saita. Improving data cleaning quality using a data lineage facility. In *Proc. DMDW'01*, page 3, 2001.
11. R. Hull. Managing semantic heterogeneity in databases. In *Proc. PODS'97*, pages 51–61, 1997.
12. P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications - a schema transformation approach. In *Proc. ER'99*, pages 96–133, 1999.
13. P. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, pages 333–348, 1999.
14. P. McBrien and A. Poulovassilis. A semantic approch to integrating XML and structured data sources. In *Proc. CAiSE'01*, pages 330–345, 2001.
15. P. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proc. CAiSE'02*, 2002.
16. A. Poulovassilis. An enhanced transformation language for the HDM. Technical report, Automed Project, 2001.
17. A. Poulovassilis. The Automed Intermediate Query Language. Technical report, Automed Project, 2001.
18. A. Poulovassilis and P. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
19. P. Trinder. Comprehensions, a query notation for DBPLs. In *Proc DBPL'91*, pages 55–68, 1991.
20. A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc ICDE'97*, pages 91–102, 1997.

An Algebra for the Composition of Ontologies [⋆]

Prasenjit Mitra and Gio Wiederhold

Infolab, Stanford University, Stanford, CA 94305, USA
{mitra, gio}@db.stanford.edu

Abstract. Information needs to be composed from multiple sources to create new information. Such composition is essential, for example, to answer queries posed by end-users on websites enabling electronic commerce. In order to facilitate the composition of information, ontologies have been used to explicitly define the vocabulary used in the information sources. Today, a large number of such ontologies are available. These ontologies in turn are semantically heterogeneous and the need arises to align and compose them before we can compose information from the sources. In this paper, we outline an algebra for the composition of ontologies and show the properties of the algebraic operators. We show how the articulation generation function determines the properties of the operators and discuss the conditions that these functions must satisfy to enable optimized composition of ontologies.

1 Introduction

Often, queries cannot be answered from information obtained from one information source but need information to be composed from multiple sources. Composing information from multiple information sources creates new and useful information. Thus, in a distributed framework like today's World-Wide Web, it is absolutely crucial for us to compose information from multiple sources.

Several approaches to integrate information from diverse information sources have been proposed and implemented with varying success [1], [2], [3], [4] etc.. However, a major problem to integrating information from various sources are the heterogeneity in their vocabularies. In order to solve this problem, ontologies are increasingly being used to explicitly define the terms and relationships used in an information source. Since the ontologies themselves are independently created, we need to resolve the semantic heterogeneity among them. Thus, before we can compose information from information sources, there arises a need to compose their corresponding ontologies [5]. Ontology-based integration of information has been widely proposed.

In this work, we propose an algebra to enable systematic composition of ontologies. Composition of ontologies can then be expressed in terms of expressions using the ontologies and operators from the algebra. The properties of the algebraic operators determine the types of optimization that can be enabled while composing the ontologies. We describe the properties of the composition operations formally and discuss how they can be used to optimize the composition process.

To resolve semantic heterogeneity among ontologies before composing them, we assume that there exists functions that take in two ontologies and generate *articulation rules* among them. An articulation rule establishes the correspondence between two (or more) terms belonging to the different ontologies that are being articulated. Functions that generate articulation rules are called *articulation generation functions*. Articulation rules generated by the articulation generation function form the basis for the composition of ontologies.

The binary operators of the algebra depend upon the articulation rules generated by the articulation generation function. For example, to compute the intersection of two ontologies we need to know which concept in one ontology is similar to which concepts in the other. Not surprisingly, therefore, the properties of the algebraic operators are determined by the articulation generation functions. Most articulation generation functions are manually written or semi-automatically generated using heuristic methods that try to match two ontologies and generate their articulation.

In this paper, we discuss the conditions the articulation generation functions should satisfy so that the composition of the ontologies can be optimized. Even though a lot of work has been done on suggesting heuristics for articulating and aligning ontologies (or mapping schemas) [6], [7], [8], [9], [10], to the best of our knowledge, no prior work has studied the properties of such heuristics and shown how the task of composition of ontologies (or databases) is affected by their properties.

The rest of the paper is organized as follows. In the next section, we discuss some preliminaries. In Section 3, we describe the algebra and its operators. In Section 4, we discuss the properties of the operators and in Section 5, we conclude.

2 Preliminaries

2.1 Ontologies

The term "ontology" has many definitions[11],[12]. It has been represented using various data formats and rule languages. Our approach is to use a definition and data format that is simple - a "least common format". The format captures the basic features common to most machine-represented ontologies and is simple enough to allow easy transformations from various other ontology formats to ours.

In its core, we represent an ontology as a graph and a set of horn-clause rules. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E. The label of a node in V is given by a non-null string. In the context of ontologies, the label is often a noun-phrase that represents a concept.

The label of an edge in E is the name of a semantic relationship among the concepts in V and can be "null" if the relationship is not known. An edge is expressed as a *Statement*

[⋆] This work has been supported by the AFOSR New World Vistas program and the DARPA DAML program.

of the form $(SubjectRObject)$, where $Subject, Object \in V$ and R is the *relationship* between them.

The semantics of the relationships are typically specified in the document it is defined in and the namespace of the relationship is tagged along to clarify the relationship we are referring to. For example, $rdfs : subClassOf$ where $rdfs$ is an alias of $http : //www.w3.org/2000/01/rdf-schema\#$ indicates that the relationship that is being used is "subclassOf" as specified in the $rdfs$ document. In the rest of the paper, we omit the namespace unless we need to differentiate between two relationships of the same name or we need to mention the source of the relationship to make our point.

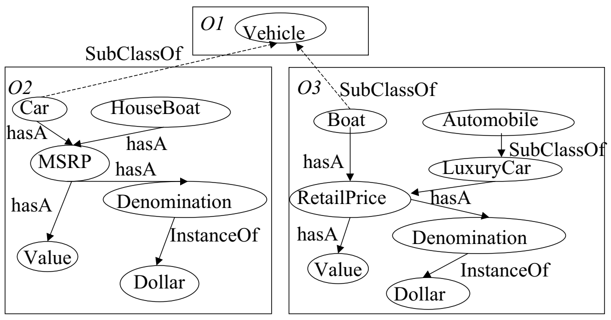
Rules in an ontology are expressed in a logic-based language. Although, theoretically, it might make sense to use first-order logic as the rule language due to its greater expressive power, in order to limit the computational complexity we will use a simpler language like Horn Clauses. A typical rule $r \in R$ is of the form :

$$CompoundStatement \Rightarrow Statement$$

A *Statement* is of the form $(Subject\ R\ Object)$. $Subject$ and $Object$ are either labels of node in the ontology graph or a variable that can be bound to one or more nodes in the ontology graph. As in an edge, the relationship R expresses a relationship between the two concepts $Subject$ and $Object$. The antecedent of the rule $CompoundStatement$ is either a Boolean value ($true$ or $false$) or a conjunction of $Statement$s. If the antecedent of the rule is $true$, we simplify the notation by dropping the body of the rule and writing the consequent as a statement that holds (like edges). Edges can be thought of as rules whose antecedent is always true. Minimally we could define an ontology as a set of concepts and rules (since edges can be expressed as rules), however, we believe the graphical representation aligns with human intuition much better and thus describe our work using the above definition. A more detailed description of the ontology format can be found in [13].

For ease of description of the algebra, we will introduce the following terminology: For a statement $s = (Subject\ R\ Object)$, $Nodes(s)$ contains $Subject(Object)$ provided $Subject(Object)$ is not a variable (that is, it is a node in some ontology graph). For an ontology $O1$, $Nodes(O1)$ represents the set of nodes consisting of each node that belongs to the ontology graph for $O1$. For a set of rules R, $Nodes(R)$ represents the union of $Nodes(s)$ for all s, such that s is a statement in any rule $r \in R$.

Example 1. We introduce a running example in Figure 1, which we will use throughout the paper. $O1$, $O2$, and $O3$ are three ontologies. We only show selected portions of the ontology graphs corresponding to the three ontologies. In order to specify which ontology a concept is defined in, we tag the name of the ontology it belongs to the name of the node. For example, the node labeled $O2.Car$ refers the concept Car as defined in the ontology $O2$. However, where the origin of the definition is not important (or is obvious) to the topic of discussion, we will simply use the concept name without mentioning the fully qualified name (that is,



Articulation Rules:

true => (O2.Car SubClassOf O1.Vehicle)

(X InstanceOf O1.Car),(X hasA X.MSRP),(Y InstanceOf X.MSRP),
(Y hasA Y.Value), (Z InstanceOf Y.Value), (Y.Value > 40,000)
=> (X InstanceOf O2.LuxuryCar)

Fig. 1. Ontologies and Articulation Rules

drop the ontology name tagged in front of it). Let the set $R = \{(O2.Car\ SubClassOf\ O1.Vehicle),$ $(O2.HouseBoat\ SubClassOf\ O1.Vehicle)\}$, then $Nodes(R) = \{O2.Car, O1.Vehicle, O2.HouseBoat\}$.

$Edges(E, n)$, where E is a set of edges and n is a node in an ontology graph, represents all edges in E incident upon or incident from the node n. Formally, $Edges(E, n) = \{s \in E |\ \exists n', l : s = (n, l, n')\ or\ s = (n', l, n)\}$. $Edges(E, N)$, where N and E are a set of nodes and edges respectively in an ontology graph, represents a set of edges $S \in E$. Both nodes (the node from which an edge is incident from and the node to which it is incident upon) of each edge in the set S must belong to the set of nodes N. Formally, $Edges(E, N) = s = (n_1, l, n_2) \in E | n_1, n_2 \in N$.

2.2 Articulation Rules and Articulation Generation Functions

To resolve heterogeneity among ontologies, we need a procedure to generate the correspondences and relationships among concepts in the ontologies. Such relationships are coded as articulation rules. We call such procedures that generate articulation rules between ontologies *articulation generation function*s. Note that the articulation generation function can be an automatic subroutine that takes in the two ontologies and outputs the articulation rules or it could be a manual effort where a human expert identifies the articulation rules by inspecting the two ontologies or a hybrid semi-automatic strategy that involves both a component that automatically suggests articulations and a human component that ratifies them. An articulation generation function takes is two ontologies (domain: the set of all possible ontologies O) and outputs a subset of the set of all possible rules (range: the set of all possible rules R) between them ($f : OxO \rightarrow 2^R$. We expect the articulation generation to be a complete function. That is, given any two ontologies, the function always terminates and outputs a set of articulation rules that link them. An articulation rule r articulating two ontologies $O1$ and $O2$ is such that $\exists O1.n \in Nodes(r)$ and $\exists O2.n' \in Nodes(r)$ for some nodes n and n' in $O1$ and $O2$ respectively.

Example 2. In our running example, we show a few of the articulation rules generated by an articulation generation function. For lack of space, all articulation rules are not shown in

Figure 1, but we show two rules graphically, and two textually at the lower part of the figure. The two graphical rules are shown by dotted arrows spanning different ontologies in contrast to the edges in an ontology indicated by solid arrows. Specifically, we see that $O2.Car$ is related via the relationship $SubClassOf$ to $O1.Vehicle$. Similarly $O3.Boat$ is related via the relationship $SubClassOf$ to $O1.Vehicle$. We show the rule expressing the first relationship both graphically and textually, and the second only graphically. The second articulation rule indicated textually at the bottom of the figure gives a Horn Clause that indicates the relationship between $O2.Car$ and $O3.LuxuryCar$. Any instance of $O2.Car$ that has a $O2.MSRP$ that, in turn, has a $O2.Value$ that is greater than $40,000$ is a $O3.LuxuryCar$. Of course, such a rule should also consider the $O2.Denomination$ of the $O2.MSRP$ but for the sake of simplicity we have omitted the denomination from the rule.

A notation we use is illustrated by $X.MSRP$. Whenever, we instantiate a concept in an ontology, we duplicate

For the sake of describing the algebra below, we now define the terminology related to articulation generation functions.

Definition 1. *An articulation generator function, f,* directly relates *two nodes $n_1 \in O1$, and $n_2 \in O2$, where $O1$ and $O2$ are the source ontologies it is articulating, iff f generates an edge $(n_1 \; R \; n_2)$ or an edge $(n_2 \; R \; n_1)$, where R is a relation.*

Example 3. In our example, the articulation function that generated the articulation rules directly relates the nodes $O2.Car$ and $O1.Vehicle$, and the nodes $O3.Boat$ and $O1.Vehicle$.

If an articulation generation function f directly relates two nodes n_1 and n_2, we will denote that hereafter as $(n_1 f : dreln_2)$.

Definition 2. *An articulation generator function f is said to be* transitively connective *iff it satisfies the following condition:*

– *if* $\forall O1, O2, O3 \in O, \exists r_1, r_2, O1.A, O2.B, O3.C | r_1 \in f(O1, O2), r_2 \in f(O2, O3), O1.A, O2.B \in Nodes(r_1), O2.B, O3.C \in Nodes(r_2)$, *then* $\exists r_3, r_4, O1.D, O3.E | r_3, r_4 \in f(O1, O3), O1.A, O3.E \in Nodes(r_3), O1.D, O3.C \in Nodes(r_4)$

In other words, if the articulation generation function discovers that $O1.A$ is related to $O2.B$ and $O2.B$ is related to $O3.C$, then a transitively connective articulation generation function will discover that $O1.A$ is related to some element in $O3$, and $O3.C$ is related to some node in $O1$.

Example 4. In our running example, if f discovers $O2.Car \; SubClassOf \; O1.Vehicle$ and $O3.Boat \; SubClassOf \; O3.Vehicle$, we expect a transitively connective articulation generator to find some node in $O2$ that is related to $O3.Boat$ (it might generate the edge $(O2.HouseBoat \; SubClassOf \; O3.Boat)$) and some node in $O3$ (presumably both $O3.Automobile$ and $O3.LuxuryCar$) that is related to $O2.Car$.

A desirable property of an articulation generation function is that it is "consistent", that is, presented with the same ontologies it generates the same set of articulation rules always. During the process of ontology composition, parts of an ontology are selected and are composed with other selected portions of ontologies to create *intermediate ontologies*. A node derived from the source ontology will then be aliased with a label comprised of the concept name along with the name of the intermediate ontology. However, both the original name as well as the alias will refer to the same concept. For example, if the node labeled $O1.Car$ in ontology $O1$ is selected and included in ontology $O3$, it can be referred to as $O1.Car$ as well as $O3.Car$.

Definition 3. *An articulation generator function f is said to be* consistent, *if and only if, for any two source ontologies $O1$, and $O2$, and any two intermediate ontologies $O3$ and $O4$ such that* $\forall n_1 \in O1, n_2 \in O2, O1.n_1 \in O3, O2.n_2 \in O4$:
$R1 = f(O1, O2)$,
$R2 = f(O3, O4)$,
$\forall r1 \in R1 | n_1, n_2 \in Nodes(r1)$,
$\exists r2 \in R2 | O1.n_1, O2.n_2 \in Nodes(r2)$, *and* $r1 \equiv r2$.

Let there be three nodes $O1.n1$, and $O1.n1'$ belonging to ontology $O1$, and $n2$, belonging to ontology $O2$. Let us assume an articulation generator function f generates a rule between nodes $n1$ and $n2$, while articulating the source ontologies $O1$ and $O2$. respectively. Let $n1$ be selected to an intermediate ontology $O3$ but $n1'$ is not in $O3$. Also, let $n2$ be selected to an intermediate ontology $O4$. Now, if we want to articulate $O3$ with $O4$, a consistent f must still generate a rule between $n2$ and $n1$ that is equivalent to the rule that it generated while articulating $O1$ and $O2$. A consistent f should generate such a rule despite the absence of some nodes (like $O1.n1'$) or edges that are present in the source ontologies but not in the intermediate ontology.

Example 5. In our running example, let us suppose that an articulation generation function f generates a rule $O3.LuxuryCarSubClassOfO2.Car$ based on its price. However, let us assume that in two intermediate ontologies $O4$ and $O5$ that contain $O3.LuxuryCar$ and $O2.Car$ respectively, only the concepts useful for describing the appearance and functionality of the cars have been chosen but not their prices. Now if f cannot generate the rule between $O3.LuxuryCar$ and $O2.Car$ - which might be perfectly reasonable - we will not call f a consistent articulation generation function.

Obviously, articulation generation functions that bank on structural matches will oftentimes not be consistent, since the structure in the neighbourhood of $n1$ and $n2$ in the intermediate ontology potentially can be much different and not match the structure in the neighbourhood of the nodes in the source ontologies. On the other hand, an articulation generation function f that matches nodes based only on their labels, for example, natural language processing systems that generate similarities of noun phrases, will be consistent since it checks only the label, which for a node remains the same in the source ontologies as well as in the intermediate ontologies.

The property of consistency places a strong restriction on articulation generation functions but as we will see later, articulation generation functions that are consistent enables a sequence of two intersections to be associative. Associativity of operators allow us to compose the ontologies in any order and the rearrangement of operators can be often used to optimize the composition process.

In this work, we do not consider articulation rules that might introduce new nodes. For example, while articulating between $PoundSterling$ and $Guilders$, an articulation generation function might generate an intermediate node called $Euro$ and then give the relation between $PoundSterling$ and the $Euro$ and that between the $Guilder$ and the $Euro$. However, the presence of such an intermediate node influences the properties of the algebraic operators. For example, if an articulation generation function generates intermediate nodes, the intersection operation between ontologies can not be guaranteed to be associative. Thus, we do not consider such articulation generation functions in this work but it is an interesting problem to handle in future.

If an articulation generation function f relates two nodes in two different ontologies using at least one intermediate node, we say that f *indirectly relates* the two nodes (using the intermediate node). We denote this as $(nf : ireln')$, where n, n' are the two nodes in different ontologies connected by the articulation generation function f.

3 The Ontology-Composition Algebra

In order to formalize the task of composing ontologies, we propose an algebra for the composition of ontologies. If we use the algebraic framework to systematically compose ontologies, we can enable optimizations depending upon the properties of the operators. In this section, we describe the ontology-composition algebra and in the next section we discuss properties of its operators.

The algebra has one unary operator: *Select*, and three binary operations:*Intersection*, *Union*, and *Difference*.

3.1 Unary Operator

Select: The Select operator is useful to select portions of an ontology that might be of interest. For example, a person who wants to buy a car and does not care about houseboats, might want to select only portions of ontology $O2$ that contain terminology about cars and cut out the portions that are not related to cars.

Definition 4. *The Select operator has two forms:*

1. *Given an ontology $O = ((N, E), R)$, and a node $n \in N$, $Select(O, n) = (G_n, R_n)$ where $G_n = (N_n, E_n)$ is a subgraph of G such that for all nodes n' in N_n, there exists a path from n to n' in G. The set $E_n = \{e = (n_1\ R\ n_2) \in E | n_1, n_2 \in N_n\}$ is such that each edge in E_n expresses a relationship between nodes n_1 and n_2 where both nodes n_1 and n_2 are in N_n Similarly, $R_n = \{r \in R | Nodes(r) \subseteq N_n\}$ is the set of rules obtained from R containing all rules in O whose concepts are all in N_n.*
2. *Given an ontology $O = ((N, E), R)$, and set of nodes V, $Select(O, V) = (G, R_v)$ where $G = (V, E_v)$. The set $E_v = \{e = (n_1\ R\ n_2) \in E | n_1, n_2 \in V$ and the set $R_v = \{r \in R | Nodes(r) \subseteq V\}$.*

Example 6. In our example, the ontology $O3$ contain the edges $(O3.LuxuryCar\ SubClassOf\ O3.Automobile)$, and $(O3.LuxuryCar\ hasA\ O3.RetailPrice)$. $Select(O3, Automobile)$ would select all nodes reachable from the node $O3.Automobile$ $(LuxuryCar, RetailPrice, Denomination, Value,$ and $Dollar)$, and the edges between them. $Select(O3, \{Automobile, LuxuryCar\})$ would on the other hand only select the nodes $O3.LuxuryCar, O3.Automobile$ and the edge $(O3.LuxuryCar\ SubClassOf\ O3.Automobile)$.

Note that a rule r in R that does not involve any node in O has $Nodes(r) = \phi$. Such a rule is included in the selected ontology since the empty set is a subset of N. For example, a rule expressing the transitivity of the relationship $SubClassOf$

$$(XSubClassOfY), (YSubClassOfZ) \Rightarrow (XSubClassOfZ)$$

contains only variables X, Y, and Z and no concepts from any ontology. Such a rule is included in any selected ontology S since the rule might be useful to reason about the relationships and concepts in S.

There is a case that could be made to include in the results of the select operation edges (and rules) that can be derived using the edges and rules available in the source ontology. For example, let us suppose that we had edges $(LuxuryCar\ SubClassOf\ Car)$, and $(Car\ SubClassOfVehicle)$ in an ontology O. $Select(O, \{Vehicle, Car\})$ would select the last edge. On the other hand $Select(O, Vehicle, LuxuryCar)$ would include the nodes $Vehicle$, and $LuxuryCar$ but no edges since there are no edges between them. We could define Select to add an edge $(LuxuryCar\ SubclassOf\ Vehicle)$ if such a relationship could be derived from the ontology O, for example, using a rule that said that the relationship $SubClassOf$ is transitive.

Similarly, it is easy to see that we could introduce additional rules over and above the ones that the current *Select* operation includes in the selected ontology. However, in order to generate these derived edges and rules, the ontology composition engine would need to interpret the rules of the ontology. In order to allow our framework to be applicable to different ontologies with different interpretation semantics for rules and because potentially we could derive an infinite number of facts in certain scenarios (say with recursive rules), the ontology composition engine does not interpret the rules so as to maintain its simplicity.

3.2 Binary Operators

Intersection Each binary operator takes as operands two ontologies that we want to articulate, and generates an ontology as a result, using the articulation rules. The articulation rules are generated by an articulation generation function.

Intersection is the most important and interesting binary operation. The intersection of two ontologies $O1 = ((N1, E1), R1)$, and $O2 = ((N2, E2), R2)$ with respect to the set of articulation rule generating function f is:
$OI_{1,2} = O1 \cap_f O2$, where $OI_{1,2} = (NI, EI, RI)$,
$NI = Nodes(f(O1, O2))$,

$EI = Edges(E1, NI \cap N1) + Edges(E2, NI \cap N2) + Edges(f(O1, O2))$,
and $RI = Rules(O1, NI \cap N1) + Rules(O2, NI \cap N2) + f(O1, O2)))$.
The nodes in the intersection ontology are those nodes that appear in the articulation rules. An edge in the intersection ontology are the edges among the nodes in the intersection ontology that were either present in the source ontologies or have been output by the articulation generation function as an articulation rule. The rules in the intersection ontology are the articulation rules that are present in the source ontology that use only concepts that occur in the intersection ontology.

Note that since we consider each node as an object instead of the subtree rooted at the node, we will get only the node in the intersection by virtue of its appearing in an articulation rule and not automatically include its attributes or subclasses. Again, a minimal linkage keeps the intersection ontologies small and avoids the inclusion of possibly irrelevant concepts. Inclusion of attributes will be required to define subclass relationships among nodes in the source ontologies precisely.

Each node in the intersection has a label which contains the URI of the source in which it appears. If the attributes of the object that it represents are required, the application's query processor has to get that information from the original source. Defining the intersection with a minimal outlook reduces the complexity of the composition task, and the maintenance costs, which all depend upon the size of the articulation.

Union The union OU between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ is expressed as $OU = O1 \cup_{AR} O2 = (VU, EU, RU)$ where
$VU = V1 \cup V2 \cup VI_{1,2}$,
$EU = E1 \cup E2 \cup EI_{1,2}$,
and $RU = R1 \cup R2 \cup RU_{1,2}$,
and where $OI_{1,2} = O1 \cap_{AR} O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies.

The set $VI_{1,2}$ is non-null only if the articulation rules do not introduce new nodes. However, even if two nodes in the ontologies being articulated have the same label, say, $O1.x$ and $O2.x$ and f indicates that they represent the same concept by generating the rule $O1.x\ Equals\ O2.x$, in the intersection and the union ontology, we would retain both $O1.x$ and $O2.x$ and the edge between them instead of collapsing them into one node. The articulation rules almost always indicate relationships between nodes in the two source ontologies and thus introduces new edges (the set $EI_{1,2}$) that were not there in the source ontologies.

Though queries are often posed over the union of several information sources, we expect this operation to be rarely applied to entire source ontologies. The union of two source ontologies is seldom materialized, since our objective is not to integrate source ontologies but to create minimal articulations and interoperate based on them. However, we do expect that larger applications will often have to combine multiple articulations and here is where the union operation is handy.

Difference The difference between two ontologies $O1$ and $O2$, written as $O1 - O2$, between two ontologies $O1 = ((V1, E1), R1)$ and $O2 = ((V2, E2), R2)$ is expressed as $OD = ((VD, ED), RD)$, where $VD = V1 - VI_{1,2}$, $ED = E1 - EI_{1,2}$, and $RD = RI - RI_{1,2}$, and where $OI_{1,2} = O1 \cap_f O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies using the articulation generation function f. That is, the difference ontology includes portions of the first ontology that are not common to the second ontology. The nodes, edges and rules that are not in the intersection ontology but are present in the first ontology comprise the difference.

3.3 Properties of the Operators

We defined the operators in the algebra on the basis of the articulation rules produced by the articulation generating function. Not surprisingly, most of the properties of the binary operations are based on the properties of the articulation generating function.

Idempotence Idempotence is a very important property that should hold for operations like union and intersection. Intuitively, we believe that the union (and intersection) of an ontology with itself should result in the same ontology.

For the union operation to be idempotent to hold, we require $O1 \cup_f O1 = O1$. Also, let $OU = O1 \cup_f O1$. Let $AR = f(O1, O2)$. From the definition of the union it follows that $Nodes(OU) = Nodes(AR)$. For union to be idempotent, we need $Nodes(OU) = Nodes(O1) = Nodes(AR)$. That is, f must generate articulation rules involving all nodes in the ontology $O1$. This requirement is obvious since we are matching the same ontology to itself and would expect a set of rules to be generated that matches each node to itself.

Although, the nodes in OU and $O1$ are the same, the former contains more edges and rules by virtue of the definition of the $union$ operator. The extra edges are self-edges between a node and itself generated by f that are included in the union ontology.

Since the edges and rules of OU and $O1$ are not the same, $OU \neq O1$, and the $Union$ operator is not idempotent. Although the strict definition of idempotence is not satisfied, we introduce the concept of *semantic idempotence*. An binary operator is semantically idempotent if and only if the result of the operation on two copies of the same ontology results in an ontology from which if we remove all self-edges and self-rules, the resulting ontology is the same as the source ontology. Recall that self-edges are edges between one node and itself and a self-rule is a rule that has only one node (concept) in its definition.

Similarly, we can show that the intersection operator is semantically idempotent, if and only if, the articulation generation function that generated the articulation rules between an ontology and itself generates self-edges and self-rules for each node in the ontology.

It is easy to see that the difference operator is not idempotent but a difference operation using a well-behaved articulation generation function should return the empty set.

The semantic idempotence property serves as a sanity check to make sure that the articulation generation functions that we employ are semantically meaningful and intuitively correct. Although, in order to guarantee that an articulation generation function is idempotent, we would have to check over the set of all (potentially infinite) ontologies, in practice we simply check to see that for the purposes of an application, the articulation generation function is idempotent for all ontologies used by the application. That is, though we do not prove the functions to be idempotent, we check to see that none of the ontologies that is useful for the application can be used to prove that the function is not idempotent. Articulation generator functions that do not satisfy the above equality are *unsound* and for the purposes of our compositions, we do not use any unsound articulation generator function.

Commutativity Commutativity of operators is another important property. If an operator is commutative, the optimizer that is generating a plan for the composition of the ontologies has the freedom to flip the order of the operands if it sees that an optimization can be achieved by flipping the order.

Theorem 1. *The intersection and union operators are commutative if and only if the articulation generation function, on which they are based, is commutative.*

The proofs of the theorems are not included in this paper for space considerations but are provided in [14]. However, strict commutativity of the articulation generation function might not be achievable or necessary in order to allow the operands to be swapped.

Consider the example where an articulation generator generates articulation rules

$$f(O1, O2) = (O1.Car\ NM.SubClassOf\ O2.Vehicle)$$

and

$$f(O2, O1) = (O2.Vehicle\ NM.SuperClassOf\ O1.Car)$$

where $O1$, $O2$ are two ontologies and *NM* refers to the namespace where the semantics of the relationships $SubClassOf$ and $SuperClassOf$ are defined. Although the articulation generation function is not commutative, the semantic information contained in the articulation rules are equivalent as long as the relations $SubClassOf$ and $SuperClassOf$ defined in the namespace *NM* are semantically similar after we invert their parameters. Thus, if the rules obtained by swapping the operands are semantically equivalent, we can swap the operands without compromising on the correctness of the result.

To capture this, we define the concept of *semantic commutativity*.

Definition 5. *An articulation generation function, f, is* semantically commutative *iff*
$f(O1, O2) \Leftrightarrow f(O2, O1) \forall O1, O2$*, where $O1$, and $O2$ are ontologies.*

and the necessary and sufficient condition for intersection to be semantically commutative is:

Theorem 2. *An intersection operator is semantically commutative iff the articulation generation function that it uses to derive the articulation rules is semantically commutative.*

To determine the semantic commutativity of an articulation generation function, we need to prove that for any pairs of ontologies, the articulation rules produced by the articulation generation function are in fact semantically equivalent. Automatically proving an articulation generator commutative or semantically commutative might be easy for the $SubClassOf$ and $SuperClassOf$ examples, but is not always feasible. In such cases, the system requires the programmer of the articulation generation function and/or the expert to indicate whether the function is semantically commutative. In the absence of such information, the system conservatively assumes that the operation is not semantically commutative if it cannot prove otherwise.

We have identified the desired properties that a ”well-behaved” articulation generation function should have so that query optimization can be enabled. The algebra plays an important part in influencing query rewriting, optimization and systematic composition of information that forms the basis for enabling interoperation among information sources.

Associativity If the operators are associative, then the order of their execution can be changed and we can optimize the execution of the query based on available statistics about the size of the ontologies and their intersections. Therefore, associativity is a very desirable property to have for the operators.

Example 7. For example, if we need to compute $(O1 \cap_f O2) \cap_f O3$ where $O1$ and $O2$ are very large ontologies and $O3 = null$. The resulting intersection will obviously be null. If the intersection operator with the articulation generator f was associative, we could rewrite the above composition as $O1 \cap_f (O2 \cap_f O3)$. If a query planner chose the former plan to execute, it would spend a lot of time articulating the first two large ontologies and then articulate it with a null ontology. On the other hand, if the intersection operator was associative, an intelligent query planner that knows the sizes of the ontologies could opt for the second formula since in that case we articulate a large ontology with the null ontology and then articulate a null ontology with the other large ontology. The presence of the null ontology in both articulations should make the second plan faster and require less storage for intermediate steps.

Intersection We need to show $(O1 \cap_{ARules} O2) \cap_{ARules} O3 = O1 \cap_{ARules} (O2 \cap_{ARules} O3)$.

Theorem 3. *The intersection operator using an articulation generation function f is associative iff f is consistent and transitively connective.*

A stricter form of a transitively connective articulation generator is one that is transitive.

Example 8. Consider the example $O1 = (car, null, null)$, $O2 = (truck, null, null)$ and $O3 = (vehicle, null, null)$. That is, the three ontologies simply have one node each and no edges or rules. Let f be an articulation generation function that is not transitively connective. The the articulation rules between $O1$ and $O2$ be as follows: $f(O1, O2) = null$. Articulating this null ontology with $O3$ returns null. Therefore, $(O1 \cup_f O2) \cup_f O3 = null$ On the other hand, let the articulation rules $f(O2, O3) = (O2.Truck, SubclassOf, O3.Vehicle)$. Therefore,

$(O2 \cup_f O3) = OI_1 = ((O2.Truck, O3.Vehicle),$ $((O2.truck, SubclassOf, O3.Vehicle)), null)$. Intersecting OI with ontology $O1$, we have $(O1 \cup_f (O2 \cup_f O3)) =$ $OI_2 = ((O1.Car, O3.Vehicle),$ $((O1.Car, SubclassOf, O3.Vehicle)), null)$. Since OI_1 and OI_2 are not the same, we see that the intersection is not associative but depends upon the order of the operand ontologies.

In a lot of situations in practice, the person wishing to compose the two ontologies does not necessarily have a preference in the order in which the ontologies are composed. In such a scenario, the composer might instruct the system to assume associativity (and thus enable rearranging of the operands), even though the articulation generation function is not provably strictly transitively connective (and thus the intersections are not entirely independent upon the order of the operands).

Union The $Union$ operation is associative if and only if $\forall O1, \forall O2, \forall O3 | (O1 \cup_f (O2 \cup_f O3)) = ((O1 \cup_f O2) \cup_f O3)$ where f is the articulation generation function and $O1$, $O2$, and $O3$ are ontologies.

Theorem 4. *The Union operation is associative if and only if the associated articulation generation function f is consistent.*

A consistent articulation generation function f is a necessary and sufficient condition for $Union$ with respect to f to be associative.

Difference Difference is not associative under any conditions. We need to show that $O1 - (O2 - O3)$ is not equal to $(O1 - O2) - O3$. Let $O1, O2, O3$ contain only one node each: n_1, n_2, and n_3 (respectively). Let f generate rules $Rule(n_1, n_2), Rule(n_1, n_3), Rule(n_2, n_3)$ while articulating $O1$ and $O2$, $O1$ and $O2$, and $O2$ and $O3$ respectively. Using the definition of difference, $(O2 - O3)$ is a null ontology since the only nodes in $O2$ and $O3$, namely n_2 and n_3 are related by f. Thus $O1 - (O2 - O3) = O1$. Again $(O1 - O2)$ is a null ontology since the only nodes in $O1$ and $O2$, namely n_1 and n_2 are related by f. Therefore, $(O1 - O2) - O3$ is a null ontology. Thus, difference is not associative.

Associativity and Articulation Generation Functions It follows from the necessary and sufficient conditions that the intersection operation is not associative when the generation of the articulation rules is done by a function that bases its decision on the context of the nodes. Functions like structural matchers depend upon the presence of certain edges between nodes in an ontology. Now, the process of intersection might not copy all edges in the neighbourhood of a node into the intersection ontology. Thus, when the intersection ontology is composed with another ontology, the same articulation rules might not be generated as would have been if the source ontology was composed with the second ontology.

Therefore, if the articulation generation function depends upon structural matchers are employed, the optimization needs to be disabled unless the matcher is guaranteed to satisfy the conditions mentioned above (primarily consistency and transitive connectiveness).

Articulation generation functions that are based on linguistic matchers just look at the words appearing to describe the concept and thus often satisfy such conditions.

With respect to optimizing compositions of ontologies, we prefer linguistic matchers to structural matchers. However, if the structural matchers generate significant semantic matches that are necessary for the application, we just have to pay the price and not optimize the composition process.

Distributivity Similarly, we show that with a consistent articulation generation function the $\cup_f$ and the $\cap_f$ operations are distributive as stated below. The distributivity property, again, enables the ontology composition engine to rewrite and reorder a composition task to optimize its performance.

Theorem 5. *: $(O1 \cup_f O2) \cap_f O3)$ = $(O1 \cap_f O3) \cup (O2 \cap_f O3)$ iff the articulation generation function f is consistent.*

4 Conclusion

In this paper, we propose an algebra for the composition of ontologies. The algebraic operators closely depend upon the properties of an articulation generation function that generates the rules on which the algebra is based. Optimizations can be enabled if the operations are commutative and associative. We identified the necessary and sufficient conditions that articulation generation functions need to satisfy before the optimizations can be turned on. As an important corollary, we find that linguistic matchers are more well behaved than structural matchers and allow more scope for optimizing the process of composing ontologies.

References

1. The stanford-ibm manager of multiple information sources http://www-db.stanford.edu/tsimmis/
2. Information integration using Infomaster http://infomaster.stanford.edu/infomaster-info.html
3. Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The information manifold. In Knoblock, C., Levy, A., eds.: Information Gathering from Heterogeneous, Distributed Environments, Stanford University, Stanford, California (1995)
4. C. H. Goh, S. E. Madnick, M.D.S.: Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html
5. Wiederhold, G.: An algebra for ontology composition. In: Monterey Workshop on Formal Methods. (1994) 56–61
6. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proceedings of the Twelfth International Conference on Data Engineering, San Jose, CA, IEEE Computer Society (2002)
7. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources: A machine-learning approach. In: SIGMOD 2002. (2001)
8. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic schema matching with cupid. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, Morgan Kaufmann (2001) 49–58
9. Noy, N., Musen, M.: Prompt: Algorithm and tool for automated ontology mergin and alignment. In: Seventh National Conference on Artificial Intelligence (AAAI-2000). (2000)

10. McGuiness, D., R.Fikes, Rice, J., Wilder., S.: The chimaera ontology environment. In: Seventh National Conference on Artificial Intelligence (AAAI-2000). (2000)
11. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. In Guarion, N., ed.: Padua Workshop on Formal Ontology. (1993)
12. Sowa, J.F.: http://users.bestweb.net/ sowa/ontology (2000)
13. Mitra, P., Kersten, M., Wiederhold, G.: A graph-oriented model for articulation of ontology interdependencies. In: Advances in Database Technology- EDBT 2000. Lecture Notes in Computer Science, 1777, Springer-Verlag (2000) 86–100
14. Mitra, P.: An algebra for semantic interoperation of information sources, http://www-db.stanford.edu/ prasen9/alg.txt. Technical report, Infolab, Stanford University (2001)

Knowledge Representation and Transformation in Ontology-based Data Integration*

Silvana Castano and Alfio Ferrara

University of Milano
DSI - Via Comelico, 39
20135 Milano, Italy
{castano,ferrara}@dsi.unimi.it

Abstract. The vision of the Semantic Web envisages the Web enriched with several domain ontologies specifying formal semantics of data for various intelligent services. The paper describes an ontology architecture for integration of heterogeneous XML data sources, where information about DTDs and their contents are represented at a semantic level by means of a semantic mapping scheme and a mediation scheme. In the paper, we first describe information transformation techniques for designing such ontology schemes with heterogeneous XML data. Then, we present a rule-based approach for automatic derivation of a DAML+OIL representation of the ontology knowledge, to enable ontology interoperability in Semantic Web.

1 Introduction

The vision of the Semantic Web envisages the Web enriched with several domain ontologies, which specify formal semantics of data, for different intelligent services for information search, retrieval, and transformation. In this paper, we focus on ontologies for the integration of heterogeneous XML data sources. The advent of the Web and of XML has dramatically increased the need for efficient and flexible mechanisms to provide integrated views over multiple heterogeneous information sources, and semantic integration of XML data sources has recently received special attention in the database community [1,2]. Research in this field is concerned with the development of methods and tools for information integration with respect to existing semantic heterogeneity and for semantics management to capture the meaning of XML data appropriately [3,4].
In this paper, we consider our reference architecture of domain ontology for XML datasource integration [5], where information about DTDs and their contents are represented at the global level by means of a *semantic mapping scheme* and a *mediation scheme*. Design techniques for building the domain ontology in a semiautomated way have been developed in the framework of the ARTEMIS/MOMIS system [6,7,8]. In this paper, we first summarize information transformation issues in ontology design, for abstracting XML DTDs into a high level representation formalism (called *X-formalism* [3]), and the schema matching and unification techniques for deriving the semantic mapping scheme and the mediation scheme. Then, we introduce a rule-based approach for the representation of the ontology knowledge in the DAML+OIL ontology language standard proposal [9,10]. In this way, ontology knowledge can be exported to Semantic Web in a uniform and standardized way, ensuring ontology interoperability.
The paper is organized as follows. In Section 2, we describe the reference ontology architecture for XML data integration. In Section 3, we describe the reference ontology formalism for XML datasource integration and the associated ontology design techniques. In Section 4, we describe ontology knowledge representation in DAML+OIL. In Section 5, we discuss related work on data integration. Finally, Section 6 presents concluding remarks and future work.

2 Ontology-based XML data integration

We consider a Web-based scenario, where XML is the standard adopted for data exchange among different datasources in a given domain. We assume that, for each datasource, data to be integrated are described by means of a DTD, the most commonly used type description for XML data. In this scenario, we follow a semantic approach to information sharing and integration by setting up a *domain ontology*, organizing knowledge about datasource DTDs and their contents (e.g., meaning of elements, sub-elements, attributes, inter-schema properties), according to a *semantic mapping scheme*, and a *mediation scheme*. Each scheme constitutes a layer of the ontology, provides a semantic view of the different datasources, and is organized as a network of *concepts* and *links*.
In the semantic mapping scheme network, we have sets (clusters) of semantically related datasource concepts, formally represented as *X-classes* defined according to the *X-formalism* we adopt for abstracting DTDs [3]. In the mediation scheme network, we have global concepts, formally represented as global X-classes, providing an integrated representation of underlying datasource concepts. A global concept has associated *mapping rules* specifying how to map the structure of the global concept to the structure of the datasource concepts in its associated cluster.

Semantic links expresses semantic relationships between concepts at a given layer (*intra-layer links*) and also between concepts at different layers (*inter-layer links*). An intra-layer link expresses an equivalence relationship between similar datasource concepts forming a cluster in the semantic mapping layer, or a generic semantic relationship between global concepts in the mediation scheme layer, or an is-a relationship between global concepts in the mediation scheme layer. An inter-layer link expresses the relationship between a cluster and the global concept representative of the cluster at the mediation scheme layer above.

* This paper has been partially funded by MIUR COFIN2000 D2I project.

```
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ELEMENT CHANNEL (BANNER, DAY+)>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY ((DATE, HOLIDAY) | (DATE, PROGRAMSLOT+))+>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME, TITLE, DESCRIPTION?)>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
<!ELEMENT DESCRIPTION (#PCDATA)>
```

http://videdot.com/project-report.html **(a)**

```
<!ELEMENT TVSCHEDULE (DATE, TVSTATION*)>
<!ELEMENT DATE EMPTY>
<!ATTLIST DATE day CDATA #REQUIRED
         month CDATA #REQUIRED
         year CDATA #REQUIRED>
<!ELEMENT TVSTATION (PROGRAMME*)>
<!ATTLIST TVSTATION name CDATA #REQUIRED
         subscription (yes | no) ’no’
         adult (yes | no) ’no’>
<!ELEMENT PROGRAMME (TIME, TITLE, VIDEOPLUS?, DESCRIPTION?)>
<!ATTLIST PROGRAMME language (irish | english | french |
         german) ’english’
         genre (general | drama | film |
                  science | quiz |
                  documentary | sport | chat |
                  politics | comedy | news |
                  soap | music | children) ’general’
         subtitles (yes | no) ’yes’
         signlanguage (yes | no) ’no’
         repeat (yes | no) ’no’>
<!ELEMENT TIME EMPTY>
<!ATTLIST TIME hour CDATA #REQUIRED
                  minute CDATA #REQUIRED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT VIDEOPLUS (#PCDATA)>
<!ELEMENT DESCRIPTION (TEXT?, CLIP?, FEATURES?, FILM?)>
<!ATTLIST DESCRIPTION rating
                  (NA | U | PG | 12 | 15 | 18) ’NA’
                  starrating (NA |1 | 2 | 3 | 4 | 5) ’NA’>
<!ELEMENT TEXT (SHORT?, LONG?)>
<!ELEMENT SHORT (#PCDATA)>
<!ELEMENT LONG (#PCDATA)>
<!ELEMENT CLIP (AUDIO*, STILL*, VIDEO*)>
<!ELEMENT AUDIO (#PCDATA)>
<!ATTLIST AUDIO source CDATA #REQUIRED>
<!ELEMENT STILL (#PCDATA)>
<!ATTLIST STILL source CDATA #REQUIRED>
<!ELEMENT VIDEO (#PCDATA)>
<!ATTLIST VIDEO source CDATA #REQUIRED>
<!ELEMENT FILM (ACTOR*, DIRECTOR?)>
<!ATTLIST FILM colour (yes | no) ’yes’
                  year CDATA #REQUIRED>
<!ELEMENT ACTOR (#PCDATA)>
<!ELEMENT DIRECTOR (#PCDATA)>
<!ELEMENT FEATURES (PRESENTER*,GUEST*)>
<!ELEMENT PRESENTER (#PCDATA)>
<!ELEMENT GUEST (#PCDATA)>
```

http://student.cs.ucc.ie/mmedia00/ak1/theWeb.html **(b)**

Fig. 1. An example of DTDs of two real datasources in the TV entertainment domain

The ontology modelling schemes give raise to a semantic search space where users can browse on concept networks at different layers through intra-layer semantic links, and move across layers through inter-layer links. The ontology supports also mediator services for query formulation and processing, to correctly answer a query formulated on the structure of a global concept. By exploiting the mapping rules, such a query can be reformulated by the mediator using the specific terminology and structure of each datasource concept involved in the definition of the global concept. Source data can then be retrieved and merged by the mediator according to the global concept structure for presentation to the user.

3 Ontology design

The domain ontology is built according to a bottom-up process, articulated in the following steps: (1) **DTD abstraction**, where DTDs associated with datasources are abstracted into a set of X-classes according to a reference formalism called X-formalism; (2) **semantic DTD matching**, where X-classes of different DTDs are grouped on the basis of semantic mappings established between them; (3) **mediation scheme design**, where X-classes belonging to the same cluster are abstracted and unified into *global X-classes*. Moreover, mapping rules are defined between global X-classes and the X-classes in the corresponding clusters to map global X-classes structure onto the structure of original XML datasources. In this section, we first summarize X-formalism and then the schema matching and unification techniques for constructing the domain ontology schemes. A detailed description of such techniques can be found in [8,7,3].

3.1 Reference formalism for DTD abstraction

To build the domain ontology, we assume that XML data to be integrated have associated a DTD, the most commonly used type description for XML documents[1]. DTDs are abstracted into a set of X-classes according to the X-formalism [3], capable of describing at a high abstraction level the contents of a given DTD, by capturing the semantics of various elements within the DTD through the concepts of X-class, property, attribute, link, and referenced X-class. In X-formalism, an X-class *xc* corresponds to an element declaration with element content in XML, and is a 6-tuple of the form *xc*=(n_{xc}, s_{xc}, P_{xc}, L_{xc}, R_{xc}, A_{xc}), where: n_{xc} is the class name; s_{xc} is the class structure; P_{xc} is a set, possibly empty, of properties; L_{xc} is a set, possibly empty, of links; R_{xc} is a set, possibly empty, of referenced X-classes (i.e., X-classes whose name appears in s_{xc}); A_{xc} is a set, possibly empty, of attributes. A property p of an X-class corresponds to an element with `PCDATA`, `any`, or `empty` content in XML, and is a triple of the form p=(n_p,t_p,A_p) where n_p is a name appearing in s_{xc}, $t_p \in \{\texttt{PCDATA,empty,any}\}$ is the property type, and A_p is a set, possibly empty, of attributes. An attribute a associated with an X-class (property or link, respectively) corresponds to an attribute defined in the attribute list declaration of an element in XML, and is a triple of the form a=(n_a,t_a,k_a), where n_a is the attribute name, t_a is the attribute type, and k_a is the attribute cardinality. A link l corresponds to an *XLink element* in XML, and is represented as a class name in s_{xc} or as an attribute of a property/class in ($P_{xc} \cup R_{xc}$). A referenced X-class r corresponds to an element with element content in the structure of a considered DTD, and is an X-class whose name appears in s_{xc}. In the remainder of the paper, the term “feature” is used to denote a property, link, referenced X-class, or attribute of a given X-class. The cardinality symbols used in the XML DTD syntax are used to derive cardinality constraints for X-class features. In particular, ‘+’ corresponds to the cardinality constraint (1,n), ‘*’ to (0,n), and ‘?’ to (0,1), respectively. If no symbol is specified, the cardinality constraint (1,1) is taken. In the X-formalism representation, an X-class, property, and attribute is represented as a rectangular, oval, and double oval node, respectively. A link is represented as an oval (or rectangular, depending on its type) labeled with the name of the link and connected by a double directed arc to the appropriate X-class node. There is an arc between two nodes if there is a containment relationship between them. An XML DTD union structure (‘|’) is represented as an `or`-labeled dashed arc crossing the arcs enter-

[1] X-formalism has been recently extended to support other schema definition formalisms for XML, such as XML Schemas or DSDs [11].

ing the class/property nodes involved in the union. An XML DTD sequence structure with a cardinality constraint applying to the whole structure is represented as an `and`-labeled arc crossing the arcs entering the involved X-class/property nodes. A description of the basic X-formalism concepts and of their graphical representation is shown in Table 1. Finally,

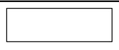
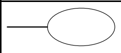
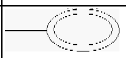
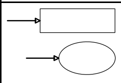
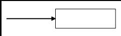
X-formalism Concept	Graphical representation	Description
X-class		Corresponds to an element declaration with element content in a DTD.
Property		Corresponds to an element with `PCDATA`, `any`, or `empty` content in a DTD.
Attribute		Corresponds to an attribute defined in the attribute list declaration of an element in a DTD.
Link		Corresponds to an *XLink* element in a DTD.
Referenced X-class		Corresponds to an element with element content in the structure of a DTD.

Table 1. Basic X-formalism concepts

cardinality constraints associated with properties, links, attributes, and referenced X-classes are explicitly represented, if different from $(1, 1)$, as labels attached to arcs; $(1, 1)$ is implicitly taken as a default.

For instance, consider the DTDs illustrated in Figure 1 describing information about two real TV entertainment datasources. The X-classes of the first DTD are `TvSchedule`, `Channel`, `Day`, and `Programslot`, while the X-classes of the second DTD are `TvSchedule`, `TvStation`, `Programme`, `Description`, `Film`, `Features`, `Text`, and `Clip`. The X-formalism graphical representation of the two DTD is shown in Figure 2 and in Figure 3, respectively.

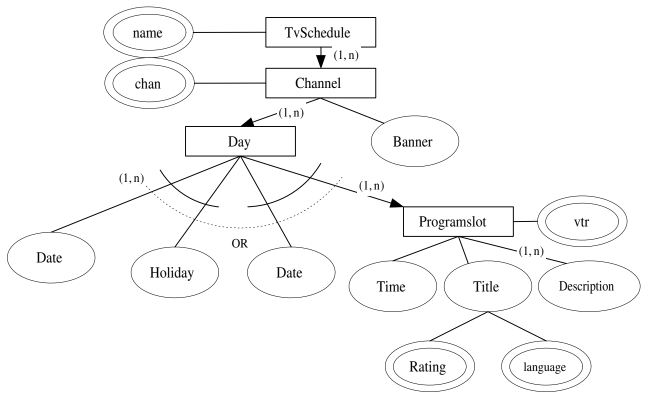


Fig. 2. X-formalism representation of the DTD in Fig. 1 (a)

3.2 Ontology design techniques

Schema matching techniques. The first step in the construction of the ontology consists in defining the semantic mapping scheme. This involves the comparison of X-classes of DTDs associated with different datasources to find similarities between them. To this purpose, ARTEMIS provides schema matching techniques based on the concept of *affinity* to identify X-classes having a semantic relationship. Affinity evaluation in ARTEMIS exploits a thesaurus of weighted terminological relationships (e.g., synonymy, hyperonymy) expressing inter-schema knowledge characterizing source DTDs. In particular, a validated thesaurus (called Common Thesaurus in ARTEMIS/MOMIS) is defined containing the following kinds of weighted terminological relationships: i) automatically extracted from WordNet for names of X-classes and features (basic ontological knowledge); ii) automatically extracted considering X-classes and structure (schema knowledge); iii) manually supplied by the designer (domain knowledge). Terminological relationships capture interschema knowledge between X-classes of different DTDs both at the intensional level and at the extensional level (i.e., with reference to possible kinds of instance overlappings). We represent the thesaurus as a network, where nodes correspond to names (of X-classes and features) and labeled, weighted edges between nodes correspond to terminological relationships. A semantic mapping is established between X-classes and their features if their names have affinity according to the terminological relationships in the thesaurus. Two names n and n' have affinity if there exists at least one path between them in the thesaurus and the strength of the path is greater than or equal to a given threshold. The highest the number of attributes, properties, links, and referenced X-classes with affinity, the higher the strength of the semantic mapping between two X-classes. A hierarchical clustering algorithm is used to find clusters of semantically related X-classes based on the semantic mapping strengths [8]. The interested reader can refer to [7,8] for a detailed discussion of affinity-based schema matching and some experimental results. For instance, with reference to the DTDs in Figures 2 and 3, ARTEMIS determines three clusters containing X-classes of both sources: cl_1 = {`S1.Tvschedule`,`S2.TVschedule`}, cl_2 = {`S1.Programslot`,`S2.Programme`}, and cl_3 = {`S1.Channel`,`S2.TVstation`}. remaining X-classes are placed in singleton clusters.

Unification techniques. The subsequent step of the ontology design is the definition of the mediation scheme. This is performed through rule-based unification techniques by which X-classes belonging to a given cluster are reconciled into *global X-classes*. The mediation scheme design process proceed as follows. In a first phase, we consider clusters of X-classes and we define preliminary global X-classes out of them, which constitute the skeleton global concepts of the mediation scheme. For each global X-class, in this stage we identify its basic features, namely, attributes, properties, and links. We leave referenced X-classes pending in this stage, to be fixed after completing this phase of the process. The second phase of the mediation scheme design process is devoted to revise preliminary global X-classes, to identify their structure and links. Finally, mapping rules

PROGRAMME	S1.Programslot	S2.Programme
ATTRIBUTES		
Language	language CDATA #IMPLIED	language (irish | english | french | german) 'english'
...	...	...
PROPERTIES		
Time	<!ELEMENT TIME (#PCDATA)>	<!ELEMENT TIME EMPTY>
Genre	NULL	genre (general | drama | film | science | quiz | documentary | sport | chat | politics | comedy | news | soap | music | children) 'general'
...	...	...
REFERENCED X-CLASSES		
Description	<!ELEMENT DESCRIPTION (#PCDATA)>	<!ELEMENT DESCRIPTION (TEXT?, CLIP?, FEATURES?, FILM?)>

Table 2. Example of mapping table for the global X-class `Programme`

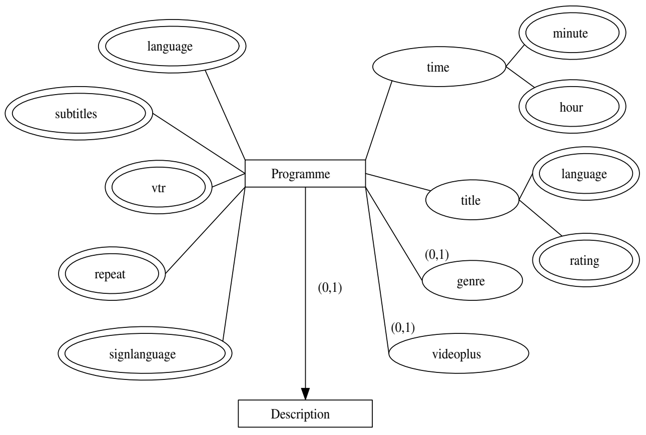


Fig. 4. Example of global X-class

between global X-classes and corresponding local X-classes belonging to the associated cluster are defined. The global X-class obtained from a singleton cluster coincides with the X-class in the cluster, that is, it has the same name and the same features. For clusters containing more than one X-class, the corresponding global X-class is obtained by reconciling properties, attributes, and referenced classes of the X-classes belonging to it by applying a set of basic *reconciliation rules* whose purpose is to unify names, types, and cardinality constraints. Two kinds of features are considered in unifying X-classes of a cluster, namely *homologous* features (i.e., attribute-attribute and property-property) and *non-homologous* ones (i.e., attribute-property and property-referenced X-class). This is due to the fact that the same concept can be represented in different ways in different DTDs, using XML elements of different kind and complexity. The reconciliation procedure first processes homologous features (i.e., attributes, properties, Xlink attributes). The reconciliation of non-homologous features is postponed till all homologous features have been reconciled and a mediated feature representative of all them has been obtained. A detailed description of the reconciliation rules for mediation scheme design is described in [5]. As an example, the global X-class `Programme` obtained from reconciliation of X-classes of cluster cl_2 is shown in Figure 4. To complete the global X-class definition, *mapping rules* are specified. For each global X-class, a persistent *mapping-table* storing all the mapping rules is generated; it is a table whose columns represent the set of the local X-classes belonging to the cluster associated with a given global X-class and whose rows represent the global X-class features. For a given row, each entry of the table contains the XML syntax of the corresponding feature of the involved local X-class in the cluster. An example of mapping rules for the global X-class `Programme` is shown in Table 2.

4 Ontology knowledge representation

In order to increase ontology interoperability for Semantic Web, a RDF and RDFSchema representation of the knowledge in our ontology is required. To this purpose, we describe ontology scheme knowledge representation using DAML+OIL. In particular, we provide a set of rules to automatically derive the DAML+OIL description of the ontology semantic mapping scheme and mediation scheme from the X-formalism formalization.
First, we define the basic DAML+OIL concepts for X-formalism. Then, we define the rules for deriving the DAML+OIL representation of X-classes and clusters of them. Finally, we define the rules for deriving the DAML+OIL representation of the global X-classes and associated mapping rules.

4.1 Basic concepts definition

The basic ontology concept to be represented in DAML+OIL is the X-class, which can be local or global depending on the scheme layer of the ontology we are considering. An X-class is defined by a name and a structure. The X-class structure is defined in terms of names of properties and/or names of other referenced X-classes. We define a DAML+OIL class `X-class` in order to represent X-classes and two DAML+OIL subclasses of `X-class` named `LocalX-class` and `GlobalX-class` to represent X-classes in the semantic mapping scheme (named local X-classes in the following) and global X-classes in

```
<daml:Class rdf:ID=LocalX-class>
 <rdfs:subClassOf rdf:resource=#X-class/>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty
      rdf:resource=#hasProperty/>
   <daml:toClass rdf:resource=#Property/>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource=#hasLink/>
   <daml:toClass rdf:resource=#Link/>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty
      rdf:resource=#hasAttribute/>
   <daml:toClass rdf:resource=#Attribute/>
  </daml:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty
      rdf:resource=#hasRefX-class/>
   <daml:toClass rdf:resource=#X-class/>
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
```

Fig. 5. DAML+OIL definition of `LocalX-class`

the mediation scheme, respectively. Furthermore, we define a DAML+OIL class `Cluster` to represent the clusters of the semantic mapping scheme. We define DAML+OIL classes `Property`, `Link`, and `Attribute`, to represent properties, links, and attributes of X-classes. Moreover, we define DAML+OIL object properties `hasProperty`, `hasLink`, `hasAttribute`, and `hasRefX-class` to represent the relationships between X-classes and their properties, links, attributes, and referenced X-classes, respectively. Attribute, property and link types are represented through the datatype properties `AttributeType`, `PropertyType`, and `LinkType`, respectively. The `AttributeType` range is the union of the attribute types allowed in XML, the `PropertyType` range is the set {`PCDATA, any, empty`}, and the `LinkType` range is the union of the property types and the features of the X-class structure. Feature cardinalities are represented by the object property `card`, whose range is the class of the cardinalities used in X-formalism, expressed as a pair (mc,MC), where mc denotes the min cardinality and MC the max cardinality. As an example, the DAML+OIL definition of `LocalX-class` is reported in Figure 5. A summary of these basic DAML+OIL concepts is given in Table 3.

4.2 Rules for semantic mapping scheme representation

For each X-class xc of the semantic mapping scheme, an instance of the DAML+OIL class `LocalX-class` is defined whose structure is derived from attributes, properties, links, and referenced X-classes of xc applying the following

DAML+OIL concept	Description
Classes	
`X-class`	Represents the general structure of X-classes in the ontology schemes.
`LocalX-class`	Represents the structure of the X-classes describing the datasource concepts in the semantic mapping scheme.
`GlobalX-class`	Represents the structure of the global X-classes describing global concepts in the mediation scheme.
`Cluster`	Represents the structure of clusters in the semantic mapping scheme.
`Property`	Represents properties of X-classes.
`Link`	Represents links between X-classes.
`Attribute`	Represents attributes of X-classes.
Object properties	
`hasProperty`	Represents the relationship between an X-class and its properties.
`hasLink`	Represents the relationship between an X-class and its links.
`hasAttribute`	Represents the relationship between an X-class and its attributes.
`hasRefX-class`	Represents the relationship between an X-class and its referenced X-classes.
`card`	Represents the relationship between each feature X-class and its cardinality.
Datatype properties	
`AttributeType`	Represents the attribute types.
`PropertyType`	Represents the property types.
`LinkType`	Represents the link types.

Table 3. Description of the basic DAML+OIL concepts

X-formalism-to-DAML+OIL translation rules.

Attribute. An attribute $a = (n_a,\ t_a,\ k_a)$ is translated into an instance of the DAML+OIL class `Attribute`, named n_a. The instance type is the attribute type t_a. The attribute cardinality k_a is represented by means of the object property `card` in the DAML+OIL class instance defined for a.

Property. A property $p = (n_p,\ t_p,\ A_p)$ is translated into an instance n_p of the DAML+OIL basic class `Property`. The property type t_p is the value of the DAML+OIL property `PropertyType`. For each attribute $a_i \in A_p$ an object property clause `hasAttribute` is defined. The target of this property is the DAML+OIL instance representing a_i. The property cardinality specified in the X-class structure is represented using the object property `card` in the DAML+OIL class instance defined for p.

Link. A link $l = (n_l,\ t_l)$ is translated into an instance n_l of the DAML+OIL class `Link`. The link type t_l is the value of the DAML+OIL property `LinkType`.

Referenced X-class. A referenced X-class r is trans-

```
<LocalX-class rdf:ID=S2.Programme>
 <card>(0,n)</card>
 <hasAttribute rdf:resource=#S2.language/>
 <hasAttribute rdf:resource=#S2.genre/>
 <hasAttribute rdf:resource=#S2.subtitles/>
 <hasAttribute rdf:resource=#S2.repeat/>
 <hasAttribute
      rdf:resource=#S2.signlanguage/>
 <hasProperty rdf:resource=#S2.time/>
 <hasProperty rdf:resource=#S2.title/>
 <hasProperty rdf:resource=#S2.videoplus/>
 <hasRefX-class
        rdf:resource=#S2.Description/>
</LocalX-class>
```

Fig. 6. Example of DAML+OIL specification for the local X-class `S2.Programme`

lated into an instance n_r of the `LocalX-class`, where n_r is the name of r. The referenced X-class cardinality specified in the X-class structure is represented using the object property `card` in the DAML+OIL specification for r.

X-class. An X-class xc = (n_{xc}, s_{xc}, P_{xc}, L_{xc}, R_{xc}, A_{xc}) is translated into an instance n_{xc} of the DAML+OIL class `LocalX-class`. If the X-class xc is a referenced X-class, it is translated according to the referenced X-class translation rule. In order to represent the X-class properties, the DAML+OIL object property `hasProperty` is used. Each property $p_i \in P_{xc}$ which satisfies `hasProperty` has to belong to the class `Property`. According to the same process, the properties `hasLink` and `hasAttribute` are used for each link $l_i \in L_{xc}$, and for each attribute $a_i \in A_{xc}$. Finally, a property `hasRefX-class` is used for each referenced X-class $r_i \in R_{xc}$ [2]. As an example, the DAML+OIL specification of the referenced X-class `S2.Programme` is shown in Figure 6.

```
<Cluster rdf:ID=cl2>
   <AffinityValue>0.667</AffinityValue>
   <ClusterMember>
      <rdf:bag>
         <rdf:li>S1.Programslot</rdf:li>
         <rdf:li>S2.Programme</rdf:li>
      </rdf:bag>
   </ClusterMember>
</Cluster>
```

Fig. 7. DAML+OIL representation on the cluster cl_2

Cluster. A cluster is represented as an instance of the class `Cluster`. The instance name is the cluster name, and a property `AffinityValue` is defined in order to represent the affinity value associated with the cluster. Moreover, a property `ClusterMember` is defined in order to represent the X-classes composing the cluster. The `ClusterMember` domain is the class `Cluster`, while its range is the class `LocalX-class`. As an example, the DAML+OIL specification of the cluster cl_2 is shown in Figure 7.

[2] The structure s_{xc} of an X-class is not explicitly represented in DAML+OIL; however it can be easily derived by considering the set of DAML+OIL object properties of the `LocalX-class`

4.3 Rules for mediation scheme representation

A global X-class is defined as an X-class plus a set MP of mapping rules, which represent the mappings between the features of the global X-class and the features of the X-classes belonging to the cluster from which the global X-class has been derived. A global X-class is translated in DAML+OIL using the rules previously described for the local X-class translation. An additional rule is now introduced for translating mapping rules in DAML+OIL.

Mapping rule. A mapping rule defines for each property, link, attribute, and referenced X-class of a global X-class the corresponding feature (if any) of each local X-class of the involved cluster. We define a basic DAML+OIL object property `MappingRule` to represent a mapping rule. For each property, link, attribute, and referenced X-class of a global X-class, a `MappingRule` property is specified to represent the corresponding features in each source X-class. For each `MappingRule`, a `GlobalFeature` and a `SourceFeature` object properties are defined to represent the feature of the global X-class and the corresponding features in local X-classes, by referring to the corresponding DAML+OIL class. If a global feature does not have a corresponding feature in a local X-class of the cluster, (i.e., a `NULL` value is specified in the mapping table), such a local feature is not listed in the `SourceFeature` DAML+OIL object property.

Global X-class. A global X-class gxc is translated into an instance n_{gxc} of the DAML+OIL class `GlobalX-class` by applying the translation rules for X-class and mapping rules. For each `hasProperty`, `hasLink`, `hasAttribute`, and `hasRefX-class` property, a `MappingRule` property is specified which links the DAML+OIL class instance representing the global feature with the DAML+OIL class instance representing the corresponding source feature. In Figure 8, we show an example of DAML+OIL specification for the global X-class `Programme` of Figure 4, with the sample mapping rules of Table 2.

5 Related work

Work related to the issues discussed in the paper regards information integration. The goal of information integration is to construct a global description, called global schema, of a multitude of heterogeneous datasources, to provide the user with a uniform query interface against the sources, independent from their location and from heterogeneity of their data. In the literature, several approaches and tools for the integration of heterogeneous data sources have been developed, and information integration architectures based on mediator/middleware data models and ontologies have been proposed [12,13,14,15]. More recently, approaches and systems have been proposed specifically devoted to semantic representation and integration of XML datasources. In the follow-

```
<GlobalX-class rdf:ID=Programme>
 <hasAttribute rdf:resource=#language/>
 <MappingRule>
  <GlobalFeature rdf:resource=#language/>
  <SourceFeature>
   <rdf:Bag>
     <rdf:li rdf:resource=#S1.language/>
     <rdf:li rdf:resource=#S2.language/>
   </rdf:Bag>
  </SourceFeature>
 </MappingRule>
 ...
 <hasProperty rdf:resource=#TIME/>
 <MappingRule>
  <GlobalFeature rdf:resource=#TIME/>
  <SourceFeature>
   <rdf:Bag>
     <rdf:li rdf:resource=#S1.TIME/>
     <rdf:li rdf:resource=#S2.TIME/>
   </rdf:Bag>
  </SourceFeature>
 </MappingRule>
 <hasProperty rdf:resource=#GENRE/>
 <MappingRule>
  <GlobalFeature rdf:resource=#GENRE/>
  <SourceFeature rdf:resource=#S2.genre/>
 </MappingRule>
 ...
 <hasRefX-class rdf:resource=#Description/>
 <MappingRule>
  <GlobalFeature rdf:resource=#Description/>
  <SourceFeature>
   <rdf:Bag>
     <rdf:li>S1.DESCRIPTION</rdf:li>
     <rdf:li>S2.DESCRIPTION</rdf:li>
   </rdf:Bag>
  </SourceFeature>
 </MappingRule>
 ...
</GlobalX-class>
```

Fig. 8. Example of DAML+OIL specification for the global X-class `Programme` of Figure 4

ing, we consider such works by focusing on schema matching and integration techniques and on metadata representation issues, which are more strictly related with our work described in the paper.

Cupid [1] is a schema matching system providing different matching techniques for discovering mappings between elements of heterogeneous schema descriptions. Metadata representation is through through graphs, using an extended Entity-Relationship model. Cupid computes similarity coefficients among elements of different schemas, using a combination of matching techniques. In a linguistic matching stage, a linguistic similarity coefficient is computed between pairs of elements, by considering their name, type and domain. In a structural matching stage, the similarity of the elements context is also considered. Finally, the similarity between each pair of elements is computed as the average of the two similarity coefficients obtained in the two phases. In this system, the focus is mainly on schema matching techniques, rather than to datasource integration. Our schema matching techniques for XML data, can be performed by taking into account both linguistic and structure information on X-classes by means of X-formalism and of the thesaurus of terminological relationships.

In the Xyleme project [2], automated data integration techniques are developed for creating a dynamic XML warehouse for the Web. In particular, a DTD classification into domains is performed based on a statistical analysis of the similarities between labels in the considered DTDs. These similarities are defined using ontologies or thesauri. Then, semantic mappings between elements in different DTDs are identified. Metadata representation at the global level is provided by the so called *abstract DTDs*. An abstract DTD is a unified representation of the DTDs associated with a set of XML documents in a given domain, and is modeled as a tree. The proposed solution for defining path mappings uses standard natural language techniques and a thesaurus, and relies on human interaction especially for identifying mappings between DTD paths. Machine-learning techniques for XML datasource integration are described [4], for matching source DTDs against a pre-defined global schema in form of DTD. The system ask the user to provide semantic mappings for a small set of datasources. A set of instance matchers, called *learners*, are trained on such mappings during a preprocessing step, in order to discover instance patterns and matching rules to be applied for matching new DTDs against the global schema. Also in this system, metadata representation is provided by XML schema trees. This system follows a "local-as-view" approach to integration, in that the global schema is given and local source schemas are defined as views over the global schema. In our approach, we follow a "global-as-view" approach to integration, and we construct the global schema (i.e., the mediation scheme of our ontology) following a bottom-up process. The mediation scheme is then mapped on local schemas, by means of mapping rules.

In [16], a methodological framework for information integration, based on a knowledge representation approach is presented. The representation of both the sources and the global domain is provided by Description Logics. The integration is performed according to a local-as-view approach, and, as such, it is focused on representation of mappings between source schemes and the global schema. The proposed approach emphazises the role of Description Logics for knowledge representation and reasoning in information integration. The DAML+OIL representation described in this paper can play an analogous role.

Main contributions of our work with respect to these proposals regard: i) the organization of the ontology integration knowledge into a semantic mapping scheme layer and a mediation scheme layer; ii) the use of X-formalism for representing source DTDs, which allows to conceptually describe DTDs contents and their role following a data model approach (i.e., by distinguishing concepts like X-class, property, link); iii) use of DAML+OIL for ontology knowledge representation, using a language ensuring compatibility and interoperability with Semantic Web.

6 Concluding remarks

In this paper, we have described ontology-based integration of heterogeneous XML data sources, where ontology knowledge is organized into a semantic mapping scheme and a mediation scheme. We have described the ontology design techniques, which are schema matching techniques, for deriving clusters of semantically related datasource concepts in the semantic mapping scheme, and unification techniques, for deriving global concepts and links in the mediation scheme. Such techniques rely on the X-formalism conceptual representation of source DTDs for mapping DTD contents to constructs of X-class, properties, attributes, links, and referenced X-class and on a thesaurus of terminological relationships specifying interschema knowledge. We have described ontology knowledge representation with DAML+OIL to export the ontology knowledge to Semantic Web in a standardized way.

Future research work will proceed in the following directions. A CORBA module for producing DAML+OIL specification of the ontology is being developed in the framework of the ARTEMIS tool environment. Actually, ARTEMIS implements the ontology using ODL_{I^3}, the ODL-based object-oriented integration language developed in MOMIS [7]. In particular, ODL_{I^3} describes heterogeneous schemas of structured and semistructured data sources in a common way and represents the mediation scheme and associated mapping rules obtained from the integration process by means of the basic construct of ODL_{I^3} class with attributes of predefined or complex type. We are studying how to derive the DAML+OIL specification of the ontology knowledge by considering directly ODL_{I^3}. In this respect, the results presented in this paper will constitute the starting base for designing the rules for the ODL_{I^3}-to-DAML+OIL representation. Moreover, the DAML+OIL knowledge representation can be the basis to export metadata knowledge to a metadata repository. We will address this problem in the context of the D2I (Integration, Warehousing, and Mining of Heterogeneous Data Sources) project, where a metadata repository has been defined, providing a centralized and unified representation of the metadata documenting all activities related to integration applications. Finally, we will study how our matching and unification techniques can be extended to the problem of matching and integrating ontologies, by considering DAML+OIL specifications as an input to the ontology design process.

References

1. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with cupid. In: Proc. of the Int. Conf. on Very Large Data Bases (VLDB'01), Rome, India (2001) 49–58
2. Reynaud, C., Sirot, J., Vodislav, D.: Semantic integration of XML heterogeneous data sources. In: Proc. of the International Database Engineering and Applications Symposium (IDEAS'01), Grenoble, France (2001) 199–208
3. Castano, S., Antonellis, V.D., di Vimercati, S.D.C.: An XML-based interorganizational knowledge mediation scheme to support B2B solutions. In: 9th IFIP 2.6 Working Conference on Database Sematics (DS-9), Hong Kong, China (2001)
4. Doan, A., Domingos, P., Halevy, A.: Reconciling schemas of disparate data sources: A machine-learning approach. In: Proc. of ACM SIGMOD 2001, Santa Barbara, California, USA (2001) 509–520
5. Castano, S., Antonellis, V.D., Ferrara, A., Ottathycal, G.K.: A disciplined approach for the integration of heterogeneous XML datasources. In: Proc. of DEXA WEBS Workshop, Aix-en-Provence, France (2002) To appear.
6. The ARTEMIS project http://www.ing.unibs.it/ ∼deantone/ interdata_tema3/Artemis/artemis.html.
7. Bergamaschi, S., Castano, S., Vincini, M., neventa no, D.B.: Semantic integration of heterogeneous information sources. Data and Knowledge Engineering **36** (2001)
8. Castano, S., Antonellis, V.D., di Vimercati, S.D.C.: Global viewing of heterogeneous data sources. IEEE Transactions on Knowledge and Data Engineering **13** (2001)
9. Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L.: DAML+OIL (March 2001) Reference Description http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.
10. Horrocks, I.: DAML+OIL: a reason-able web ontology language. In: Proc. of EDBT 2002. (2002)
11. Castano, S., Antonellis, V.D., di Vimercati, S.D.C., Melchiori, M.: Semi-automated extraction of ontological knowledge from XML datasources. In: Proc. of DEXA WEBH Workshop, Aix-en-Provence, France (2002) To appear.
12. Fernandez, M., Florescu, D., Kang, J., Levy, A., Suciu, D.: Catching the boat with strudel: Experiences with a web-site management system. In: Proc. of ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA (1998) 414–425
13. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., et al., J.U.: The tsimmis approach to mediation: Data models and languages. Journal of Intelligent Information Systems **2** (1997) 117–132
14. Haas, L., Miller, R., Niswonger, B., Roth, M., Schwarz, P., Wimmers, E.: Transforming heterogeneous data with database middleware: Beyond integration. IEEE Data Engineering Bulletin **1** (1999) 31–36
15. Levy, A., Rajaraman, A., Ordille, J.: Querying heterogeneous information sources using source descriptions. In: Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96), Mumbay (Bombay), India (1996) 251–262
16. Calvanese, D., Giacomo, G.D., Lenzerini, M., Nardi, D., Rosati, R.: Knowledge representation approach to information integration. In: Proc. of AAAI Workshop on AI and Information Integration, AAAI Press/The MIT Press (1998) 58–65

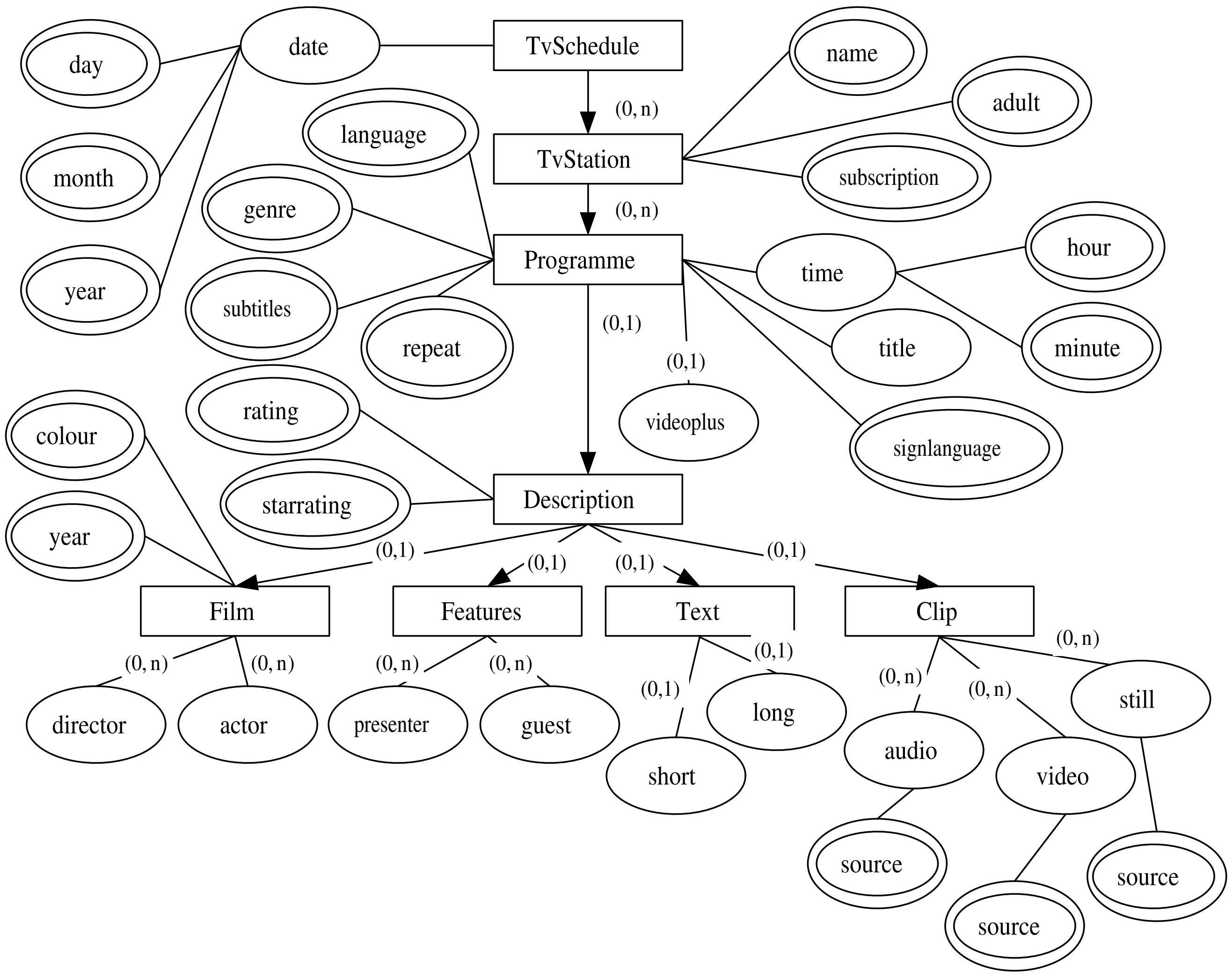


Fig. 3. X-formalism representation of the DTD in Fig. 1 (b)

MAFRA — A MApping FRAmework for Distributed Ontologies in the Semantic Web

Alexander Maedche[1], Boris Motik[1], Nuno Silva[1 2], and Raphael Volz[1]

[1] FZI Research Center for Information Technologies at the University of Karlsruhe, D-76131 Karlsruhe, Germany
{maedche,motik,silva,volz}@fzi.de
[2] ISEP Instituto Superior de Engenharia, Instituto Politecnico do Porto, Portugal

Abstract. Ontologies as means for conceptualizing and structuring domain knowledge within a community of interest are seen as a key to realize the Semantic Web vision. However, the decentralized nature of the Web makes achieving this consensus across communities difficult, thus, hampering efficient knowledge sharing between them. In order to balance the autonomy of each community with the need for interoperability, mapping mechanisms between distributed ontologies in the Semantic Web are required. In this paper we present MAFRA, an interactive, incremental and dynamic framework for mapping distributed ontologies in the Semantic Web.

1 Introduction

The current WWW is a grea1t success with respect to the amount of stored documents and the number of users. However, the ever-increasing amount information on the Web places a heavy burden of accessing, extracting, interpreting and maintaining information on the human users of Web. Tim Berners-Lee, the inventor of the WWW, coined the vision of Semantic Web, providing means for annotation of Web resources with machine-processable metadata providing them with background knowledge and meaning (see [2]). Ontologies as means for conceptualizing and structuring domain knowledge are seen as the key to enabling the fulfillment of the Semantic Web vision.

However, the de-centralized nature of the Web makes indeed inevitable that communities will use their own ontologies to describe their data. In this vision, ontologies are themselves distributed and the key point is the mediation between distributed data using mappings between ontologies. Thus, complex mappings and reasoning about those mappings are necessary for comparing and combining ontologies, and for integrating data described using different ontologies [15]. Existing information integration systems and approaches (e.g., TSIMMIS [6], Information Manifold [8], Infomaster[3], MOMIS[4], Xyleme [5]) are “centralized” systems of mediation between users and distributed data sources, which exploit mappings between a single mediated schema and schemas of data sources. Those mappings are typically modeled as views (over the mediated schema in the local-as-view approach, or over the sources schemas in the global-as-view approach) which are expressed using languages having a formal semantics. For scaling up to the Web, the “centralized” approach of mediation is probably not flexible enough, and distributed systems of mediation are more appropriate.

[3] http://infomaster.stanford.edu/infomaster-info.html
[4] http://sparc20.ing.unimo.it/Momis/
[5] http://www.xyleme.com

Building on this idea and on existing work, we introduce in this paper MAFRA, an Ontology MApping FRAmework (MAFRA) for distributed ontologies in the Semantic Web. Within MAFRA we provide an approach and conceptual framework that provides a generic view onto the overall distributed mapping process. The distributed nature of Semantic Web entails significant degrees of information redundancy, incoherence and constant evolution, thus changing the nature of the ontology mapping problem: instead of creating a static specification document relating entities in two ontologies, a continuous, incremental, interactive and highly dynamic process supporting mapping evolution is required to scale up to the ever-changing nature of ontologies being mapped. Establishing a mapping between two ontologies is an engineering process of consensus building between two communities already agreeing on common ontologies for their own respective domains. This task implies negotiation, so attention is paid to providing means for cooperative mapping. Thus, proposed framework offers support in all parts of the ontology mapping life-cycle.

Organization of this paper. In section 2 we motivate our work by introducing an application for whose success a solution to the distributed ontology mapping problem is required. Based on this application, we have collected the requirements for developing MAFRA. In section 3 we introduce the underlying conceptual architecture of MAFRA. In section 4 we focus on mapping representation and present the current status of our semantic bridging ontology and discuss its features. Section 5 presents the realized mapping implementation within KAON - an ontology and Semantic Web application framework[6]. Before we conclude a short discussion of related and future work is given in section 6.

2 MAFRA – Application Scenarios

Design of MAFRA recognizes specific requirements of several concrete application scenarios. In this section we present one of these scenarios and discuss its respective requirements.

“People can’t share knowledge if they do not speak a common language”. This simple insight accurately characterizes what makes knowledge management a challenging task. Its goal to reach global knowledge access within different departments of an enterprise is usually difficult due to the fact that different departments usually encompass different vocabularies, which hinders communication. Large companies

[6] http://kaon.semanticweb.org

typically consist of departments such as Human Resources, Production, Sales, Marketing and Finance. By using ontologies, the task of collecting, organizing, and distributing the knowledge within one department may be solved – ontologies provide a sound semantic basis for the definition of meaning that can be understood by both humans and machines. Also, a single department is typically small enough so that achieving consensus among interested parties is feasible. However, designing a large-scale ontology covering the needs of all departments has shown to be a very difficult task due to effort, scale and maintainability. Interoperability between departments can then be achieved by mapping of ontologies of each department. It is anticipated that mapping existing ontologies will be easier than creating common ontology because a smaller community is involved in the process. It is important to emphasize that we do not consider a closed world and centralized information integration system as a possible solution for the problem introduced above.

Ontologging[7] is an ontology-based environment tackling this problem. It builds on Semantic Web standards with the goal of enabling next generation knowledge management applications allowing management and usage of multiple ontologies. An important requirement within the development of the Ontologging multi-ontology system is that there exists extensive tool support for supporting the overall mapping process. A specific requirement was the support of automatic detection of similarities of entities contained in the two different department ontologies.

3 Conceptual Framework

An ontology mapping process, as defined in [14], is the set of activities required to transform instances of a source ontology into instances of a target ontology. By studying the process and analyzing different approaches from the literature [14] we observed a set of commonalities and assembled them into the MAFRA conceptual framework, outlined in Figure 1. The framework consists of five horizontal modules describing the phases that we consider fundamental and distinct in a mapping process. Four vertical components run along the entire mapping process, interacting with horizontal modules.

3.1 Horizontal Dimension of MAFRA

Within the horizontal dimension, we identified following five modules:

Lift & Normalization. This module focuses on raising all data to be mapped onto the same representation level, coping with syntactical, structural and language heterogeneity [16]. Both ontologies must be normalized to a uniform representation, in our case RDF(S), thus eliminating syntax differences and making semantics differences between the source and the target ontology more apparent [14]. To facilitate that, we developed a LIFT tool providing means to bring DTDs, XML-Schema, and relational databases to the structural level of the ontology. Lift is not further elaborated in this paper - we shall simply assume that the source and target ontologies are already represented in RDF-Schema with their instances in RDF.

[7] http://www.ontologging.com

Similarity. This module establishes similarities between entities from the source and target ontology. Similarity between conceptual models is hard to measure and often establishing a suitable similarity measure is a very subjective task. Several different similarity measures have been proposed in literature [14,3,5,10,1], focusing on different aspects of ontology entities. We don't further elaborate on this issue, as it is not in scope of this paper.

Semantic Bridging. Based on the similarities computed in the previously described phase, the semantic bridging module is responsible for establishing correspondence between entities from the source and target ontology. Technically, this is accomplished by establishing semantic bridges - entities reflecting correspondence between two ontology entities. Apart from the semantic correspondence, additional "procedural" information is needed to further specify the transformation to be performed, e.g. translation of measures like currencies. Semantic bridging is further discussed in section 4.

Execution. This module actually transforms instances from the source ontology into target ontology by evaluating the semantic bridges defined earlier. In general two distinct modes of operation are possible, namely offline (static, one-time transformation) and online (dynamic, continuous mapping between source and the target) execution. Execution issues further discussed in section 5.

Post-processing. The post-processing module takes the results of the execution module to check and improve the quality of the transformation results. The most challenging task of post-processing is establishing object identity - recognizing that two instances represent the same real-world object [7]. Furthermore, by computing statistical properties of transformed instances, it is possible to check whether semantic bridges were underspecified.

3.2 Vertical Dimension of MAFRA

The vertical dimension of MAFRA contains modules that interact with horizontal modules during the overall mapping process. Following four modules have been identified and will be only shortly mentioned in this paper:

Evolution. This modules focuses on keeping semantic bridges obtained by the "Semantic Bridge" module, which must be kept in synchrony with the changes in the source and target ontologies. Evolving ontologies on the Semantic Web result in an update requirement of the corresponding semantic bridges. Although this may be achieved by reapplying the mapping process, this is probably not the most efficient or accurate way. Thus, the mapping process must have an evolution component that will reuse the existing semantic bridges in adapting them to new requirements.

Cooperative Consensus Building. The cooperative Consensus Building module is responsible for establishing a consensus on semantic bridges between two communities participating in the mapping process. This is a requirement as one has to choose frequently from multiple, alternatively possible mappings .The amount of human involvement required to achieve consensus may be reduced by automating the mapping process as much as possible.

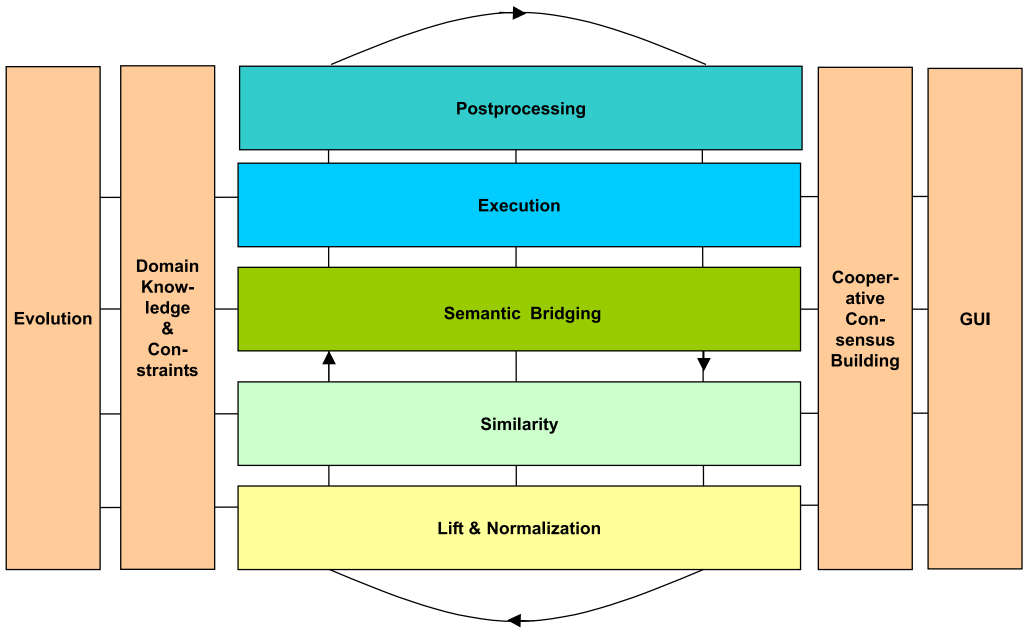


Fig. 1. Conceptual Architecture

Domain Constraints and Background Knowledge. The quality of similarity computation and semantic bridging may be dramatically improved by introducing background knowledge and domain constraints, e.g. by using glossaries to help identify synonyms or by using lexical ontologies, such as WordNet or domain-specific thesauri, to identify similar concepts.

Graphical User Interface. Mapping is a difficult and time consuming process, which is not less difficult than building an ontology itself, i.e. deep understanding of both conceptualizations required on human side, thus extensive graphical support must be given and it is a separate issue how this can be achieved in an optimal way. The graphical user interfaces (GUI) modules allows the users drive the mapping process, provide domain constraint and background knowledge, create semantic bridges, refine bridges according to the results of the execution module, etc. Some aspects of the GUI are further elaborated in section 5.

4 Semantic Bridging

As mentioned in subsection 3.1, the role of the semantic bridging component is to semantically relate entities from the source and target ontologies. This is achieved by creating so-called semantic bridges. A role of a semantic bridge is to encapsulate all necessary information to transform instances of one source ontology entity to instances of one target ontology entity. In the rest of this section we first explore the nature of semantic bridges by analyzing their different dimensions determining each bridge. Next we discuss our approach of using a meta-ontology to enable the specification of semantic bridges. At last we give an example of how semantic bridges can be defined between two domain ontologies.

4.1 Dimensions of Semantic Bridges

The nature of semantic bridges may be understood by considering different dimensions, each describing one particular aspect of a semantic bridge. By analyzing ontologies used on the Semantic Web, we identified following five dimensions of semantic bridges:

1. **Entity dimension** reflects the type of ontology entities being bridged,
2. **Cardinality dimension** reflects the number of ontology entities being bridged,
3. **Structural dimension** reflects the way how elementary bridges may be combined into more complex bridges,
4. **Constraint dimension** reflects constraints applied during the execution phase to instances from the source ontology,
5. **Transformation dimension** reflects how instances of the source ontology are transformed during the mapping process.

Entity dimension. Semantic bridges may relate the ontology entities *(i)* concepts (modeling classes of objects from the real world), *(ii)* relations (modeling relationships between objects in the real world), and, *(iii)* attributes (modeling simple properties of objects in the real world) and *(iv)* extensional patterns (modeling the content of the instances).

Cardinality dimension. This dimension determines the number of ontology entities at both sides of the semantic bridge, ranging from $1 : 1$ to $m : n$. However, we have found that in most cases $m : n$ is not a common requirement, so $1 : n$ and $m : 1$ suffice. Even when $m : n$ are encountered, often they may be decomposed into m $1 : n$ bridges.

Structural dimension. This dimension reflects the way how elementary bridges may be combined into more complex bridges. We distinguish between the following different relations that may hold between bridges:

– **Specialization** allows a bridge to reuse definitions from another bridge and provide additional information (e.g. a bridge relating Employee concepts from two ontologies may be a specialization of a more general bridge relating Person concepts),

– **Abstraction** is a variation of the type of the superclasses. When this attribute is set, the specified bridge should not be executed independently, but only as superclass of another.
– **Composition** relation between to bridges specifies that a bridge is composed of other bridges,
– **Alternatives** relation between bridges specifies a set of mutually exclusive bridges.

Constraint dimension. The constraint dimension permits to control the execution of a semantic bridge. It reflects relevant constraints applied during the execution phase to instances from the source ontology. Constraints act as conditions that must hold in order the transformation procedures is applied onto the instances of the source ontology, e.g. the bridge evaluate only if the value of the source instance matches a certain pattern.

Transformation dimension. This dimension reflects how instances of the source ontology are transformed during the mapping process. Transformations assume different complexity and variety depending on the ontologies being bridged.

4.2 Semantic Bridging Ontology (SBO)

Within our approach four different types of relations between entities, a particular semantic bridge exists. A specification of all available semantic bridges, organized in a taxonomy, is a semantic bridging ontology (SBO). To actually relate the source and target ontology, the mapping process creates an instance of SBO containing semantic bridge instances, each encapsulating all necessary information to transform instances of one source entity to instances of the target entity. In the following sections we will describe the semantic bridging ontology in more detail.

Figure 2 describes the most important entities of the semantic bridging ontology. We refer to the five, previously described semantic bridge dimensions:

– Three basic types of entities are considered: Concepts, Relations and Attributes,
– The class SEMANTIC BRIDGE is the most generic bridge, it defines the relations to source and target entities. It is specialized according to the entity type and according to cardinality. Though, there are many combinations of entity types and cardinality bridges that are not explicitly specified, it is important to mention that they can be easily specialized from more general bridges.
– The class SERVICE represents a class used to reference resources that are responsible to connect to, or describe transformations. This class is intended to be used to describe these transformations resources. Because services are normally external to the execution engine, it is required to describe some fundamental characteristics like name, interface (number and type of arguments) and location. Argument and its sub classes Arg and ArgArray permits to describes these characteristics in a simple and direct form.
– RULE is the general class for constraints and transformation-relevant information, which provides a relation to the service class.
– The class TRANSFORMATION is mandatory in each semantic bridge except if the semantic bridge is set as abstract. It uses the inService relation to link to the transformation procedure, and any execution engine and function specific attributes in order to specify extra requirements;
– The class CONDITION represents the conditions that should be verified in order to execute the semantic bridge. Condition is operationally similar to transformation in the sense that it must specify all the extra requirements for the function that test the conditions. Because any semantic bridge may have a condition, it allows to control complex transformations according to both the schema and instances data, specially in combination with SemanticBridgeAlt and the Composition constructs.
– The COMPOSITION modelling primitive identified above is supported by the hasBridge relation in the SEMANTICBRIDGE class. It has no cardinality limit nor type constraint which allows any semantic bridge to aggregate many different bridges. Those semantic bridges are then called one by one, and processed in the context of the former.
– The ALTERNATIVE modelling primitive is supported by the SemanticBridgeAlt class. It groups several mutual exclusive semantic bridges. The execution parser checks each of the bridges condition rules and the first bridge which conditions hold is executed while the others are discarded.

In the following, we will describe how the semantic bridging ontology has been represented so it may be used within Semantic Web applications.

SBO represented in DAML+OIL. DAML+OIL [8] has been choosen to represent the semantic bridge ontology. DAML+OIL builds on and extends RDF-Schema and provides a formal semantics for it. One of the goals in specifying the semantic bridge ontology was to maintain and exploit the existent constructs and minimize extra constructs, which would maximize as much as possible the acceptance and understanding by general Semantic Web tools. The SBO ontology is available online at http://kaon.semanticweb.org/2002/04/SBO.daml.

4.3 Example

Let us consider Figure 3 where a small part of two different ontologies are represented. The ontology on the left side (o1) describes the structure of royal families and associated individuals. These concepts are combined with events, both individual events (birth date and death date) and families events (marriages and divorces). The ontology on the right side (o2), characterizes individuals using a very simple approach. It is mainly restricted in representing if the individual is either a Man or a Woman. Unlike o1 that extensively enumerates marriages and divorces, o2 is concerned just with the number of marriages. The goal of this example is to specify a mapping between the source and target ontology (o1 and o2 respectively), using the developed semantic bridge ontology

[8] http://www.daml.org/2001/03/daml+oil-index.html

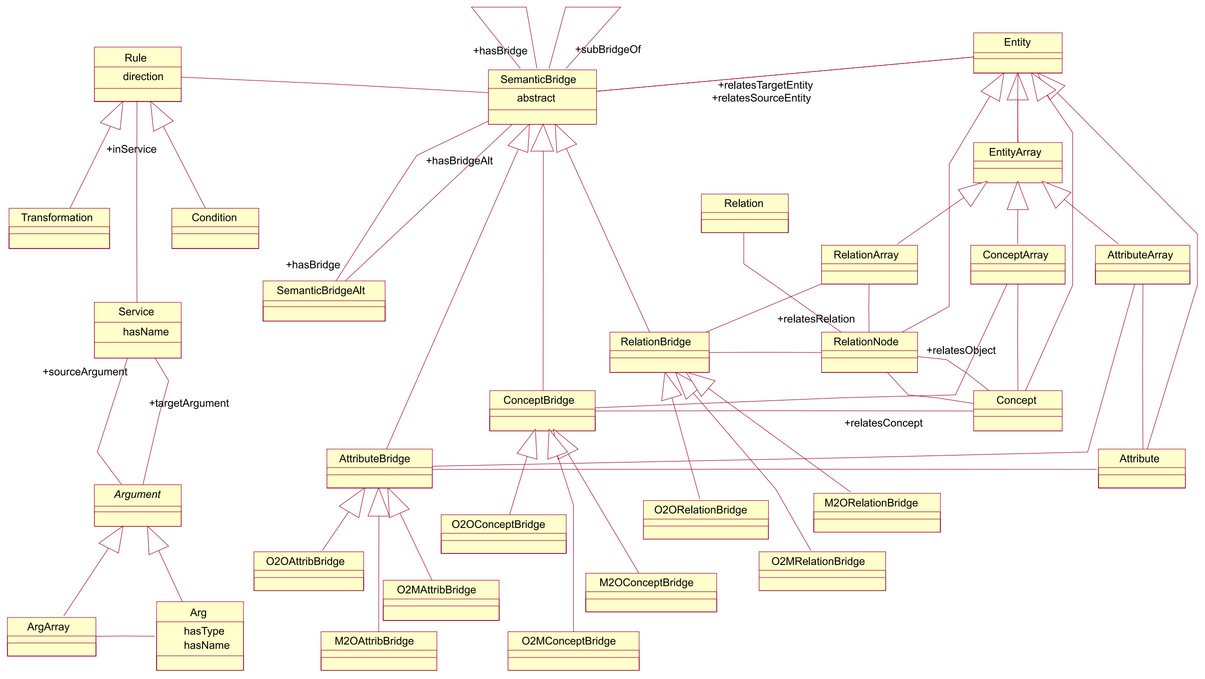


Fig. 2. Bridging Ontology view in UML

(SBO). In order to exploit the SBO potentialities, the mapping specification follows the structure of the ontologies being mapped, normally in the form of a taxonomy. Therefore, a mapping structure represented according to SBO tends to arrange bridges in a hierarchical way.

First, the mapping must define the two ontologies being mapped. Additionally, one may specify top-level semantic bridges which serve as entry points for the translation, even if there are not mandatory. In this case the translation engine starts executing the ”Individual-Individual” bridge.

```
<Mapping rdf:ID="mapping">
    <relatesSourceOntology rdf:resource="&o1;"/>
    <relatesTargetOntology rdf:resource="&o2;"/>
    <hasBridge rdf:resource="#Individual-Individual"/>
</Mapping>
```

Notice that the target ontology intends to create instances of either ”o2:Woman” or ”o2:Man”, but not ”o2:Individual”. In object oriented terminology ”o2:Individual” class is said to be abstract. It is therefore required to state that this concept bridge should not be used to create instances, but serve just as support to sub bridges, like it happens in object oriented paradigm. SBO uses the abstract property in these circumstances. If no abstract property is specified or if it is set to FALSE, then the concept bridge is considered as non-abstract.

It is now necessary to set the alternative between ”o1:Individual” and either ”o2:Woman” or ”o2:Man”. This situation is specified by a SemanticBridgeAlt. In this case the alternatives are two ConceptBridge’s: ”Individual-Woman” and ”Individual-Man”. Bridges may be numerically ordered which can useful if the last bridge has no specified condition. Both rdf:_n like syntax and the one presented are allowed to specify the order.

```
<SemanticBridgeAlt rdf:ID="ManOrWoman">
    <hasBridge>
        <Seq ordinal="1">
            <bridge rdf:resource="#Individual-Woman"/>
        </Seq>
    </hasBridge>
    <hasBridge>
        <Seq ordinal="2">
            <bridge rdf:resource="#Individual-Man"/>
        </Seq>
    </hasBridge>
</SemanticBridgeAlt>
```

The alternative ConceptBridge’s are presented next: ”Individual-Woman” and ”Individual-Man”.

```
<ConceptBridge rdf:ID="Individual-Woman">
    <subBridgeOf rdf:resource="#Individual-Individual"/>
    <relatesSourceEntity rdf:resource="#Individual"/>
    <relatesTargetEntity rdf:resource="#Woman"/>
    <whenVerifiedCondition rdf:resource="#isFemale"/>
</ConceptBridge>

<ConceptBridge rdf:ID="Individual-Man">
    <subBridgeOf rdf:resource="#Individual-Individual"/>
    <relatesSourceEntity rdf:resource="#Individual"/>
    <relatesTargetEntity rdf:resource="#Man"/>
</ConceptBridge>
```

Both bridges rely on the ”Individual-Individual” bridge to translate ”o2:Man” and ”o2:Woman” inherited attributes from ”o2:Individual”. Hence, both are specified as sub-bridges of ”Individual-Individual” concept bridge. Additionally, ”Individual-Woman” concept bridge specifies the whenVerifiedCondition property to ”isFemale”. As remarked bellow, this condition is responsible to test if the individual is of feminine sex. If the condition is verified the bridge is executed. Otherwise, and because the condition is tested in the context of a SemanticBridgeAlt, the next concept bridge in the alternative is processed. The next concept bridge in the alternative is ”Individual-Man” which has no associated condition, and therefore it is unconditionally executed.

Respecting the translation process, consider that an ”o1:Individual” instance is to be translated. The translation engine seeks for bridges relating ”o1:Individual” to any o2 entity. Three are found, but one of them is abstract and is therefore rejected. The other two are both defined in the context of a SemanticBridgeAlt. The SemanticBridgeAlt choosing/exclusion process starts. One of the bridges (or eventually none if none of the associated conditions is verified) is selected. The concept bridge must then create a tar-

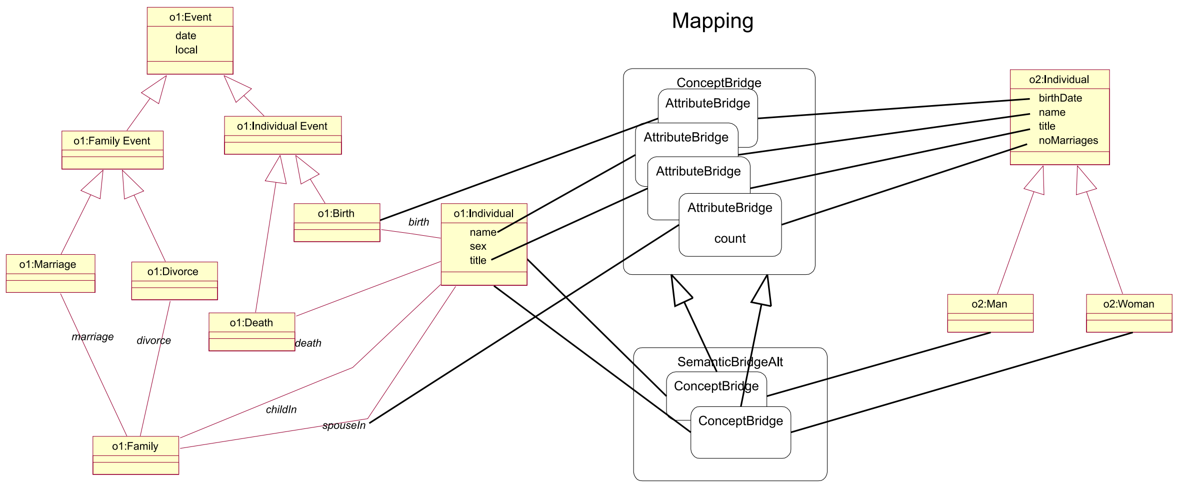


Fig. 3. UML representation of two small ontologies

get instance which will serve as context for complementary bridges.

Complementary attribute bridges are in this example simple 1:1 attribute bridges, relating one attribute from o1 to an attribute in o2, through the associated transformation.

```
<AttributeBridge rdf:ID="name-name">
    <relatesSourceEntity rdf:resource="#name"/>
    <relatesTargetEntity rdf:resource="#name"/>
    <accordingToTransformation rdf:resource="#copyName"/>
</AttributeBridge>

<Transformation rdf:ID="copyName">
    <mapSourceArgument>
        <MapArg>
            <from rdf:resource="#name"/>
            <to>sourceString</to>
        </MapArg>
    </mapSourceArgument>
    <mapTargetArgument>
        <MapArg>
            <from>targetString</from>
            <to rdf:resource="#name"/>
        </MapArg>
    </mapTargetArgument>
    <inService>CopyString</inService>
</Transformation>
```

The ”name-name” attribute bridge for example, bridges ”o2:Individual.name” to ”o2:Individual.name”. The associated transformation in this bridge has the responsibility to copy/create the attribute and assign it to the concept instance. Remember that the concept instance has been created by the concept bridge previously.

Concerning the transformation, it intends to map between the bridge entities and the transformation service arguments. This mapping specification varies according to the service be requested, either in type, cardinality and used tags. For example, the ”copyName” transformation specifies the ”CopyString” service to be called. This service expects to receive a source argument called ”sourceString” and the output is named ”targetString”. The transformation maps ”sourceString” with the attribute ”o1:Individual.name” and ”targetString” to the ”o2:Individual.name”. ”title-title” attribute bridge is very similar to the previous and is not be presented.

In contrast, ”marriages” attribute bridges for example, are slightly different from previous ones. Notice that the source entity is not an attribute but a relation to another concept. Normally an AttributeBridge would not be correctly applied. However, since this is a very common mapping pattern the translation engine allows to process the relation as an attribute. That could eventually be a problem if the translation service expects an attribute. However, the ”CountRelations” service expects a relation which is the case of ”spouseIn” and therefore no problem occurs. A similar situation occurs with ”birth-birth” AttributeBridge. Once again there is no problem because the source entity is accepted as an attribute and the rest is up to the transformation and its associated service.

```
<AttributeBridge rdf:ID="mariages">
    <relatesSourceEntity rdf:resource="#spouseIn"/>
    <relatesTargetEntity rdf:resource="#noMariages"/>
    <accordingToTransformation rdf:resource="#countSpouses"/>
</AttributeBridge>

<Transformation rdf:ID="countSpouses"> <putServiceArgument>
        <MapArg>
            <from>relation</from>
            <to rdf:resource="#spouseIn"/>
        </MapArg>
    </putServiceArgument>
    <mapTargetArgument>
        <MapArg>
            <from>count</from>
            <to rdf:resource="#noMariages"/>
        </MapArg>
    </mapTargetArgument>
    <inService>CountRelations</inService>
</Transformation>

<AttributeBridge rdf:ID="birth-birthDate">
    <relatesSourceEntity rdf:resource="#birth"/>
    <relatesTargetEntity rdf:resource="#birthDate"/>
    <accordingToTransformation rdf:resource="#Birth"/>
</AttributeBridge>


<Transformation rdf:ID="Birth">
    <putServiceArgument>
        <MapArg>
            <from>1</from>
            <to rdf:resource="#birth"/>
        </MapArg>
    </putServiceArgument>
    <putServiceArgument>
        <MapArg>
            <from>2</from>
            <to rdf:resource="#date"/>
        </MapArg>
    </putServiceArgument>
    <mapTargetArgument>
        <MapArg>
            <from>targetString</from>
            <to rdf:resource="#birthDate"/>
        </MapArg>
    </mapTargetArgument>
    <inService>RoyalDate</inService>
</Transformation>
```

Finally, the ”isFemale” condition is considered. This condition is responsible to verify if an instance of an individual is of feminine sex. In this case the pattern refers to the fact that the value of sex attribute has value ”F”. Normally, the services applied in a condition return a boolean value. However, this constraint would depend on the translation engine once it is possible to create a table of correspondences between boolean types and other types. For example, it would be reasonable to consider a true result if the service returns a set of entities or false if it return a empty set.

```
<Condition rdf:ID="isFemale">
```

```
    <putServiceArgument>
        <MapArg>
            <from>1</from>
            <to rdf:resource="#sex"/>
        </MapArg>
    </putServiceArgument>
    <putServiceArgument>
        <MapArg>
            <from>pattern</from>
            <to>F</to>
        </MapArg>
    </putServiceArgument>
    <inService>CascadeAndMatch</inService>
</Condition>
```

5 Implementation

MAFRA is currently under development within the KAON Ontology and Semantic Web Framework[9]. KAON offers a framework and a common set of tools for realizing scalable and reliable ontology-enabled Semantic Web applications. The architecture underlying KAON is depicted in Figure 4, with elements split into three layers as described next.

– The **Application and Services Layer** contains components providing interface to KAON. Human agents typically use one of user interface applications realized within OntoMat - a UI framework for ontology applications offering easy integration and interoperability of different tools for managing ontologies and metadata. KAON-Portal is a framework for the generation of Web portals from ontology-based data. Interoperability with non-Java platforms is realized through a Web-service interface. Furthermore, machine agents use the Web Service interface to access KAON methods and functionality.

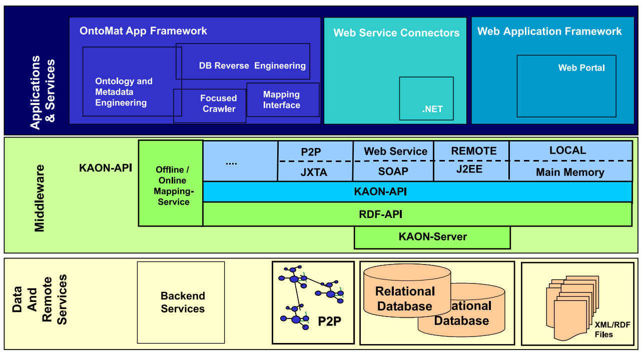
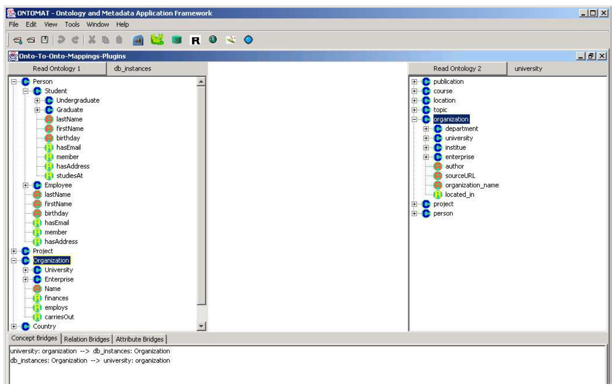


Fig. 4. KAON Architecture

– The **Middleware Layer** is focused around KAON API, providing primitives for ontology access and manipulation via different means. The middleware layer also includes a remote, J2EE-based implementation of KAON-API which allows to work multiple user on the same mapping task. Mapping execution is realized within this layer.
– The **Data and Remote Services Layer** is the back-end part of KAON. For local, in-memory operation storage based on a modified version of RDF API may be used. As mentioned above for enabling scalable and concurrent applications, KAON RDF Server is realized within the J2EE framework. Atomicity of updates is ensured by using transactions.

[9] http://kaon.semanticweb.org

As specified in section 4, mapping is specified and represented as an instance of a bridging ontology. Therefore, mappings can be created using OntoMat-SOEP – a tool for ontology and metadata management. However to simplify the task of establishing mappings, a plug-in for OntoMat has been implemented. A screen-shot of the user interface for mapping specification is presented in Figure 5. In this example two ontologies have been opened side by side, and in between an instance of the semantic bridging ontology is created using a simplified user interface.



university: organization --> db_instances: Organization
db_instances: Organization --> university: organization

Fig. 5. Creating Mappings Using KAON Tools

As mentioned earlier mapping execution is implemented within KAON API – the focal point of the KAON architecture. KAON API models the domain of ontology applications, providing classes such as Concept, Relation, Instance etc. and means for their creation and manipulation. Consequently, instances of the semantic bridging ontology can be expressed using KAON API constructs and processed and stored in the desired storage systems available within KAON.

The KAON mapping service supports the mapping execution phase under two distinct modes of operation:

– **Offline** (static) execution, transforming source ontology instances into target ontology instances once, without later synchronization,
– **Online** (dynamic) execution, where a connection between source and target ontology instances is constantly maintained.

Execution in either of the two modes is currently developed on the basis of the run-time structure model depicted in Figure 6, taking the upper half of the KAON mapping service box for offline execution and the lower half for online execution.

Offline execution is supported as a batch process. To execute, a previously generated instance of the semantic bridging ontology and an instance of the ontology to be transformed are passed to the offline mapping service. It can then perform the execution and generate an instance of the target ontology by applying transformations from the semantic bridging ontology. The offline mapping service as been implemented in Java.

Online execution is more complex, since the connection between instances of the source and target ontology is constantly maintained - notifications of changes to source ontology instance are mapped to changes in the target ontology

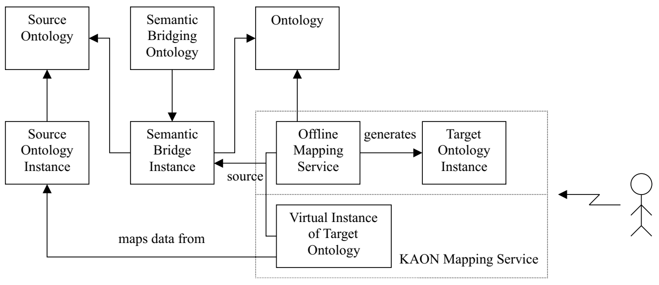


Fig. 6. Mapping Run-time Structure

instance and propagated to the user. The user has no means to detect that mapping is going on. To achieve this, as in the offline case, first an instance of semantic bridging ontology must be created. It and the source ontology instance are used to create a virtual instance of the target ontology handling online mapping. Execution occurs dynamically - e.g., when the user queries for all instances of a concept of the target ontology, the query is mapped into a query for the source ontology instance and executed there. Upon execution, the list of all instances obtained is then mapped into all instances of the target ontology and reported to the user.

6 Related Work

Much research has been done in the area of information integration. Existing information integration systems and approaches (e.g., TSIMMIS [6], Information Manifold [8], Infomaster[10], MOMIS[11], Xyleme [12]) are "centralized" systems of mediation between users and distributed data sources, which exploit mappings between a single mediated schema and schemas of data sources. Those mappings are typically modeled as views (over the mediated schema in the local-as-view approach, or over the sources schemas in the global-as-view approach) which are expressed using languages having a formal semantics. For scaling up to the Web, the "centralized" approach of mediation is probably not flexible enough, and distributed systems of mediation are more appropriate.

Furthermore, mapping approaches can mainly be distinguished along the following three categories: discovery, [14,3,5,10,1], mapping representation [9,1,11,13] and execution [4,11]. However, none of the proposed solutions has really encompassed the overall mapping process specially considering the evolution and consensus building of semantic bridges. Having this in mind, we have introduced the Ontology MApping FRAmework (MAFRA) as a basis for managing and executing mapping between distributed ontologies in the Semantic Web. Within MAFRA we provide an approach and conceptual framework that provides a generic view and figure onto the overall mapping process. In this paper we have set a specific focus on the semantic bridging phase corresponding to the mapping representation category. The approaches which resemble our approach more closely are [13] and [12]. Basically, our work has been motivated by the work done in [13], where an ontology has been specified for the translation between the domain-knowledge-base components and problem-solving-method components. The approach that comes nearest to ours has been described in [12]. They describe an approach for integrating vocabularies including means for mapping discovery and representing mappings with a focus on B2B applications (product catalogues) has been described. In contrast to our work, the RDFT ontology describes a set of core bridges to *(i)* lift XML tags to the RDF model and *(ii)* to define bridges between RDF(S) classes and properties and to *(iii)* translate transformation results back to XML. In the paper [12] it remains unclear, how execution specific information in the form of the constraint and transformation dimension is attached to the bridges. Furthermore, it is also not discussed if the overall process is executed statically or dynamically, where we offer both solutions.

[10] http://infomaster.stanford.edu/infomaster-info.html
[11] http://sparc20.ing.unimo.it/Momis/
[12] http://www.xyleme.com

7 Conclusion and Future Work

Ontologies may used for achieving a common consensus within a user community about conceptualizing, structuring and sharing domain knowledge. Based on the application scenario provided by Ontologging we have motivated that it is unrealistic to assume that one single ontology for different communities of users is realistic in real-world applications. We argue that decentralization has been one of the key elements for the scalability of the World Wide Web and its underlying applications. In order to balance the autonomy of each community with the need for interoperability, mapping mechanisms between ontologies have been proposed. In this paper we presented the Ontology Mapping Framework (MAFRA) supporting the interactive, incremental and dynamic ontology mapping process in the context of the Semantic Web. In this paper a specific focus has been set on the semantic bridging phase where we have provided a detailed description of a semantic bridge meta-ontology, that is instantiated when mapping between two domain ontologies.

In the future much work remains to be done. First, depending on the domain ontologies, data sources, application scenarios, user participation, capabilities and other factors further semantic bridges may be necessary. For example, procedural mechanisms may complement the taxonomy of semantic bridges. Thus, we consider the semantic bridging ontology as evolving. Second, considering the mapping process as a consensus building process of two communities, we will on the basis of our technological infrastructure KAON, perform an experiment how multi-user mapping may be efficiently supported. Third, we will develop an integrated LIFT tool that allows to lift several existing data representations including relational databases, XML-Schema, DTDs onto the same data model. Executing a dynamic mapping process keeping the autonomy of the different input data will be a challenging task.

Acknowledgements. Research for this paper was financed by European Commission, IST, project "Ontologging" (IST-2000-28293) and by Marie Curie Fellowship on Semantic Web Technologies. Special thanks to Gabor Nagypal for fruitful discussions on defining the semantic bridging ontology and Oliver Fodor for stimulating discussions on the lift component and cooperative mapping. Thanks to the students

Frank Westerhausen and Zoltan Varady who did the implementation work for the graphical user interface and the static transformation engine.

References

1. S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic integration of heterogeneous information sources. In *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering*, volume 36, pages 215–249. Elsevier Science B.V., 2001.
2. T. Berners-Lee. *Weaving the Web*. Harper, San Francisco, 1999.
3. W. Cohen. The whirl approach to data integration. *IEEE Intelligent Systems*, pages 1320–1324, 1998.
4. T. Critchlow, M. Ganesh, and R. Musick. Automatic generation of warehouse mediators using an ontology engine. In *Proceedings of the 5 th International Workshop on Knowledge Representation meets Databases (KRDB'98)*, 1998.
5. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the World-Wide Web Conference (WWW-2002)*, 2002.
6. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System. In *Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995.*, 1995.
7. S. Khoshafian and G. Copeland. Object identity. In *Proceedings of the 1st ACM OOPSLA conference, Portland, Oregon, September 1986.*, 1985.
8. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of VLDB-96, 1996*, 1996.
9. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conferences on Very Large Databases*, pages 49–58, 2001.
10. A. Maedche and S. Staab. Measuring similarity between ontologies. In *Technical Report, E0448, University of Karlsruhe*, 2001.
11. P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000)*. Konstanz, Germany, 2000.
12. B. Omelayenko. Integrating Vocabularies: Discovering and Representing Vocabulary Maps. In *Proceedings of the First International Semantic Web Conference (ISWC-2002), Sardinia, Italy, June 9-12, 2002.*, 2002.
13. J. Y. Park, J. H. Gennari, and M. A. Musen. Mappings for reuse in knowledge-based systems. In *Technical Report, SMI-97-0697, Stanford University*, 1997.
14. E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
15. M.C. Rousset. Standardization of a web ontology language. *IEEE Intelligent Systems, March/April 2002*, 2002.
16. P.R.S. Visser, D.M. Jones, T.J.M. Bench-Capon, and M.J.R. Shave. An analysis of ontology mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering, Stanford CA., USA*, pages 164–72, 1997.

Conceptual Normalisation of XML Data for Interoperability in Tourism

Oliver Fodor[1], Mirella Dell'Erba[1], Francesco Ricci[1], Antonella Spada[1], and Hannes Werthner[1 2]

[1] eCommerce and Tourism Research Laboratory
ITC-irst, Italy
{fodor, dellerba, ricci, spada, werthner}@itc.it
[2] University of Trento, Italy

Abstract. Currently many tourism systems and standardisation initiatives adopt XML as a standard for data representation. Together with XML, XML Schema definition language is becoming more widely used for describing the structure and constraining the contents of the documents. However neither XML itself nor XML Schema provides sufficient mechanisms for the representation of data semantics.
In this paper we discuss the issue of conceptual normalisation of XML documents. Conceptual normalisation is introduced as a pre-processing step of our ontology-mediated framework for the harmonisation of electronic tourism systems. We propose to use RDF Schema as a mechanism for conceptual modelling of data sources and investigate the relation between semantics carrying constructs of XML Schema and RDF Schema. Accordingly we elaborate on the idea of transforming XML data into a conceptual model corresponding to RDF statements and vice-versa. Finally we propose a declarative solution and a high-level architecture of a toolkit facilitating this process.

1 Introduction

Tourism in its nature is an industry strongly dependent on information exchange. In reality, however, the domain is highly fragmented mainly due to the heterogeneity of data sources.

The problem of bringing together heterogeneous and distributed systems is known as the interoperability problem. The data heterogeneity is a well-known obstacle, and the current approach to achieve the data interoperability is, mainly, to write ad-hoc data interface programs for each pair of communicating systems. Experience shows that development and maintenance of these programs is expensive in terms of both time and money. The total effort required increases with the square of the number of communicating systems. Ontology-mediated document exchange provides a solution for this scalability problem. The data model of a source document is aligned with the representation specified by the ontology maintained by mediator. The data is transformed accordingly in both ways.

Nowadays the W3C standard for semi-structured data representation XML (eXtensible Markup Language) has been adopted by many systems and domain specific standards for data exchange over the Web. XML documents are often in accordance to the schematic description provided by XML Schema. Since XML Schema describes hierarchical structure for data representation it can be considered as a logical model for XML data. Nevertheless, XML Schema does not provide a unified mechanism for representing semantics of the data. Therefore an additional pre-processing step is necessary in order to facilitate the ontology-mediated solution for interoperability. Explicit representation of concepts of the data sources enables to align them with the concepts of the mediation ontology.

Conceptual schemata can be used within the mediation task to describe the semantics of information sources and to make the content explicit. Mediation based on conceptual schemata instead of logical is preferable because conceptual schemata have more clear semantics and are not bound to specific data structures. Presence of conceptual schemata simplifies identification and association of semantically corresponding information concepts and consequent resolution of semantic conflicts.

In this paper we present a bottom-up approach to the problem of *conceptual normalisation* of XML data sources. We identify and discuss several related issues: re-engineering of conceptual schemata from existing logical schemata, associating between logical and conceptual schemata and transforming of data instances into representation corresponding to the associated conceptual schema and its notions.

The conceptual normalisation process, introduced here, is a part of the ontology-mediated interoperability platform Harmonise[3]. Harmonise aims at the development of a comprehensive solution for electronic tourism systems providing them the possibility to freely cooperate by exchanging information in a seamless manner. However we believe that the issue of conceptual normalisation is relevant for a wider variety of solutions for data integration. In Harmonise we decided to make use of the W3C standards and recommendations for the Semantic Web. We believe that relying on widely accepted standards and technologies will make the Harmonise solution compatible with related initiatives. Therefore we use RDF Schema language (RDFS) for the representation of local conceptual schemata and RDF (Resource Description Framework) metadata format for representing the data instances.

This document is structured as follows: In the following Section we will introduce the overall picture of the Harmonise platform. The Harmonisation Consortium elaborated user requirements for the interoperability task and analysed relevant tourism standards to be considered as input for the harmonisation process. An ontology-mediated solution has been proposed, distinguishing two major tasks: conceptual normalisation and semantic mapping. In Section 3 we will focus on the conceptual normalisation as a pre-processing step for the semantic mapping task. We will describe four stages of the normalisation process: the re-engineering stage supporting the creation of conceptual schemata, association discovery and association definition stages for linking be-

[3] http:// www.harmonise.org/

tween the logical and conceptual schemata and the dynamic lift stage taking care of the XML-to-RDF transformation and vice versa. In Section 4 we will propose a declarative solution relying on a set of customisable normalisation templates providing basic association facilities between XML Schema and RDF Schema. Finally, in Section 5, we will draft a high-level architecture for the normalisation toolkit as a part of the Harmonise solution.

2 Harmonise - Interoperability for Tourism

The Harmonise project is an initiative financed by the European Commission (EC), under the 5th Framework RTD Programme, contract number IST-2000-29329. The primary aim of the project is to establish an open international consortium - Tourism Harmonisation Network (THN) - including major tourism stakeholders, domain experts and IT professionals. The THN will serve as a forum for discussing the interoperability issues and coordinating the related activities within the tourism domain. Further, Harmonise aims to provide a solution for the interoperability problem in tourism by means of the so-called Harmonise Platform.

The Harmonise Platform is an ontology-mediated solution relying on the newest technologies for knowledge representation and following the philosophy of the Semantic Web, i. e. adding semantic annotations to the data sources. The goal is to allow the participating tourism organisations to keep their proprietary data format and simultaneously cooperate with each other, by exchanging information in a seamless manner. This will be possible through the help of a mediator module - the *Harmonisation tool (H-tool)* - providing a mediation service between different proprietary data formats and transforming the fragmented services, data and events in a common environment, where those can be easily and transparently distributed. This mediator acts as a semantic gateway between systems, permitting the receiver to view the source as an extension of its own information system, without any concern for the differences in names and representations of data.

The H-tool is based on the following three technologies as illustrated in Figure 1:

– A tourism ontology *IMHO (Interoperability Minimum Harmonisation Ontology)* modelling and storing the basic concepts used in representing the content of information exchanges in tourism transactions.
– An interchange format *HIR (Harmonise Interchange Representation)* suitable to represent the instance data used within interoperable tourism transactions.
– A set of *mapping rules* defining the transformation of data from the proprietary local format into the Harmonise Interchange Representation, and vice versa. Each system participating in Harmonise will maintain its own set of rules; the mapping rules will be defined on the base of the semantic correspondence of the local data models with respect to the IMHO.

2.1 XML as Common Input Format

The Harmonise Consortium agreed on XML as the common syntax for systems participating in the harmonisation process. This constraint was carefully decided after a wide discussion with major tourism organisations, domain experts, industry leaders and IT professionals. While the XML format was already accepted in some major industry standards (e. g. OTA[4]), several standards are currently in the preparation phase or already undertaking the evolution step towards XML. This is also true for the legacy systems of participating National Tourism Boards and their local standards (Finland, France, Portugal). Further it was agreed that logical schemata (XML Schema) of the XML documents will be also provided by the participants.

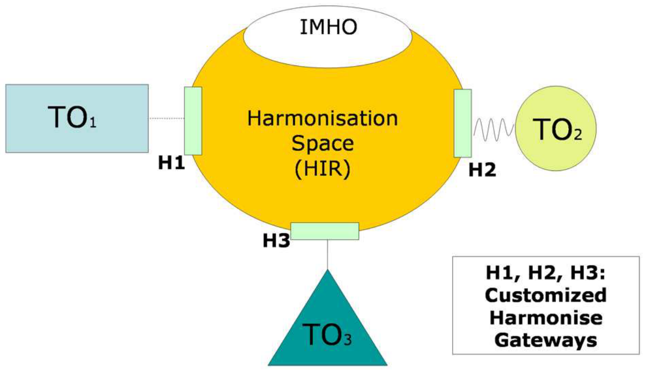

IMHO
TO1
Harmonisation Space (HIR)
H1
H2
TO2
H3
TO3
H1, H2, H3: Customized Harmonise Gateways

Fig. 1. Harmonise solution

Nevertheless, it is intended to keep the Harmonise solution open for possible future extensions towards other physical level formats.

2.2 Interoperable Minimal Harmonisation Ontology and Harmonise Interchange Representation

The Harmonise project will build the Interoperability Minimum Harmonisation Ontology by adopting the OPAL (Object, Process, Actor modelling Language) methodology. The core of OPAL is a business and enterprise ontology framework. The proposed framework is not a new language for ontology modelling, but intends to build, on top of an existing ontology language, domain-oriented constructs more familiar to enterprise experts than the basic modelling primitives of existing languages. The additional constructs are intended to ease the task of the tourism expert, by hiding the complexity of the underlying ontology representation formalism and supporting the construction of effective enterprise ontology.

As a starting point, it was decided to use RDF Schema language for representing the IMHO. This decision was taken based on an analysis of several ontology representation languages (DAML+OIL, UML, XML Schema) as a compromise between the expressive power, user friendliness and global distribution of the language. Although RDF Schema has limited expressive power we consider it as sufficient for the purpose of Harmonise whereas still allowing future extensions following new standards for representing ontologies.

Accordingly, RDF metadata format was adopted by Harmonise for representing the Harmonise Interchange Representation (HIR).

[4] OpenTravel Alliance, http://www.opentravel.org/

2.3 Analysis of the relevant standards

An analysis of relevant tourism standards (SIGRT[5], TourInFrance[6], OTA, xCBL[7], IFITT RMSIG[8], UN/EDIFACT TT&L[9], CEN/TC 329[10]) has taken place in the early phases of the Harmonise project. A comparison of overlapping concepts has shown that two different kinds of conflicts arise when an XML document has to be translated from one format to another one:

– Semantic clashes
– Structural clashes

Semantic clashes. Semantic clashes are clashes between concepts of different standards, or more precisely, between specific conceptual models or ontologies behind different standards. Typical semantic clashes are completely different concepts, different naming of concepts or different granularity. Identified semantic conflicts have been classified in eight categories. A detailed description can be seen in [13].

Table 1. Sample of semantic clashes

Different naming	`PostCode` vs. `PostalCode`
Different position	`Postcode` in `Address` rather than in `ContactInfo`
Different scope	`TelephonePrefix` and `TelephoneNumber` separated vs. `Prefix_TelephoneNumber` as single concept

Structural clashes. Structural clashes are caused by the heterogeneity of XML representation. Using XML format the same concept can be expressed in several different ways. XML Schema enables constraining of XML documents but this was designed for constraining the content of XML documents not for the conceptual representation. Within XML, structural clashes are mainly caused by the different usage of specific constructs, e.g. by a different usage of attributes rather than embedded elements or by expressing concepts in enumeration values.

Usually freely designed XML documents used for specific application purposes do not provide sufficient information about the semantics of the data. The semantics of XML elements used by Web applications is hard-coded into the applications and is typically not available in machine-processable form. This applies also to documents with available structural schemata (XML Schema), which in the most cases define the syntactical structure of XML documents without unified implicit representation of their meaning.

The example in Table 2 shows three different ways of expressing the concept `PostalCode` in XML.

Table 2. Structural heterogeneity of XML

```
<ContactInformation>
   <Address PostalCode="X-1220">
   Wannaby Street 59, Dreamtown</Address>
</ContactInformation>
```
```
<ContactInformation>
   <Address>
      <Street>Wannaby Street 59</Street>
      <City>Dreamtown</City>
      <PostalCode>X-1220</PostalCode>
   </Address>
</ContactInformation>
```
```
<ContactInformation>
   <Address>
       Wannaby Street 59,
       <PostalCode>X-1220</PostalCode>
       Dreamtown
   </Address>
</ContactInformation>
```

In [5] the authors show that an attempt to resolve both kinds of conflicts within one transformation step causes a scalability problem. Such a solution would lead to a rather complex set of rules and consequently to possible performance problems of the processing engine.

The separation between semantic and structural clashes indicates the need of a distinction between corresponding steps in the overall transformation process. Due to these arguments the Harmonise solution introduces a pre-processing step called conceptual normalisation. This step enables a separation of semantic mapping (resolution of the semantic clashes) from the concrete physical representation of data being transformed. In case different physical representations will be used in the future, the semantic mapping definitions will still remain valid.

Figure 2 depicts the two steps of data transformation from local to the harmonised representation (HIR).

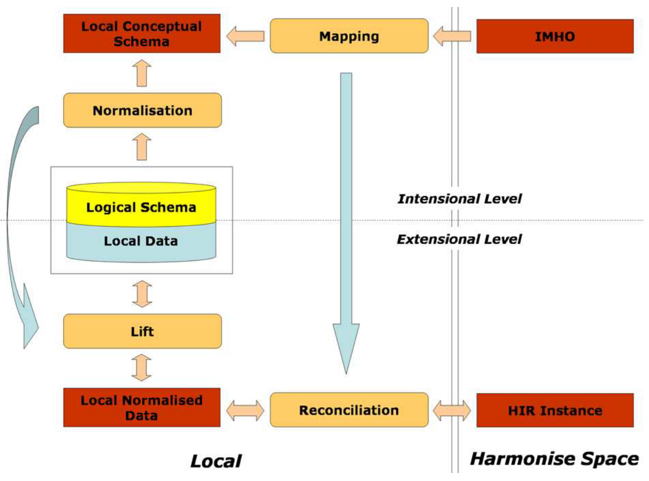


Fig. 2. Harmonise steps (forward harmonisation)

5 Sistema de Informação de Gestão de Recursos Turísticos, Portugal
6 National Tourism Board, France
7 XML Common Business Library
8 IFITT Reference Model Special Interest Group
9 United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport – Travel Tourism & Leisure
10 European Committee for Standardization/ Technical Committee Tourism Services

In the following Section we will discuss the basic issues related to conceptual normalisation.

3 Conceptual Normalisation

As described above, we consider conceptual normalisation as a pre-processing step facilitating the overall mediation task of the Harmonise ontology-based solution. Systems willing to participate in the harmonisation process will be encouraged to add semantics to their data by means of conceptual schemata and normalisation maps described later in this paper. We assume that there are no conceptual schemata available and a re-engineering based on existing data structures will be necessary in order to obtain them. However, we argue that available logical schemata (XML Schemata) are a promising base for the extraction of conceptual schemata. Obviously not all information about the data semantics can be extracted from the logical schemata, therefore an intervention of a system engineer is also considered. Within Harmonise we aim to develop a toolkit fully supporting the process of conceptual normalisation of XML schemata and respective data manipulation.

The process of conceptual normalisation is a bottom-up approach based on the existence of local logical schemata. We have identified four stages within the normalisation process as indicated in Figure 3. These stages will take place in a sequential step-by-step fashion.

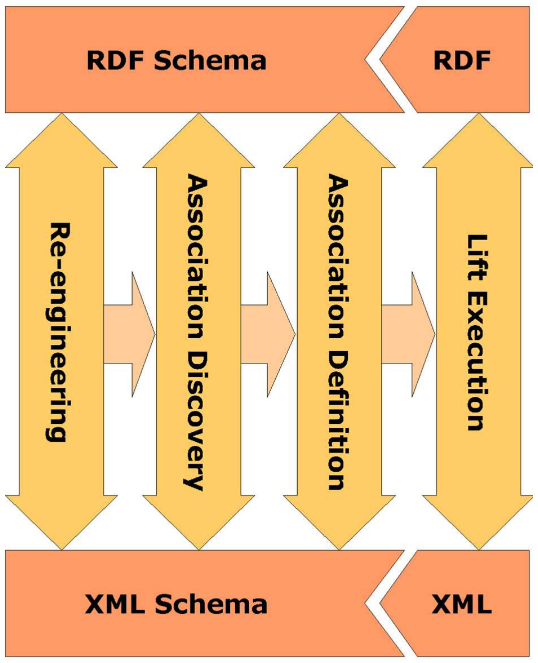


Fig. 3. Stages of conceptual normalisation process

Conceptual Level. To facilitate the semantic interoperability additional information has to be provided on the data sources. The semantics has to be expressed in an unified form (human and machine understandable) and referred to by the data to be harmonised. Therefore, Harmonise addresses the concept of *Normalized Conceptual Schemata* as explicit definition of the data semantics. Consequently, a new level of abstraction is to be considered for the representation of local data. We call this level *conceptual level*, also commonly referred to as data model level or semantic level.

The conceptual level has been introduced in the context of a three-schema framework [1] for database management systems. Three-layered approach for information representation on the Web was proposed in [2], where the semantic level was introduced above the lower object and syntax layers. Three layers for information integration (syntax, data model, ontology) were also adopted by [6] where data model layer serves for elimination of syntactical ambiguities of XML format.

Normalised Conceptual Schema. Originally, conceptual schemata played a key role in the development of a large variety of systems. In the above mentioned ANSI framework a conceptual schema is defined as “a unique central description of the information that might be in the database”. Conceptual schema is considered as a product of the conceptual modelling process.

In the Harmonise solution, the purpose of Normalised Conceptual Schema is to provide a unified human and machine understandable description of the concepts of local systems and relations among them. The important characteristic of NCS is its ability to be aligned with the formal representation of the Interoperable Minimal Harmonisation Ontology (IMHO). This intermediary product will simplify the process of semantic mapping since the mappings will be defined over representations of the concepts at the same abstraction level (conceptual level).

As stated in Section 2.2, in Harmonise the IMHO is represented by RDF Schema language. Therefore we propose to use RDF Schema as formalism for expressing the Normalized Conceptual Schemata for the systems participating in Harmonise. The convenience of RDF Schema for conceptual modelling has been discussed in [2], [3], [5] and variety of other sources. Accordingly, we use RDF format as a mechanism for representation of the data instances.

Conceptual Schema Re-engineering. Assuming that no conceptual schemata exist at the local sites these have to be obtained by a re-engineering of the available local data structures. The available XML Schemata will serve as the basis for the re-engineering of a corresponding conceptual schema. The elementary constructs and default semantics carrying elements of the logical schemata will be analysed in order to obtain an abstract conceptual model defining them. The semantic aspects of the logical schemata, e.g. the naming or granularity, will be preserved within the process. The purpose of this step is to resolve the structural heterogeneity of XML Schema and to separate the conceptual layer from the physical representation of the data.

Association Discovery. Association discovery is an intermediary step between the re-engineering and association definition and serves as a bridge between them. The knowledge obtained in the re-engineering step is used here to propose basic associations between the underlying logical and just produced conceptual schemata. Associations are derived from the default semantic correspondences of both schemata.

We consider a possible merge of the re-engineering and discovery phase and we will investigate the benefits in our future work.

Association Definition In this stage the associations between logical and conceptual schemata will be made explicit. The linking between the building constructs will be defined based on the associations proposed in the previous step and the explicit knowledge of the data structures. Therefore this step will be mainly depending on a human intervention. The definitions produced here will allow the projection of the underlying logical schema (tree) onto the conceptual schema (graph) defined in the re-engineering step and vice-versa. The symmetric character of the definitions is necessary for the backward harmonisation step where data is translated from the harmonised to the local representation.

Execution - Lift. According to the previously defined associations the data instances must be transformed so that they correspond to the representation of the concepts. Essentially, in this step RDF statements are derived from the XML documents and vice versa. This transformation of the data instances facilitates their further processing in the mediator. The lift can be seen as wrapping and un-wrapping of the data where the content is preserved, only some meta-information about the concepts is added. As already mentioned both directions must be supported therefore we distinguish between XML-to-RDF lift and RDF-to-XML extraction.

4 Declarative Approach

To the problem of conceptual normalisation we propose a declarative approach. This is based on a set of customisable *normalisation bridges* enabling declarative definition of the associations between logical and conceptual schemata. Further we propose a set of *normalisation heuristics* based on default semantic interpretations of XML Schema constructs. These heuristics will support automated and also manual phase of the re-engineering step. Normalisation heuristics will be interrelated with the association bridges, which enables to partially automate also the association discovery step. Therefore we consider our declarative approach as semi-automatic.

It is obvious that there is only a partial overlapping with respect to the semantic expressiveness of XML Schema and RDFS. The dimensions of e.g. enumeration of property values or basic data types cannot be preserved within the normalisation process. Hence, conceptual normalisation is a lossy process. However we consider the information lost acceptable for the initial release of the Harmonise solution. In the future this drawback can be removed by an extension of RDFS or by adoption of higher level modelling formalism based on RDFS.

Normalisation Ontology. First of all, we aim to create a normalisation ontology identifying all concepts related to the conceptual normalisation process. The mission of this ontology is to specify an unambiguous human comprehensible reference model as a base for the specification of normalisation bridges. The ontology will cover all elementary constructs of XML (e. g. element, attribute), XML Schema (e.g. element declaration, type definition), RDF (e. g. object, object identity), RDFS (e. g. class, property) and mechanisms for defining associations among them (normalisation heuristics, normalisation bridges). We intend to specify this ontology in common formats, like DAML+OIL, RDFS and OPAL.

Normalisation Heuristics. A set of heuristics related to the default semantic interpretations can be applied to support the modelling process in a semi-automatic way. These heuristics address the components of XML Schema and suggest their correspondences with components of RDF Schema. Usually, there are several possibilities to interpret an XML Schema construct in RDFS. To narrow down the number of options we also look at other relevant characteristics of XML Schema components, e.g. its naming (named, anonymous) or position in the schema (top-level, nested).

Consequently, the association discovery phase introduced in Section 3 can make benefit of the normalisation heuristics and accordingly automatically derive associations between the underlying schemata.

Table 3. Sample heuristics for conceptual modelling

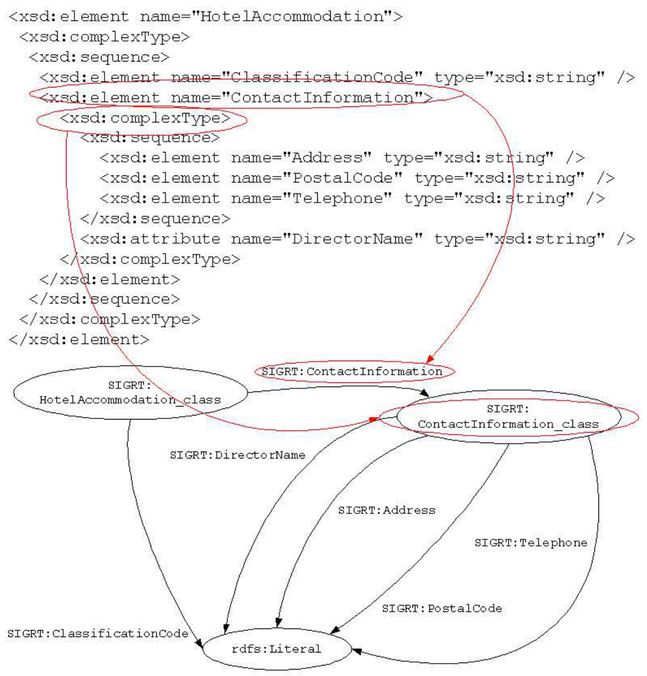
Given an unnamed complex type definition. Create a concept (rdfs:Class) and name it with the name of the declared parent element plus suffix "_class".
Given a local element declaration declaring the element E of a complex type. Create a relation (rdf:Property) and name it with the name of the element E. Set the domain of E to the class defined by the parent of the element declaration and the range of E to the class defined by the element's type.

Table 3 introduces two related sample heuristics. Figure 4 shows how a RDFS conceptual schema is created based on an existing XML Schema fragment in accordance to these heuristics.

Normalisation Bridges. Normalisation bridges are a set of customisable templates implementing associations between XML Schema and RDFS. The purpose of normalisation bridges is to associate components of two concrete schemata in order to support transformation of the data instances. Normalisation bridges are customized within the association definition phase.

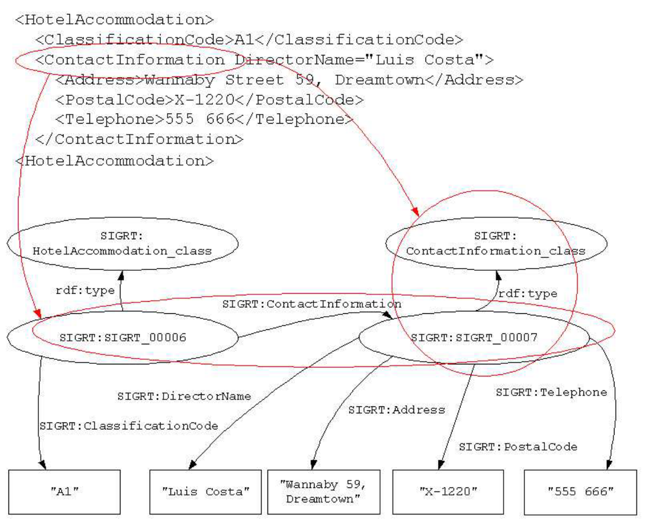
Each component to be associated by a normalisation bridge must be unambiguously addressed. To achieve this, we currently evaluate several techniques for naming and referencing components and structures of both schemata: XML Namespaces, XPath language and Uniform Resource Identifier (URI). We intend to rely on the URI mechanism for naming of RDF resources and XPath like naming convention (normalized uniform names) for XML Schema components. The later is described in detail in [16].

Normalisation templates must support associations in both directions. We expect that some association definitions will be symmetric whereas in some cases an explicit definition for each direction will be required. Also we expect that often several bridges will be relevant for one information item in



```
<xsd:element name="HotelAccommodation">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="ClassificationCode" type="xsd:string" />
   <xsd:element name="ContactInformation">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="Address" type="xsd:string" />
      <xsd:element name="PostalCode" type="xsd:string" />
      <xsd:element name="Telephone" type="xsd:string" />
     </xsd:sequence>
     <xsd:attribute name="DirectorName" type="xsd:string" />
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Fig. 4. Re-engineering of a conceptual schema



```
<HotelAccommodation>
  <ClassificationCode>A1</ClassificationCode>
  <ContactInformation DirectorName="Luis Costa">
    <Address>Wannaby Street 59, Dreamtown</Address>
    <PostalCode>X-1220</PostalCode>
    <Telephone>555 666</Telephone>
  </ContactInformation>
<HotelAccommodation>
```

Fig. 5. XML data transformation (lift)

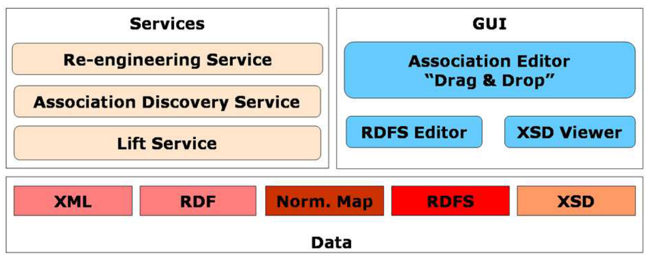


Fig. 6. Normalisation toolkit

the data instance. E. g. following two bridges apply to one occurrence of the element `ContactInformation` in the XML instance of the schema from Figure 4:

```
<xsd2rdfs_bridges:XSD2RDFSBridge
    rdf:ID="ContactInformation_element2property"
    xsd2rdfs_bridges:XSD_NuN="HotelAccommodation/*/ContactInformation"
    xsd2rdfs_bridges:RDFS_URI="SIGRT:ContactInformation" />

<xsd2rdfs_bridges:XSD2RDFSBridge
    rdf:ID="ContactInformation_type2class"
    xsd2rdfs_bridges:XSD_NuN="HotelAccommodation/*/ContactInformation/*"
    xsd2rdfs_bridges:RDFS_URI="SIGRT:ContactInformation_class" />
```

A set of customized normalisation bridges builds a *Normalisation Map*. Its mission is to support the transformation process of underlying data instances. A Normalisation Map will be explicitly related to the schemata it applies to. An appropriate mechanism will be introduced in order to explicate relations between bridges and their composition to build the map.

Figure 5 shows how a piece of XML document is transformed into RDF with respect to the schemata from Figure 4 and the corresponding map.

5 Harmonise Normalisation Toolkit

In order to support conceptual normalisation we aim to develop a normalisation toolkit as a part of the Harmonise Platform. Figure 6 displays the high-level architecture of the toolkit including several components. In this Section we will briefly describe these components. Naturally, in order to provide a solid solution an extensive requirement study and use case analysis will take place within the next phases of the Harmonise project. Consequently, we intend to develop a prototype solution as a module based on the KAON[11] ontology infrastructure.

The normalisation toolkit described in this paper supports only manual process of semantic normalisation. The proposed solution can be extended with automated engines, supporting the re-engineering process of conceptual schemata and the association discovery process.

RDFS Editor & Viewer. In the first release only manual creation of conceptual schemata will be supported. This requires an intervention of a system engineer who is familiar with the local data structures. This person manually extracts the concepts and relations from the existing XML Schemata and creates an appropriate RDFS model. The basic component supporting this process will be an editor tool providing an easy-to-use graphical user interface for editing RDFS concepts. Naturally, this tool will also serve as a viewer for already existing models. KAON provides an ontology-management component called SOEP which we consider as a candidate to re-use.

XML Schema Viewer. We agreed that XML Schema is the initial input for the normalisation process. Therefore, a component enabling viewing of schema documents is also necessary. A convenient visualization method for the XSD format will be proposed considering both efficiency and human readability. The functionality of this component can be limited to passive browsing.

Association Editor. Once we can browse through the normalized conceptual schemata (RDFS) and the underlying logical schemata (XSD) we need to provide a support for the definition of the associations among their elementary building blocks. For this purpose a GUI editor is foreseen. The

[11] http://kaon.semanticweb.org/

tool will cooperate with both schema viewers and provide a support for intuitive definition of the mappings, e.g. in a "drag-and-drop" fashion. The editor will implement the normalisation templates described earlier in this document. The output of this editor will be the Normalisation Map including a set of association definitions (normalisation bridges).

Lift Engine. This component will process a Normalisation Map at input and make use of it in order to transform the data to be harmonised. The lift engine will support both directions: XML-to-RDF lift and RDF-to-XML extraction. We expect to compile the normalisation map into a lower-level transformation language suitable for transforming XML, or RDF documents. XSLT is a good candidate for the XML-to-RDF whereas a RDF query language might be more appropriate for the RDF-to-XML direction.

6 Related Work

There is a rather big number of approaches to ontology-mediated integration of heterogeneous data sources. Most of these approaches, however, focus on integrated view over relational databases. Nevertheless, there are approaches [8] for mapping between XML documents although these mostly focus on direct transformation without considering the semantic level.

In [6] a three-level solution for B2B catalogue integration is presented. The authors introduce an RDF mapping language RDF-T [7], which allows to map between two RDF Schemata and translate RDF instance documents accordingly. Recently, RDF-T has been extended with basic support for mapping XML tags to RDFS resources. We follow the evolution of RDF-T and consider the language as a possible candidate for our normalisation bridges.

A rule-based approach to DTD to conceptual schema conversion is presented in [4]. This proposal bases on a set of conversion rules that takes into account DTD elements and attributes, syntactical constructs and heuristics related to their default semantic interpretations to generate a conceptual schema. This approach doesn't introduce any mechanism for an explicit definition of the associations between DTDs and corresponding conceptual schemata.

In [15] the authors propose to combine XML Schema with RDF Schema to enhance interoperability. Since this approach is relevant for proprietary designed schemata within applications it is not suitable for existing schemata defined by already adopted standards.

7 Conclusions

In this paper we introduced the process of semantic normalisation as a part of ontology-based harmonisation platform for tourism. Since the Harmonise solution builds on common standards for data representation and especially on the fundamental technologies for the Semantic Web, this approach can be also relevant for applications in other domains. Currently there are many systems and standards based on the XML technology and more are to be expected. Therefore our solution could be considered as a generic application for the Semantic Web.

From the perspective of electronic tourism systems participating in the harmonisation process we consider the semantic normalisation as an approximation step towards modern sophisticated solutions for interoperability. We believe that tourism is a very specific domain and its evolution is tightly coupled with the evolution of the information and communication technologies, the Internet in particular. Within Harmonise we intend to disseminate new promising technologies in order to preserve the tourism domain as one of the leading domains in the World Wide Web.

Currently we are working on a high-level ontology for the conceptual normalisation (heuristics and bridges) and its formal description. Within the next phases of the Harmonise project we intend to develop a prototype of the normalisation toolkit as a module for the KAON ontology framework.

8 Acknowledgement

Hereby we would like to thank the partners of Harmonise project for their contribution to the realisation of this paper. These are as follows:

– T6 (Italy)
– ICEP (Portugal)
– IFITT (Austria, IRL)
– CNR-IASI (Italy)
– LINK (Portugal)
– EC3 (Austria)

References

1. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems 75-02-08. FDT - Bulletin of ACM SIGMOD Vol. 7, No. 2 (1975) 1-140
2. Melnik, S., Decker, S.: A Layered Approach to Information Modelling and Interoperability on the Web. Proceedings of the Workshop on the Semantic Web at the Fourth European Conference on Research and Advanced Technology for Digital Libraries ECDL-2000, Lisbon, Portugal (2000)
3. Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I.: The Semantic Web: the Roles of XML and RDF. IEEE Internet Computing, Sept./Oct. 2000, Vol. 4, No. 5 (2000) 63-74
4. dos Santos Mello, R., Heuser, C. A.: A Rule-Based Conversion of DTD to a Conceptual Schema. Proceedings of the 20th International Conference on Conceptual Modeling ER 2001, Yokohama, Japan (2001) 133-148
5. Omelayenko, B., Fensel, D.: Analysis of B2B Catalogue Integration Problems. Content and Document Integration, In: Filipe, J., Sharp, B., Miranda, P. (eds.): Enterprise Information Systems III, Kluwer Academic Publishers, Dordrecht (2002) 270-277
6. Omelayenko B., Fensel D.: Scalable Document Integration for B2B Electronic Commerce. Submited (2001)
7. Omelayenko, B., Fensel, D., Klein, M.: RDF-T: An RDF Transformation Toolkit. Submitted (2001)
8. Su, H., Kuno, H., Rundensteiner, E. A.: Automating the Translation of XML Documents. Technical Report WPI-CS-TR-01-13, Computer Science Department, Worcester Polytechnic Institute (2001)
9. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. IEEE Computer, March 1992, Vol. 25, No. 3 (1992) 38-49

10. Visser, U., Stuckenschmidt, H., Vögele, T., Wache, H.: Enabling Technologies for Interoperability. Submitted to Transactions in GIS (2001)
11. Harmonise technical committee: Harmonise Project Technical Summary. http://www.harmonise.org (2002)
12. Harmonise technical committee: D.1.1: Electronic Classification of Tourism standards. http://www.harmonise.org (2002)
13. Harmonise technical committee: D.1.2: Report on comparison on standard methodologies. http://www.harmonise.org (2002)
14. Dell'Erba, M., Fodor, O., Ricci, F., Spada, A., Werthner, H.: Harmonise: A Solution for Data Interoperability. Submitted (2002)
15. Hunter, J., Lagoze, C.: Combining RDF and XML Schemas to Enhance Interoperability Between Metadata Application Profiles. Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China (2001) 457-466
16. Brown, A., Fuchs, M., Robie, J., Wadler, P.: XML Schema: Formal Description. W3C Working Draft, http://www.w3.org/TR/xmlschema-formal/ (25 September 2001)

RDFT: A Mapping Meta-Ontology for Business Integration

Borys Omelayenko

Division of Mathematics and Computer Science
Vrije Universiteit, De Boelelaan 1081a, 1081 HV,
Amsterdam, The Netherlands
borys@cs.vu.nl

Abstract. To create new added value the Semantic Web has to provide new means for business integration enabling open world-wide cooperation between various enterprises. Existing business integration techniques assume the enterprises having similar conceptual models of the objects being exchanged, and this limits their application areas. The Semantic Web needs to possess a technique capable of mapping different conceptual models in the business integration context. We propose a mapping meta-ontology built on top of RDF Schema using previous experiences in modelling mappings, specifics of RDF Schema and business integration tasks.

1 Introduction

Historically business integration has been performed within the Electronic Document Exchange[1] (EDI) paradigm via costly Value-Added Networks (VANs) that use private exchange protocols and provide full range of network services for large companies. Each EDI implementation requires a substantial labor effort to program and maintain, and this makes EDI unacceptable for small and medium enterprises searching for cheap and easy solutions. In this respect the World Wide Web and its successor the Semantic Web provide open standard representation means for modeling business information on the Web and allowing development a new generation of integration solutions.

At present time integrating two companies basically means integrating their database inputs and outputs available as EDI or XML documents. The databases are either accessible directly or via the Internet and with formal specification of possible operations on them are now often referred as Web services.

A number of toolkits have been developed to help the user in performing such kind of integration. The MS BizTalk[2] tool supports the integration of different databases accessible directly via SQL (or via EDI documents and appropriate wrappers). The documents pass through several steps. First, source documents are transformed into XML representation by means of wrappers. Second, the source XML schema is mapped to the target XML schema with the BizTalk Mapper tool that provides a user interface for mapping and generates XSLT scripts able to translate instance XML documents according to the maps. Finally, the resulting database record or EDI document is created from the target XML document.

Similar tasks are performed during the integration of different web services described in Web Service Definition Language[3] WSDL. In each message the service is expected to receive or generate an XML document according to the XML Schema specified in the service description. Capestudio[4] contains an XML mapping tool helping the user to map two XML schemas and automatically generate the correspondent XSLT file, similar to the Biztalk's Mapper. In addition, it provides advanced concerning working with online services and WSDL service descriptions (the same functionality is achieved with wrappers and SQL server in the BizTalk's database integration scenario).

The new generation of tools, e.g. the Unicorn[5] toolkit, uses ontologies as structured vocabularies that help the users in developing the mapping rules. In this case the user maps document's elements to ontological concepts and uses the hierarchy of ontological terms to navigate the elements. These terms are used to provide mediating names for database attributes, and do not constitute a mediating conceptual model. For example, it cannot support the case when in the source database a single object is stored in a single record while in the target database the same object is splitted up into several records.

At their present status the integration tools help in specifying the maps between quite similar conceptual models and require substantial programming effort for aligning different conceptual models. We aim at creation of an architecture that allows mapping different conceptual models via a mediating conceptual model.

The paper is organized as follows. The business integration task is outlined in Section 2 with typical conceptual models that need to be mapped, the mapping meta-ontology for aligning conceptual models represented in RDF Schema is discussed in Section 3, and the mapping tool is shown in Section 5, followed by conclusions and future research directions.

2 The Integration Task

Different companies play different roles in business collaborations and hence develop different (and often partial) models of business objects. Each document is specialized for a certain operation and represents an aggregation of properties of several objects specially grouped to support the operation. There does not exist any single exchanged document that represents a complete model of an object.

The mediator needs to aggregate these partial models to construct the mediating model and perform document integration via the aggregated mediating model rather than directly translate the documents.

[1] www.x12.org
[2] www.biztalk.org
[3] http://www.w3.org/TR/wsdl
[4] http://www.capeclear.com/products/capestudio/
[5] http://www.unicorn.com/

The mediating concepts might have the following features:

– All available knowledge about the objects is coming from input and output messages.
– The model must represent all the objects being exchanged and be sufficient for producing all the necessary documents required for interchange.
– The objects tend to change in time, and these changes are often marked with the timepoints of the documents or message validity times indicated in the messages.
– The model must evolve on-line with new customers coming to the mediator and bringing new views of the concepts they are willing to exchange.
– The documents are sometimes assigned to events in a non-trivial way, and a substantial effort may be needed to link the documents to workflow activities [1].

We treat the enterprise integration task as a service integration tasks. We assume that the enterprises, which we are going to integrate, are represented as Web services specified in WSDL. For each of them WSDL specifies the messages that are expected and/or produced and XML Schemas of the documents being transferred with the messages. These messages are produced by the company's ERP system [2], and the logic behind them is not accessible to the integration service, and even not important for the integration.

Accordingly, the integration service interacts with the world via messages (events) described in WSDL that have XML documents attached, which structure is specified in the XML Schemas included into WSDL descriptions.

2.1 Getting Conceptual Models from XML structures

Each message produced or expected by a company has an XML Schema specifying its structure. XML DTDs and Schemas are traditionally regarded as structural information that possesses no formal semantics, however they clearly preserve a large piece of knowledge about the domain objects they represent. A well-defined XML structure captures most of part-of relations between the domain objects and/or literals.

DTDs, which are much less expressive than XML Schemas, can be easily converted to reasonable conceptual models for the objects being described in the documents. A rule-based approach for extracting conceptual models from DTDs [3] provides a list of heuristic rules of two types: lexical rules that generate a taxonomy based on the inclusion relationship between XML tag names and restructuring rules that transfer XML trees into a conceptual model handling object nesting, cardinality constraints, etc.

Let us consider a DTD sample

```
<!ELEMENT A (B,C,(D|E))>
```

where all the elements `B`, `C`, `D`, and `E` are defined as literals `(#PCDATA)` and show how it may be converted to a conceptual model with the a slightly modified version of rules from [3].

Expression `D|E` is processed with rule `C(Choice)` that creates a new element `cD_E`. Elements `D` and `E` are converted to lexical concepts `cD` and `cE` that are connected to `cD_E` with relations `D` and `E` depicted in Figure 1. A composite element `A` is converted to concept `cA` (rule `CE-Composite Element`). Each of the elements `B` and `C` is converted to a literal concept `cB` or `cC` respectively (rule `SCE-Simple Component element`) and relations `B`, `C`, and `D_E` are added. More complicated examples incur other rules concerning repeating elements, cardinality, and complicated composite DTD elements.

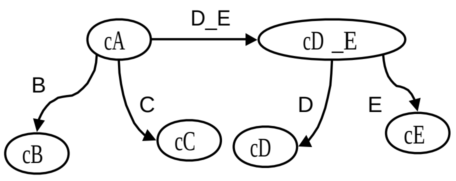


Fig. 1. A fragment of a conceptual model constructed from DTD

A similar algorithm has been proposed for converting XML DTDs to relational schemas and back [4]. Such algorithms take over all the routine tasks of converting DTDs to conceptual models. As a result the user can concentrate at extracting domain objects described in the documents and aligning them to the mediating or upper-level ontologies.

Finally, these conversion techniques ease the step of shifting the focus of the integration task from document interchange to concept exchange discussed in this paper.

2.2 RDF Modelling

There exist a number of means for representing conceptual models: ER and UML (conceptual modeling) F-logic, RDB, etc. (databases), DAML+OIL, description logic, etc. (knowledge engineering). Open business integration techniques on the Semantic Web might use Web standards for representing conceptual models on the Web, and currently RDF and RDF Schema [5] seem to be the best candidates for such a standard. There exist several extensions to RDF Schema, e.g. DAML+OIL[6] proposal that has more expressive power than RDF Schema itself. However, we naturally expect core languages to be adopted wider and faster than the extensions and hence we first concentrate on the use of RDF Schema still keeping in mind these extensions.

RDF (and its schema language RDF Schema) was primarily targeted at annotation of existing documents with conceptual models. The models are targeted at capturing some *partial* knowledge and we face a number of problems while representing *complete* models of events and documents. This leads to the following drawbacks:

– Properties are defined in RDF as first-class objects and they exist independently from classes. Such an interpretation is a bit unhandy for our needs. For example, the property `Address` must be mapped in different ways depending on whether it is attached to class `Invoice`, where it stands for billing address, or to class `DeliveryOrder`, where it stands for delivery address. RDF provides the means to distinguish between different object-property-value triples on the level of instance

[6] http://www.w3.org/TR/daml+oil-reference

documents, where each property is assigned to a certain class. However, property-class assignments are indistinguishable at the level of RDF Schema. DAML+OIL suffers the same problem.
- RDF Schema uses the `rdfs:domain` term which specifies the classes to which the property can be attached. Multiple occurrence of `rdfs:domain` has conjunctive semantics, that is if property `Address` can be used with two classes `Invoice` and `DeliveryOrder` then listing two `rdfs:domain` classes in the definition of `Address` is a wrong way to go. Such statement means that it can be used with a class, a subclass of both `Invoice` and `DeliveryOrder`, and such a class will most like have no sense from the domain point of view. One can model the fact that property `Address` can be used with both `Invoice` and `DeliveryOrder` is to create an artificial superclass for both `Address` and `DeliveryOrder` that has `Address` attached, but is not really informative from the domain point of view. This problem shows up in any attempt to build a mapping ontology where the same property of the ontology, e.g. `SourceClass`, needs to occur many times pointing to different source classes.
- RDF Schema provides an easy way to represent reification: the basic `rdf:Resource` class (the `Thing`) is an instance of and a subclass of itself. This creates certain difficulties in meta-modelling and tool development. In the integration tasks we need to model at three different layers: instance data, conceptual models, and meta-models (e.g. mapping meta-ontologies). These three layers must be tighly integrated and still very well distinguished. Hence, the introduction of three different `Thing`s may be suitable: instance-level resource, schema-level resource, and a meta-resource.
- Two types of things are mixed in common understanding of RDF: universal resource locators URL's that point to a file on the web and should be treated as filenames and universal resource identifiers URI's that look the same as URL's but represent logical names of the things in RDF Schema. Sometimes RDF Schemas stored in separate files need to be treated as logical units and statements about these files need to be created and processed.

We tried to overcome these problems in our model described in Section 3.

2.3 Things to be Mapped: Events, Documents, and Vocabularies

To be able to reason about the inputs and outputs of the companies being integrated we developed a conceptual model of WSDL (the part of WSDL specification that is important for the context of our mapping work skipping protocols and bindings). The model is directly derived from the WSDL specification and provides an RDF Schema for RDF annotations for WSDL documents.

Specifically, WSDL defines the following basic elements of services:

- `Types` that provide links to XML Schemas of the messages exchanged;
- Abstract definitions of `Messages` in accordance with `Types`;
- `Port Types` that specify input and output messages;
- `Bindings` that specify concrete protocols and formats for messages according to `Port Types`.

These elements allow describing the services but do not represent any temporal knowledge about the messages needed for the integration.

The Process Specification Language[7] PSL temporal ontology includes the classes: `activity`, `activity occurrence`, `timepoint`, and `objects`. Activities are performed during certain time intervals marked with timepoints, and each activity occurrence specifies a snapshot of an activity at a moment of time. The objects are defined as things that are not timepoints, not activities and not activity occurrences, and do not possess temporal properties. They may participate in activities at certain timepoints as defined by the `activity-timepoint-object` relation[8]. PSL provides basic temporal semantics of time intervals and timepoints, constraints on objects participating at certain activities, etc.

Accordingly, we extended our WSDL model with PSL ontology. The class diagram of the composite ontology is presented in Figure 2. In contains two root concepts: `mediator:Thing` and `psl:Thing`, subclasses of the former correspond to WSDL concepts and subclasses of the latter correspond to PSL classes. These classes are linked with a number of properties depicted in Figure 3. In both figures classes are represented with circles properties are shown with labeled edges linking the classes.

We must note that the Web Service Flow Language[9] WSFL built on top of WSDL provides the means to specify temporal and workflow information for services, and in some future it will make some of this model redundant. However, WSFL has a longer way to go to become a world-wide standard than WSDL or PSL. And again, in this work we focus at kernel technologies that has big chances of being widely accepted and used rather than at extensions of those.

WSDL annotations made according to our ontology allow performing inference over WSDL descriptions to validate the links established between the enterprises. Also they represent a bit stronger formalization of the services, e.g. by specifying legal ordering of the events. A sample of a realistic sequence of events with their order and timeout values is presented in Figure 4.

3 Mapping Meta-Ontology

The Common Warehouse Model (CWM) Specification [6] by the Object Management Group[10] provides a general architecture for a mapping ontology to be adopted by each specific mapping application. We adopt it to mapping RDF Schemas and specific concepts like events, messages, vocabularies, and XML-specific parts of conceptual models that

[7] http://www.mel.nist.gov/psl/

[8] RDF Schema is bounded to binary relations and provides no means to specify the ternary `activity-timepoint-object` relation. To model it we had to introduce a special class `psl:activity_at_timepoint_relation` attached to `psl:object` and linked to `psl:activity` and `psl:timepoint`.

[9] http://xml.coverpages.org/wsfl.html

[10] http://www.omg.org/

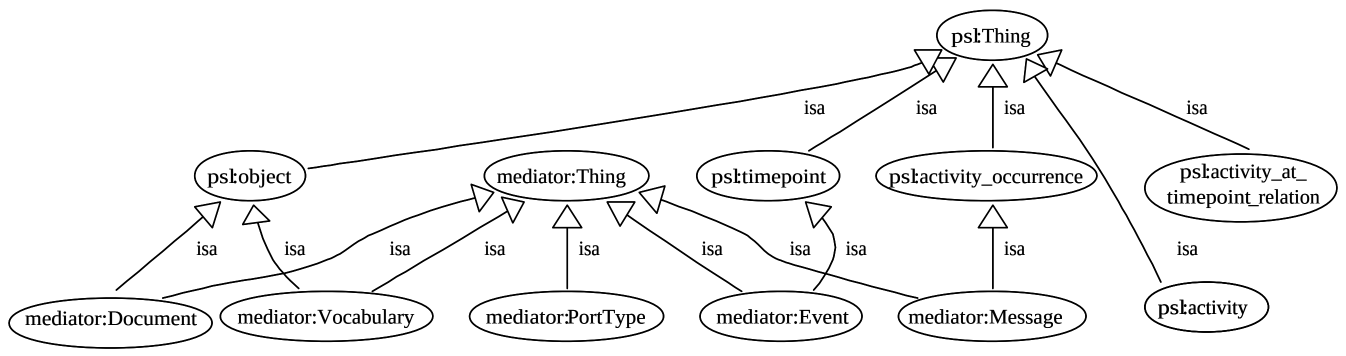


Fig. 2. WSDL extension (class diagram)

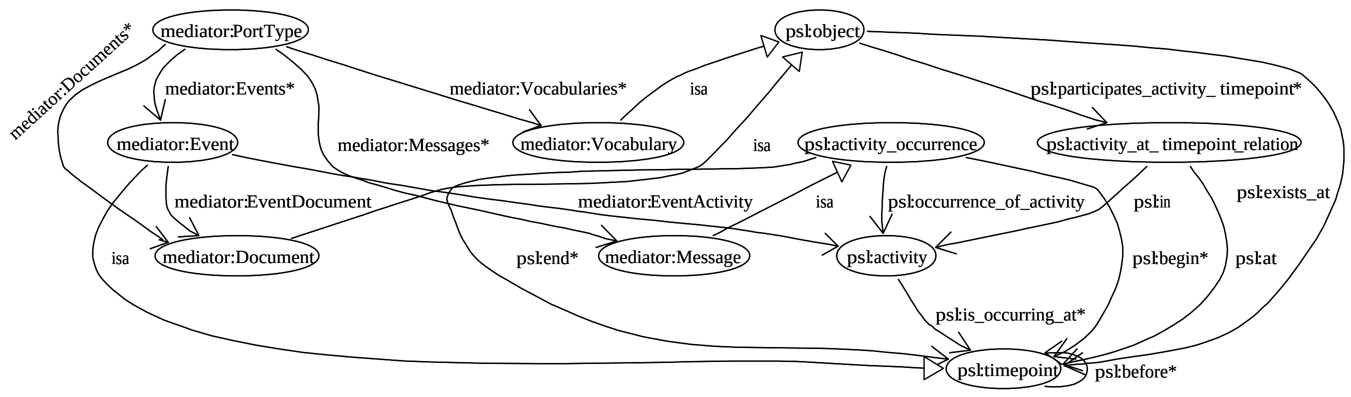


Fig. 3. WSDL extension (descriptions of classes)

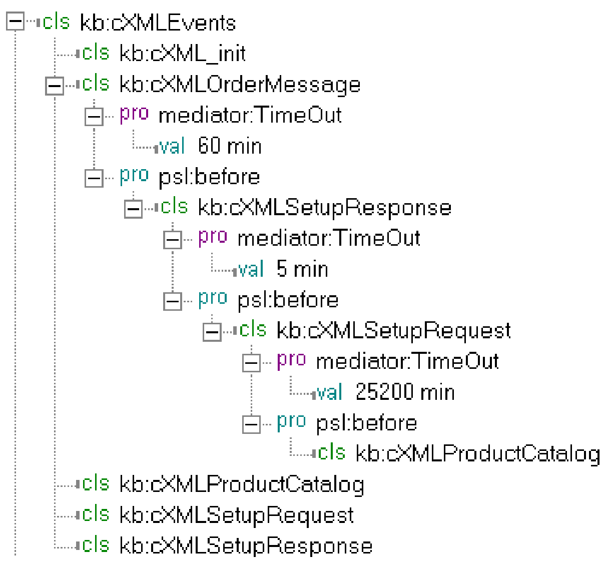

```
cls kb:cXMLEvents
    cls kb:cXML_init
    cls kb:cXMLOrderMessage
        pro mediator:TimeOut
            val 60 min
        pro psl:before
            cls kb:cXMLSetupResponse
                pro mediator:TimeOut
                    val 5 min
                pro psl:before
                    cls kb:cXMLSetupRequest
                        pro mediator:TimeOut
                            val 25200 min
                        pro psl:before
                            cls kb:cXMLProductCatalog
    cls kb:cXMLProductCatalog
    cls kb:cXMLSetupRequest
    cls kb:cXMLSetupResponse
```

Fig. 4. Event sequence

occur in the business integration tasks. CWM specifies the following basic primitives for mapping:

– Class `TransformationMap` a subclass of the `Transformation` class. The former is a container for `ClassifierMaps`.
– Each `ClassifierMap` links two groups of classes with two properties source and target with multiple occurrence referring to the groups of source and target classes being mapped with the map.
– Each `ClassifierMap` connects two groups of properties with the `featureMap` property that links it to the `FeatureMap` class connecting `source` and `target` properties (called features). `FeatureMap` also includes a link to another `ClassifierMap` mapping property values.
– Each `ClassifierMap` may connect a group of properties and a group of classes with the `ClassifierFeatureMap` class referenced via the `cfMapproperty`.

The CWM model concerns mapping generic conceptual models and seems to be too expressive for our needs. Mapping classes that normally refer to physical objects to properties that refer to properties of those is a large source of misunderstanding in the business tasks, while conceptually there is nothing wrong with it. In RDF meta-model both properties and classes are treated in a uniform way, while at the level of models those are clearly differentiated. Accordingly, we need to skip it in our mapping framework.

The RDFT (RDF Transformation) mapping meta-ontology[11] specifies a small language for mapping XML DTDs to/and RDF Schemas specially targeted for business integration tasks. The class diagram for the basic top-level classes is presented in Figure 5 using the same notation as our previous drawings.

[11] http://www.cs.vu.nl/ borys/RDFT

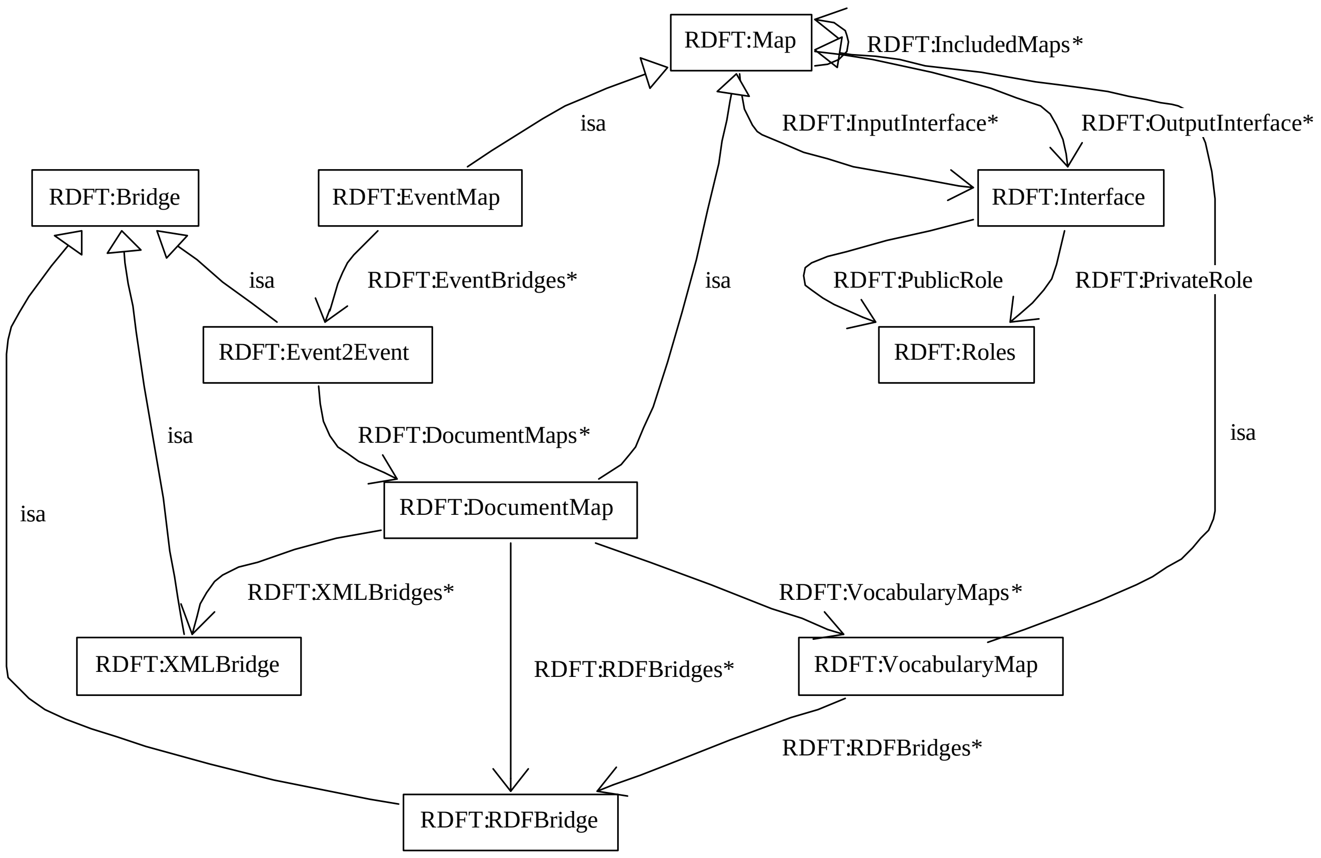


Fig. 5. Main RDFT classes: Bridge, Map, Interface, BridgeRelation, and Roles

The basic RDFT class is `Bridge` that connects two concepts (similar to the CWM's `ClassifierMap`). `Bridge` describes common properties of bridges allowing only one-to-many and many-to-one bridges opposite to CWM allowing many-to-many mappings [12].

The bridges also contain the `Relation` property pointing to one of the relations subclassed from the `BridgeRelation` class: `EquivalenceRelation` or `VersionRelation`.

– `Equivalence` bridges specify that the source element of a one-to-many bridge is equivalent to the target set of elements, and the source set of elements is equivalent to the target element for many-to-one bridges. E.g. a one-to-many bridge connecting a source class to a group of target classes states that the target group is semantically equivalent with respect to the instance data transformation task to the source element.
– `Version` bridges specify that the target set of elements form a (later) version of the source set of elements. Opposite to equivalence bridges, they assume that both source and target concepts belong to the same domain (or document standard), and may refer to two concepts with the same name (but different namespaces indicating versions), and imply that all the relations that held for the original concept must hold for the versioned concept, if the opposite is not stated explicitly.

Several types of `Bridges` are defined in RDFT:

– `Event2Event` bridges link different events, specify temporal event generation conditions, and link the events to the messages transmitted with them. They connect instances of the meta-class `mediator:Event`.
– Two kinds of `RDFbridges`: `Class2Class` and `Property2Property` bridges between RDF Schema classes and properties. `Class2Class` bridges contain `SourceClass` and `TargetClass` properties pointing to `rdfs:Class` instances. `Property2Property` bridges contains `SourceProperty` and `TargetProperty` properties pointing to `rdf:Property` instances. Again, only one-to-many and many-to-one bridges are allowed in RDFT. In RDF Schema properties are defined as first-class objects together with classes, and they capture most of domain knowledge. Classes mostly specify aggregation of properties, and thus we do not include class-to-property and property-to-class bridges in RDFT.
– Four kinds of `XMLBridges`: `Tag2Class` and `Tag2Property` bridges link source XML DTD tags and target RDF Schema classes and properties. `Class2Tag` and `Property2Tag` bridges connect RDF Schema classes and properties, and the elements of the target DTD. These are constructed from `SourceTag` and `TargetTag` properties in addition to

[12] Largely the objective of CWM is to model, i.e. to understand the things, while the objective of RDFT is to be able to transform instance data according to the mappings that imposes certain architectural restrictions.

the Source/Target-Class/Property properties mentioned above.

In some cases it is possible to declaratively specify the correspondence of property or class values linked by a bridge (e.g. by specifying the set of Class2Class bridges). If it is not feasible then we use the Expression property of a Bridge. It specifies an XPath [7] expression transforming instance data. XPath defines the means for two tasks: addressing data elements in XML documents and performing element or attribute value transformations (Chapter 4 of the specification). We use only the second part of the XPath functions (e.g. substring_before).

We need to create an ontology of a DTD to represent different DTDs in our framework and performing inference involving DTD elements and attributes (e.g. to check whether all the elements are mapped with RDFT bridges). The DTDs themselves are available to the mediating service, and only DTD elements and attributes must be annotated. Accordingly, we introduce two classes to represent them: XMLElement and XMLAttribute, subclasses of XMLTag. Properties SourceTag and TargetTag take XMLTag instances as their values.

Several assignments of a property to a class (e.g. a property with multiple cardinality) are not distinguishable in RDF Schema because they are not that significant from the modeling point of view. However, they are crucially important from the instance data transformation perspective, and we introduce class Role to specify each property-class assignment.

The bridges are grouped into maps. Each Map is a collection of bridges serving a single purpose. The maps are identified by their names[13] and form minimal reusable modules of RDFT bridges. Each Map can include other maps (the IncludedMaps property) and serves as a container for Bridges (the Bridges property). The maps use some class and property names within the bridges inside the maps and may be reused with other names passed as formal parameters. These parameters are formalized with the Interface class depicted in Figure 5. Each Interface class contains two properties: PublicRole and PrivateRole that specify the correspondence between external and internal names, correspondingly.

Connecting two services means connecting their events (instantiating the EventMap depicted in Figure 5) consisting of Event2Event bridges. Each of the bridges points to a DocumentMap aligning the documents attached to events, and, in turn, consisting of XMLBridges, RDFBridges and VocabularyMaps.

We do not impose any restriction on class names of a user ontology conforming the RDFT meta-ontology. Instead, all of them are marked as instances of RDFT meta-classes, as it is illustrated with a Class2Class bridge in Figure 8. In the figure user's classes are marked as S0, S1, and T0, and a bridge connecting them is marked with B_01.

It is important to differentiate the role of RDFT as a meta-ontology versus a template. Assume a definition of a metaclass mC and a definition of property mP with domain mC and range rdfs:Literal. Class iC, an RDF instance of mC will be a class definition that in addition to RDF Schema standard properties for a class definition (e.g. rdfs:subClassOf) possesses instantiated literal property iC.

However, we would expect *property definition* of iC to be applicable to mC rather than the *property itself*. So, we would rather need to call our RDFT ontology as a template ontology, while the term 'template' is not specified within RDF Schema. The instantiation semantics of RDF Schema class definitions, opposite to properties, seems to be acceptable for our needs.

Another important notion related to RDFT is its completeness, i.e. possibility to map arbitrary RDF Schemas. The term 'map' can be defined in different ways and each definition assumes a set of relations that need to be represented with the maps. In our case we try to map different part-of decompositions of object's properties, and one-to-many and many-to-one bridges seem to be sufficient for that. For extracting parts of properties we use quite expressive XPath language.

The main contribution of any architectural solution is to extract several most important tasks in the area and provide the means restricted in expressiveness but explicitly supporting these tasks. The unsupported cases may be handled by programming in expressive languages. We follow this principle in our mapping framework.

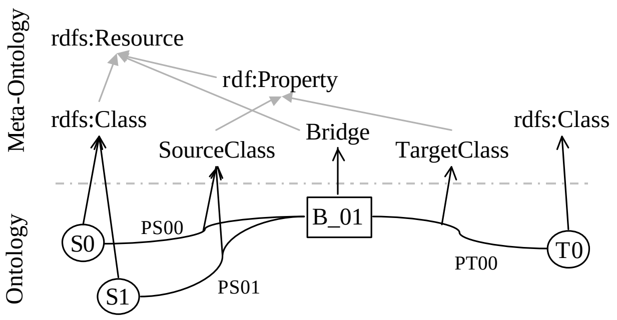


Fig. 8. The use of RDFT meta-ontology

4 Using Expressive Extensions of RDF Schema

It is always possible to express RDFT with expressive extensions of RDF Schema. However, before using them we should always ensure that this does not make things worse. People using DAML+OIL use specific DAML+OIL extensions quite seldom, and mostly they pick some minor but convenient extensions ignoring really expressive language constructs [8].

The business integration scenario is more specific and restricted that a generic modeling scenario. Business objects are quite simple and well-defined. Unlike the general case, most or all the objects might have physical representation, they are explicitly named, and are a part of a very shallow taxonomy.

[13] In this case we have a collision of resource identifiers URI's that's are used in RDF Schema and represent a logical name without any associated physical object and resource locators URL's that point to files somewhere on the Web. The maps are identified by their URI's but should be accessible via their URL's as reusable files with bridges.

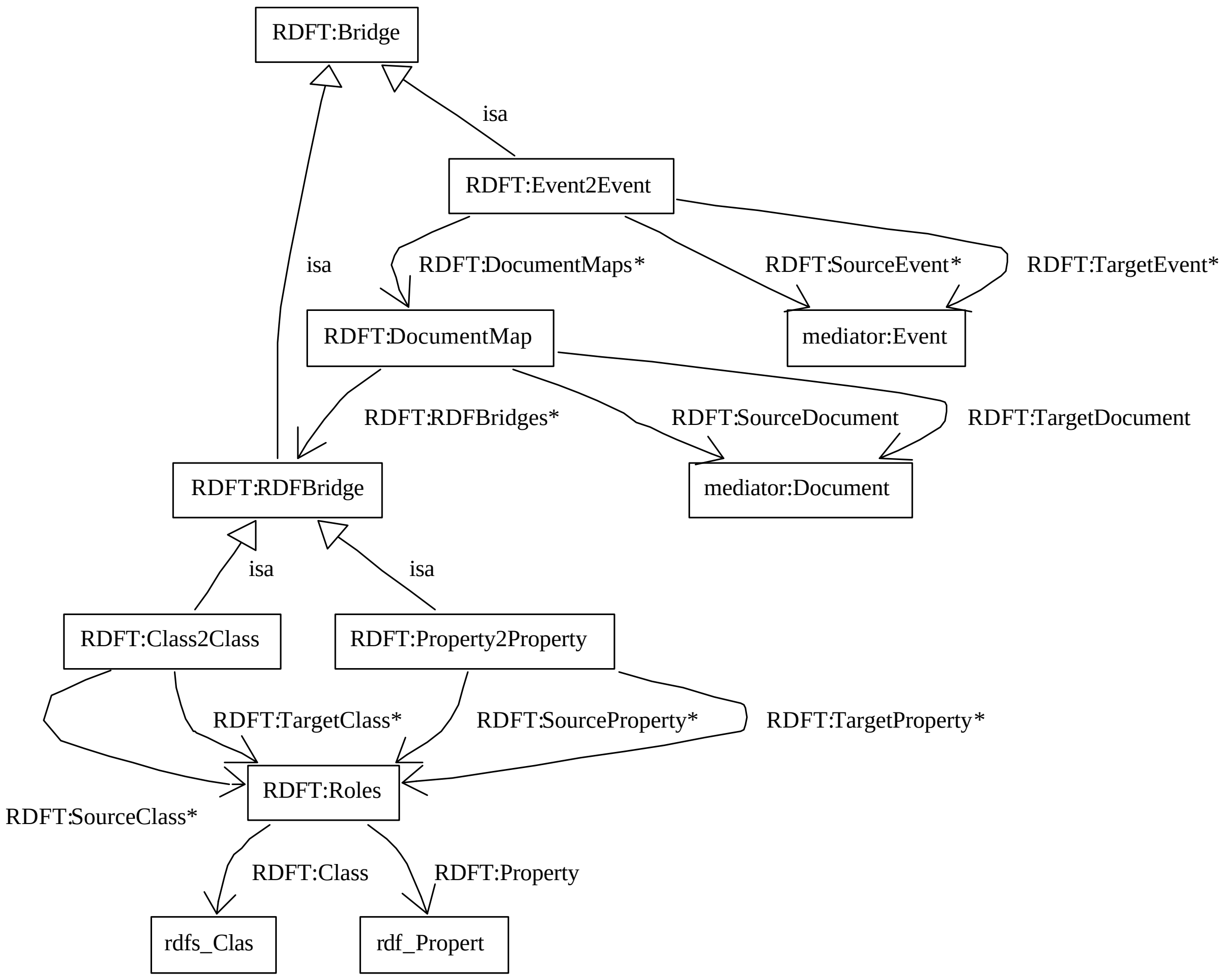
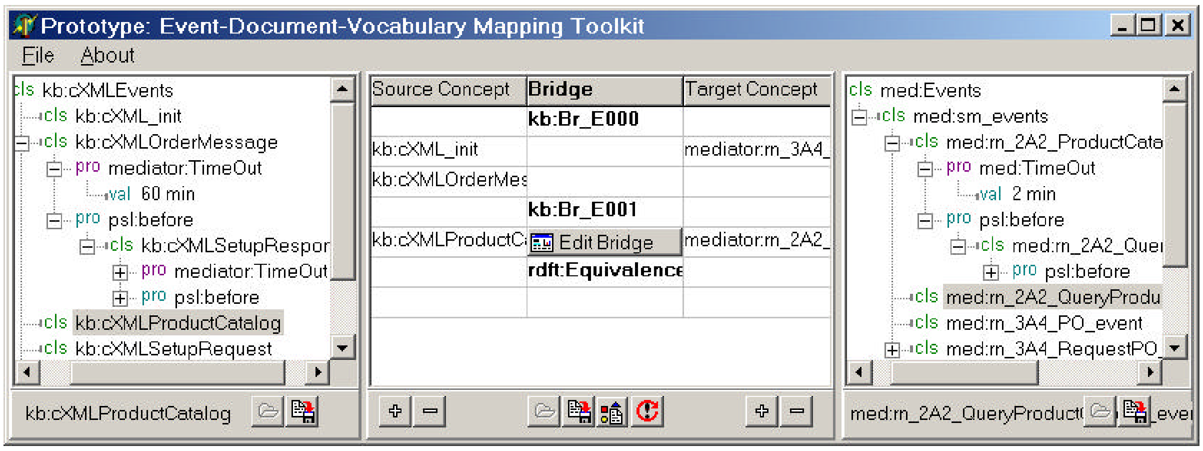


Fig. 6. Bridges and Roles



```
cls kb:cXMLEvents
  cls kb:cXML_init
  cls kb:cXMLOrderMessage
    pro mediator:TimeOut
      val 60 min
    pro psl:before
      cls kb:cXMLSetupRespor
        pro mediator:TimeOut
        pro psl:before
  cls kb:cXMLProductCatalog
  cls kb:cXMLSetupRequest

kb:cXMLProductCatalog
```

Source Concept	Bridge	Target Concept
	kb:Br_E000	
kb:cXML_init		mediator:m_3A4_
kb:cXMLOrderMes		
	kb:Br_E001	
kb:cXMLProductC:	Edit Bridge	mediator:m_2A2_
	rdft:Equivalence	

```
cls med:Events
  cls med:sm_events
    cls med:m_2A2_ProductCata
      pro med:TimeOut
        val 2 min
      pro psl:before
        cls med:m_2A2_Quei
          pro psl:before
    cls med:m_2A2_QueryProdu
    cls med:m_3A4_PO_event
    cls med:m_3A4_RequestPO_

med:m_2A2_QueryProduct( eve
```

Fig. 7. RDFT tool support: mapping two event sequences

Accordingly, we do not see any immediate needs for using more expressive language than RDF Schema.

The intention of the mapping is to be able parse the maps and generate XML transformation scripts able to translate instance XML documents attached to the messages. That is, for each target DTD element we need to trace which property of which object corresponds to this element, and then trace whether there is a map to a document from the source models. Finally, we need to find out which element from the source DTDs contains the original description of the property.

We need to use search-based inference engines, e.g. CLIPS[14], to find these chains. Then, each chain needs to be translated into an XML transformation language, e.g. XSLT.

5 Tool Support

We are currently developing a prototype tool providing an advanced map browsing and editing facilities and guiding the user through the event, document and vocabulary integration tasks. All these tasks are quite similar from the implementation point of view: all of them can be treated as RDF Schema mapping tasks. In the case of event mapping (see Figure 7 for a screenshot) the prototype tool allows the user to browse two RDF Schemas representing two event sequences and create RDFT maps between the schemas. Each bridge can be edited in details and necessary vocabulary bridges are edited with the same RDFT map editor applied to document schemas instead of event sequences.

Several important features are still under development in the tool: an interface to online web service described in WSDL, an inference engine, and an RDFT execution module. In addition, there are a number of small things that are too annoying to ignore them and too unimportant to talk about them.

6 Conclusions

We proposed a service integration architecture to match the following requirements:

– Allow switching from document transformation and exchange between the services to concept and model exchange.
– Annotate WSDL descriptions with concepts from temporal PSL ontology and some concepts from the business integration domain.
– Contain a mapping meta-ontology specifying mapping information between DTD elements, RDF Schema classes and properties, and events.
– Allow performing inference-based checks for completeness and consistency of the mappings.
– Allow compiling the data transformation chains represented by the mappings to a low-level XML transformation language.

In the paper we present our current progress in satisfying these requirements, namely switching between document-based to concept-based integration, WSDL annotation, mapping meta-ontology and preliminary steps in inference-based validation and compilation.

[14] http://www.ghg.net/clips/CLIPS.html

Acknowledgements. The author would like to thank Dieter Fensel, Christoph Bussler, Michel Klein, and Volodymyr Zykov for their helpful discussions and the reviewers for their comments.

References

1. Bae, H., Kim, Y.: A Document-Process Association Model for Workflow Management. Computers in Industry **47** (2002) 139–154
2. Bussler, C.: Modeling and Executing Semantic B2B Integration. In: Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce / E-Business Systems (RIDE-2EC'2002) (In conjunction with ICDE-2002), San Jose, USA, IEEE CS Press (2002)
3. Mello, R., Heuser, C.: A Rule-Based Conversion of a DTD to a Conceptual Schema. In Kunii, H., Jojodia, S., Solvberg, A., eds.: Conceptual Modeling - ER'2001. Number 2224 in LNCS, Yokohama, Japan, Springer (2001) 133–148
4. Lee, D., Chu, W.: CPI: Constraint-Preserving Inlining algorithm for mapping XML DTD to relational schema. Data and Knowledge Engineering **39** (2001) 3–25
5. Brickley, D., Guha, R.: Resource Description Framework (RDF) Schema Specification 1.0. Technical report, W3C Candidate Recommendation, March 27 (2000)
6. CWM: Common Warehouse Model Specification. Technical report, Object Management Group (2001)
7. Clark, J.: XSL Transformations (XSLT). Technical report, W3C Recommendation, November 16 (1999)
8. van Harmelen, F.: The Complexity of the Web Ontology Language. IEEE Intelligent Systems **17** (2002) 71–73

Enabling Services for Distributed Environments: Ontology Extraction and Knowledge Base Characterisation

Derek Sleeman[1], Stephen Potter[2], Dave Robertson[2], and W. Marco Schorlemmer[2]

[1] Department of Computing Science,
University of Aberdeen
sleeman@csd.abdn.ac.uk
[2] Division of Informatics,
University of Edinburgh
stephenp@aiai.ed.ac.uk, {dr,marco}@dai.ed.ac.uk

Abstract. Existing knowledge base resources have the potential to be valuable components of the Semantic Web and similar knowledge-based environments. However, from the perspective of these environments, these resources are often under-characterised, lacking the ontological and structural characterisation that would enable them to be exploited fully.
In this paper we discuss two currently independent services, both integrated with their environment via a brokering mechanism. The first of these services is an ontology extraction tool, which can be used to identify ontological knowledge implicit in a knowledge base. The second service involves characterising a given knowledge base in terms of the topic it addresses and the structure of its knowledge. This characterisation should permit a knowledge base to be located and assessed as a potential candidate for re-use in a more intelligent and flexible manner. The discussion of some related research into brokering systems illustrates the roles that these services can play in distributed knowledge architectures as precursors to problem-directed transformation and reuse of knowledge resources.

1 Introduction

The principal challenge for the Semantic Web community is to make machine-readable much of the material that is currently human-readable, and thereby enrich web operations from their current information-based state into a knowledge-centric form. Towards this end, for instance, the IBROW project is addressing the complex task of developing a brokering system which, given a knowledge base/knowledge source and a specification of the processing to be performed, would find an appropriate problem solver and perform any necessary transformation of the knowledge sources [1]. The focus of this paper is primarily on our research into two complementary, and currently independent, techniques that enable brokering systems to become more effective and more intelligent. These techniques, then, are intended to facilitate the reuse and transformation of knowledge.

The first of these techniques involves the extraction of domain ontologies from existing knowledge bases. Work on ontologies has played a central role in recent years in Knowledge Engineering, as ontologies have increasingly come to be seen as the key to making (especially web) resources machine-readable and -processable. Systems have been implemented which help individuals and groups develop ontologies, detect inconsistencies in them, and merge two or more. Ontologies are seen as the *essence* of a knowledge base, that is, they capture, in some sense, what is commonly understood about a topic by domain experts. For a discussion of how ontologies are often developed, see [2]. Recently, systems have been implemented which help domain experts locate domain concepts, attributes, values and relations in textual documents. These systems also often allow the domain expert to build ontologies from these entities; it has been found necessary, given the shortcomings of the particular text processed, to allow the domain expert, as part of the knowledge modelling phase, to add entities which are thought to be important, even if they are not found in the particular text [3].

Reflecting on this process has given us the insight that *knowledge bases*[3] themselves could act as sources of ontologies, as many programs essentially contain a domain ontology which although it may not be complete, is, in some sense, consistent. (Since if it were inconsistent this would lead, under the appropriate test conditions, to operational problems of the system in which the ontology is embedded). Thus, the challenge now becomes one of extracting ontologies from existing knowledge-based systems. The following section describes one approach for doing this, from, in the first instance, Prolog knowledge bases. As well as enabling their re-use, this technique can also be seen as performing a transformation of these knowledge bases into their implicit ontological knowledge.

In any distributed environment, before it can be re-used or transformed, an appropriate knowledge resource must be located. Doing this efficiently is not a trivial task, since it requires the ability to identify that a resource fulfils certain domain requirements and structural criteria without entailing the need to analyse the entire content of that resource. The second technique discussed in this paper addresses this problem, attempting to summarise the essentials of a knowledge base for this particular purpose.

The rest of the paper is structured as follows. Section 2 describes, with examples, the technique for acquiring ontological knowledge from knowledge bases in a (semi-)automatic fashion. Section 3 discusses the approach to characterising knowledge bases, describing them in a concise and succinct

[3] Throughout this paper, by “knowledge base” we mean some knowledge-bearing computer program, not necessarily expressed in some dedicated knowledge representation language, but for which the decision has been made to express the knowledge at a semantic, conceptual level. For our purposes, however, we assume that an explicit ontology describing the terms of such a knowledge base is *not* available.

manner to better allow their re-use. Section 4 gives a brief overview of a brokering system, in order to illustrate the role that the two techniques can play in facilitating knowledge services within distributed environments. Section 5 discusses related work, and, to conclude, Section 6 summarises the paper.

2 Extracting Ontologies from Prolog Knowledge Bases

The method used to hypothesise ontological constraints from the source code of a knowledge base is based on Clark's completion algorithm [4]. Normally this is used to strengthen the definition of a predicate given as a set of Horn clauses, which have single implications, into a definition with double-implication clauses. Consider, for example, the predicate $member(E, L)$ which is true if E is an element of the list, L:

$$\begin{array}{l} member(X, [X|T]) \\ member(X, [H|T]) \leftarrow member(X, T) \end{array}$$

The Clark completion of this predicate is:

(1) $member(X, L) \leftrightarrow L = [X|T] \ \vee \ (L = [H|T] \wedge member(X, T))$

Use of this form of predicate completion allows us to hypothesise ontological constraints. For example, if we were to assert that $member(c, [a, b])$ is a true statement in some problem description then we can deduce that this is inconsistent with our use of $member$ as constrained by its completion in expression (1) above because the implication below, which is an instance of the double implication in expression (1), is not satisfiable.

(2)
$member(c, [a, b]) \rightarrow [a, b] = [c|T] \ \vee \ ([a, b] = [H|T] \wedge member(c, T))$

Normally Clark's completion is used for transformation of logic programs where we are concerned to preserve the equivalence between original and transformed code. It therefore is applied only when we are sure that we have a complete definition for a predicate (as we had in the case of $member$). However, we can still apply it in "softer" cases where definitions are incomplete. Consider, for example, the following incomplete definition of the predicate $animal(X)$:

$$\begin{array}{l} animal(X) \leftarrow mammal(X) \\ animal(X) \leftarrow fish(X) \end{array}$$

Using completion as above, we could derive the constraint:

$$animal(X) \rightarrow mammal(X) \ \vee \ fish(X)$$

This constraint is over-restrictive since it asserts that animals can only be mammals or fish (and not, for instance, insects). Nevertheless, it is useful for two purposes:

– As a basis for editing a more general constraint on the use of the predicate '$animal$'. We describe a prototype extraction tool, which includes a basic editor, for these sorts of constraints in Section 2.1.
– As a record of the constraints imposed by this *particular* use of the predicate '$animal$'. We describe an automated use of constraints under this assumption in Section 2.2.

2.1 A Constraint Extraction Tool: the *EXTRACTexP* System

We have produced a basic system for extracting ontological constraints of the sort described above from Prolog source code. Our tool can be applied to any standard Prolog program but is only likely to yield useful constraints for predicates which contain no control-effecting subgoals (although non-control-effecting goals such as $write$ statements are accommodated). While, in theory at least, the approach can be applied to programs of any size, we will now demonstrate the current tool using an example involving a small number of predicates.

Figure 1 shows the tool applied to a simple example of animal classification, following the introduction of the previous section. The Prolog code is:

```
animal(X) :- mammal(X).
animal(X) :- fish(X).
mammal(X) :- vertebrate(X), warm_blooded(X),
                                          milk_bearing(X).
fish(X) :- vertebrate(X), cold_blooded(X), aquatic(X),
                                         gill_breathing(X).
```

which corresponds to the Horn Clauses:

(3)
$$\begin{array}{rl} animal(X) & \leftarrow mammal(X) \\ animal(X) & \leftarrow fish(X) \\ mammal(X) & \leftarrow vertebrate(X) \wedge warm\_blooded(X) \\ & \qquad \wedge milk\_bearing(X) \\ fish(X) & \leftarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \\ & \qquad \wedge gill\_breathing(X) \end{array}$$

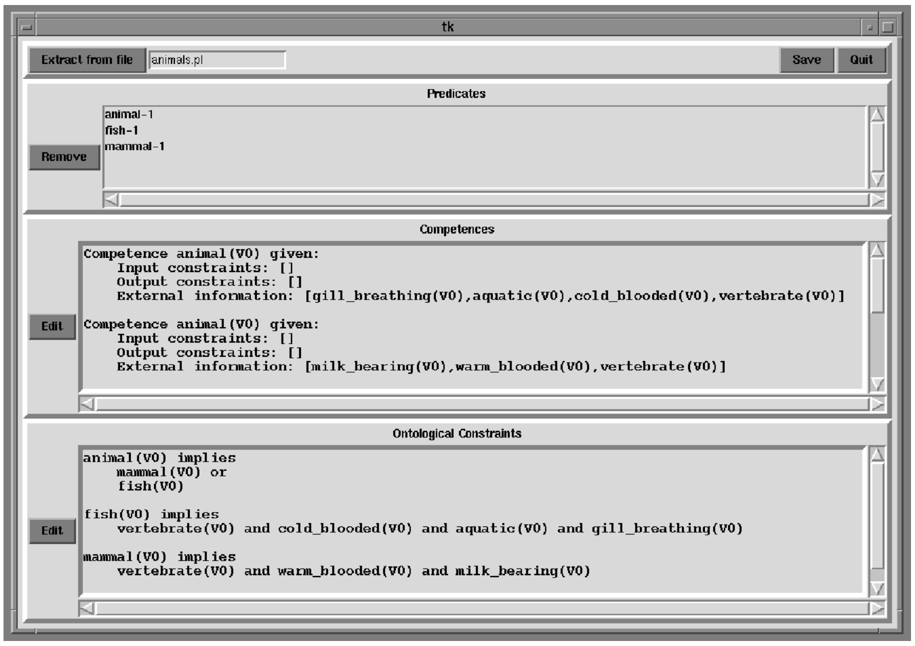
The constraints extracted for this program (seen in the lower window of Figure 1) are:

(4)
$$\begin{array}{rl} animal(X) & \rightarrow mammal(X) \vee fish(X) \\ fish(X) & \rightarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \\ & \qquad \wedge gill\_breathing(X) \\ mammal(X) & \rightarrow vertebrate(X) \wedge warm\_blooded(X) \\ & \qquad \wedge milk\_bearing(X) \end{array}$$

If it is deemed necessary, the user of the tool can then choose to edit manually the constraints. We show in Section 2.2 how these constraints, which, in this case, were extracted completely automatically from the Prolog source code, can be used to check another Prolog program purporting to adhere to the same ontology.

2.2 Ontological "Safe Envelopes"

The idea of running programs within ontological "safe envelopes" was introduced in [5]. Programs are run according to the normal execution control regime of the language concerned but a record is kept of the cases where the execution uses terminology which does not satisfy a given set of ontological constraints. When this happens we say the execution has strayed outside its safe envelope (from an ontological point of view). This sort of checking is not intended to alter the execution of the program in any significant way, only to pass back retrospective information about the use of terminology during an execution. This style of checking can be implemented elegantly for languages, such as Prolog, which permit meta-interpretation, allowing us to define the control structure for execution explicitly and then to augment this with appropriate envelope checking. The Horn clauses shown



```
animal-1
fish-1
mammal-1
```

```
Competence animal(V0) given:
    Input constraints: []
    Output constraints: []
    External information: [gill_breathing(V0),aquatic(V0),cold_blooded(V0),vertebrate(V0)]

Competence animal(V0) given:
    Input constraints: []
    Output constraints: []
    External information: [milk_bearing(V0),warm_blooded(V0),vertebrate(V0)]
```

```
animal(V0) implies
    mammal(V0) or
    fish(V0)

fish(V0) implies
    vertebrate(V0) and cold_blooded(V0) and aquatic(V0) and gill_breathing(V0)

mammal(V0) implies
    vertebrate(V0) and warm_blooded(V0) and milk_bearing(V0)
```

Fig. 1. Ontology extraction tool

in expression (5) provide a basic example (extended versions of this appear in [5]).

(5)

$$\begin{array}{l} solve(true, \{\}) \\ solve((A \wedge B), E_a \cup E_b) \ \leftarrow \ solve(A, E_a) \wedge solve(B, E_b) \\ solve((A \vee B), E) \ \leftarrow \ solve(A, E) \ \vee \ solve(B, E) \\ solve(X, E \cup \{C|(X \rightarrow C \ \wedge \ not(C))\}) \ \leftarrow \ clause(X, B) \\ \hfill \wedge solve(B, E) \end{array}$$

In the expressions above, $clause(X, B)$ means that there is a clause in the program satisfying goal X contingent on conditions, B (where there are no conditions, B has the value $true$). The implication $X \rightarrow C$ is an ontological constraint of the sort we are able to derive in the extraction tool of Section 2.1. The operators $\leftarrow$, $\wedge$, $\vee$, and $\cup$ are the normal logical operators for (left) implication, conjunction, disjunction and union, while $not(C)$ is the closed-world negation of condition C.

The effect of the meta-interpreter above is to test each successful goal in the proof tree for a query against the available ontological constraints. The first clause of (5) matches the goal $true$, which, as might be expected, violates no ontological constraints (and so, the empty set is returned). The second and third clauses deal with conjunctions and disjunctions of goals respectively. In the case of the former, the union of the sets of violated constraints is returned; in the latter case, the set generated by the succeeding goal is returned.

In the final clause, if an asserted `clause(X,B)` is found which satisfies the current goal, X, then the conditions, B, of this goal become subgoals of the interpreter, while the goal itself is tested against the ontological constraints. If a constraint exists ($X \rightarrow C$) that is not found to be consistent with the known facts of the current situation ($not(C)$, under the closed-world assumption), then it is added to the set of violated constraints. When a goal and its subgoals have been solved, then the interpreter exits with success, returning the set of all violated constraints; if, on the other hand, a goal cannot be solved, then the interpreter fails.

For example, suppose we have the following information about animals, `a1` and `a2`, using the animal ontology of Section 2.1.

```
animal(a1).
vertebrate(a1).
warm_blooded(a1).
milk_bearing(a1).
animal(a2).
vertebrate(a2).
cold_blooded(a2).
terrestrial(a2).
```

We could query this database in the normal way, for example by giving the goal `animal(X)` which yields solutions with `X = a1` and `X = a2`. If we want to perform the same query while checking for violations of the ontological constraints we extracted in Section 2.1, then each of these facts is asserted in the form, e.g., `clause(animal(a1),true)`, and we pose the query via the meta-interpreter we defined above — the appropriate goal being `solve(animal(X), C)`. This will yield two solutions, as before, but each one will be accompanied by corresponding ontological constraint violations (as corresponding instances of the variable `C`). The two solutions are:

`X = a1` $\qquad$ `C =` $\{\}$
`X = a2` $\qquad$ `C =` $\{mammal(a2) \vee fish(a2)\}$

When presented with the first goal, `animal(a1)`, the interpreter matches this with `clause(animal(a1),true)` from the database; the precondition `true` generates no ontological problems, and from expression (4), the constraint $mammal(a1) \vee fish(a1)$ is placed on `animal(a1)`. Now, the additional facts in the database and the other ontological constraints allow the conclusion `mammal(a1)` to be drawn, so it is *not* the case that $not(mammal(a1) \vee fish(a1))$ is true (as tested by the fourth clause of the interpreter), so no constraints are violated, and the empty set is returned.

The solution of the second goal, `animal(a2)` proceeds in a similar fashion, but in this instance, the constraints and database facts do not allow either `mammal(a2)` or `fish(a2)` to be proved. Hence, under the closed-world assumption, $not(mammal(a2) \vee fish(a2))$ *is* true, and so this constraint has been violated (this in spite of the fact that the database allows the goal `animal(a2)` itself to be proved).

2.3 Extracting Ontologies from Other Sorts of Knowledge Bases

The majority of knowledge sources are not in Prolog so for our extraction tool to be widely applicable it must be able to deal with other sorts of source code. This would be very hard indeed if it were the case that the ontological constraints we extract have to encompass the entire semantics of the code. Fortunately, we are not in that position because it is sufficient to extract some of the ontological constraints from the source code — enough to give a partial match when brokering or to give a starting point for constraint editing. The issue when moving from a logic-based language, like Prolog, to a language perhaps having more procedural elements is how much of the ontological structure we can extract. We discuss this using CLIPS as an example.

Suppose we have the following CLIPS facts and rules:

```
(deftemplate person "the person template"
        (slot name)
        (slot gender (allowed-symbols female male)
                                       (default female))
        (slot pet))

(deftemplate pet "the pet template"
        (slot name)
        (slot likes))

(deffacts dating-agency-clients
        (person (name Fred) (gender male) (pet Tiddles))
        (person (name Sue) (pet Claud))
        (person (name Tom) (gender male) (pet Rover))
        (person (name Jane) (pet Squeak))
        (pet (name Tiddles) (likes Claud))
        (pet (name Claud) (likes Tiddles))
        (pet (name Rover) (likes Rover))
        (pet (name Squeak) (likes Claud)))

(defrule compatible
        (person (name ?person1) (pet ?pet1))
        (person (name ?person2) (pet ?pet2))
        (pet (name ?pet1) (likes ?pet2))
        =>
        (assert (compatible ?person1 ?person2)))
```

To extract ontological constraints from these using the current version of the EXTRACTexP tool we must translate these CLIPS rules into Horn clauses. We outline below, in informal terms, the transformation algorithm needed for this task:

- For each CLIPS rule, take the assertion of the rule as the head of the Horn clause and the preconditions as the body of the clause.
- Consider each head, body or CLIPS fact as an object term.
- For each object term, refer to its `deftemplate` definition and translate it into a series of binary relations as follows:
    - Invent an identifier, I, for the instance of the object.
    - The relation $object(T, I)$ gives the type of object, T, referred to by instance I.
    - The relation $A(I, V)$ gives the value, V, for an attribute A of instance I.

Applying this algorithm to our CLIPS example yields the Horn clauses shown below:

$$compatible(Person1, Person2) \leftarrow$$

$$\begin{array}{l} object(person, O1) \wedge name(O1, Person1) \wedge pet(O1, Pet1) \wedge \\ object(person, O2) \wedge name(O2, Person2) \wedge pet(O2, Pet2) \wedge \\ object(pet, O3) \wedge name(O3, Pet1) \wedge likes(O3, Pet2) \end{array}$$

$$\begin{array}{llll} object(person, p1) & name(p1, fred) & gender(p1, male) & pet(p1, tiddles) \\ object(person, p2) & name(p2, sue) & gender(p2, female) & pet(p2, claud) \\ object(person, p3) & name(p3, tom) & gender(p3, male) & pet(p3, rover) \\ object(person, p4) & name(p4, jane) & gender(p4, female) & pet(p4, squeak) \end{array}$$

$$\begin{array}{lll} object(pet, x1) & name(x1, tiddles) & likes(x1, claud) \\ object(pet, x2) & name(x2, claud) & likes(x2, tiddles) \\ object(pet, x3) & name(x3, rover) & likes(x3, rover) \\ object(pet, x4) & name(x4, squeak) & likes(x4, claud) \end{array}$$

This does not capture the semantics of the original CLIPS program, since, for example, it does not express notions of state necessary to describe the operation of CLIPS working memory. It does, however, chart the main logical dependencies, which is enough for us then to produce ontological constraints directly from EXTRACTexP. This translation-based approach is the most direct route to constraint extraction using our current tool but we anticipate more sophisticated routes which perhaps do not translate so immediately to Horn clauses.

Extending this technique beyond knowledge representation languages to enable the extraction of ontological information from conventional procedural languages such as C would prove difficult. Programmers of these languages have no incentive to express their code at a conceptual level, with the result that the ontological constraints, insofar as they are expressed, tend to be embedded in the control elements and structure of the code to a greater extent. Code written in object-oriented languages, such as Java and C++, is potentially more susceptible to ontological extraction of this sort, since the object-oriented paradigm encourages the programmer to codify the concepts of the domain in an explicit and structured manner (the CLIPS templates in the above examples can be viewed as simple objects in this sense). However, we have yet to investigate the possibilities of mining conventional object-oriented code for ontological information.

3 Characterising Knowledge Sources in a Distributed Environment

Reuse of both problem solving components and knowledge sources is a holy grail of knowledge engineering. While there has been considerable discussion of re-use of problem solving algorithms in Knowledge Engineering [6] and in Software Engineering [7], there has been much less work on reuse of knowledge bases/sources. But if a company has spent a great deal of time and resource in developing a knowledge base for, say, the design of an engine, it would seem prudent, if it were possible, to use that same knowledge as the basis for a diagnostic knowledge base. In general one could imagine designers making such requests over the internet/company intranet:

> "I am looking for a knowledge base which discusses the design specification of machines for grape harvesting."

In general, these requests can be characterised as "Require knowledge base on topic T" or, more likely, "Require knowledge base on topic T where the knowledge conforms to certain constraints C". The ability to respond to a request of this form would be an important step towards creating the sort of environment in which the re-use of knowledge components is a commonplace.

We plan to address such knowledge base characterisation issues as follows: Firstly, we will decide what is the principal topic, T, of a given knowledge base. Secondly we will develop a series of other programs (or *filters* if one uses a LARKS-like nomenclature [8]) to look for different kinds of structure/constraints in the knowledge base. Each of these is dealt with briefly below.

3.1 Knowledge Base Topic Identification

Our current EXTRACTexP system can analyse a Prolog knowledge base, and can extract all the predicates (and their arities) which it contains (see top window of the tool in Figure 1). Using knowledge of Prolog, its basic constructs like `read`, `write`, etc. are discarded, leaving a set of domain terms. These terms could then be propagated through a predefined ontology (*c.f.* the spreading activation through a Semantic Network which was postulated in the '70s as a possible model to explain human focus of attention change [9]). This ontology would contain relevant concepts within the universe of discourse.

As a simple example, suppose we have the ontology depicted in Figure 2. If the concepts `Apples` and `Pears` were passed to the system, it would suggest that `Fruit` might be the relevant focus of the knowledge base. Similarly, providing the set `{Apples, Pears, Potatoes, Carrots}` would suggest that `Fruit-Vegetables` might be the focus, and if one provided `{Apples, Potatoes, Chicken, Game}` it would suggest `Food` might be the focus. We plan subsequently to extend the system so that it will be able to detect two or more principal topics, *e.g.* `Fruit` and `Food Processing`, drawn from a number of complementary ontologies.

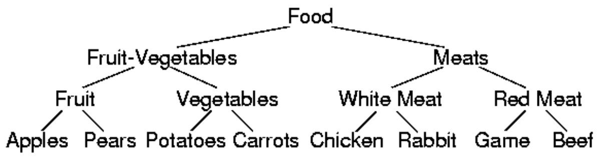


Fig. 2. Example classification ontology

3.2 Knowledge Base Structure Filters

Filters which detect the structure of a knowledge source might:

- Constrain knowledge sources such that a high percentage of their knowledge elements contain entities from both ontologies. So using the example from the last point, an appropriate goal for filtering might be "$> P$% of elements would contain elements from the Fruit/Food ontology and the Food Processing ontologies".
- Require that elements of the knowledge base be strongly related. Earlier in the COCKATOO system we demonstrated that we could acquire knowledge bases/data sets which were essentially consistent with an Extended BNF grammar [10]. Here, with the 'essentials' of the required knowledge expressed through such a grammar, rather than using this approach to acquire a knowledge base conforming to that grammar, it can instead be used to check whether existing knowledge resources display an acceptable degree of coherence with respect to the grammar. To enable such an approach, it is likely that the elements of the knowledge source would need to be marked up in XML or some comparable notation. As an illustration, below we give a section of such an EBNF grammar we used in the earlier work to describe rock formations [10]:

```
formation      → <lithology>+
lithology      → (<rock> <lithology-depth>
                              [<lithology-length>])
rock           → (<rock-type> <rock-hardness>)
rock-type      → (shale | clay | chalk | granite
                                         | other)
rock-hardness  → (very-soft | soft | medium | hard
                                     | very-hard)
```

4 Knowledge Services and Brokering

In work reported elsewhere ([11]), we have been pursuing parallel research into brokering mechanisms for knowledge resources; the purpose of this section is to give a brief overview of this work and to indicate how it relates to the knowledge services described above which are the principal focus of this paper.

If the potential of the internet as a provider of knowledge-based services is to be fully realised, there would seem to be a need for automated brokering mechanisms that are able to match a customer's knowledge requirements to appropriate knowledge providers. One of the fundamental difficulties encountered when considering how to enable this sort of transaction lies in the 'semantic mismatch' between customer and provider: how should a provider advertise its services and a customer pose its queries so that advertisement and query can be matched by the broker, and the transaction successfully completed?

One possible solution to this problem, as a number of researchers into such agent-based architectures have realised (for example, see [12,13,8]), lies in the use of ontological knowledge. Since a well-built ontology can be seen as a conceptual 'language' expressing what is essential about a domain, and uses terms that are common to that discipline, it offers some basis for enabling communication between customer and provider. However, while there may be a large number of existing knowledge resources, not all are accompanied by explicit, machine-processable ontologies; unless some alternative approach were available, any potential gains to be made through the re-use of these resources would have to be offset against the effort involved in 'reverse-engineering' their ontologies manually. The ontology extraction tool described above in Section 2 offers one such alternative approach, by which an ontology can be constructed (semi-) automatically, thus facilitating and encouraging the reuse of knowledge.

As we conceive it, then, for the purposes of advertising its capabilities to a broker, a knowledge resource describes itself using the term:

$$k\_resource(Name, Ontology, CompetenceSet)$$

where:

- *Name* is the unique identifier of this resource;
- *Ontology* is the ontology to which the resource adheres, and by which its services can be understood, and;
- *CompetenceSet* is a set of the services, or *competences* that the resource provides and which it is making available through the broker. Each item in this set is of the form $competence(C, In, Out, G_e)$ where:
  - C is a term of the form $G \leftarrow P$, where G is a goal which is satisfiable by the resource, given the satisfaction of the conditions P.
  - In is a set of constraints placed on variables in C which must hold before the competence can be utilised (successfully).
  - Out is a set of constraints placed on variables in C which hold after the competence has been applied.
  - G_e is a set of competence goals that are known to be necessary for the successful discharge of this competence and that must be supplied by some external agent.

As should be evident, the manner in which a resource advertises its services has a major impact on the effectiveness and extent of the brokering that can be performed. We find that, although relatively concise, the above information is rich enough to allow the broker to configure complex and detailed responses to the requests it receives. When successful, these responses are in the form of one or more brokerage structures, each describing a sequence of steps invoking the available competences of knowledge resources, which, when executed in order, should achieve the target.

Without going into too much detail about the construction of these sequences, an incoming request for service, in the form of a goal described in terms of some ontology in the system,[4] is matched against available competence-goals; the sets In, Out and G_e place additional constraints on any matches. These constraints take the form of either an ontological check of some item, or else of an additional goal that must be satisfied by the system, in which case the broker is invoked recursively. Of particular interest here is the notion of *bridges* in the system; a bridge (which will usually be constructed manually) allows terms (and thus, competences) described according to one ontology to be described according to a second ontology.[5] Bridges are a powerful concept for extending the range of the knowledge and capabilities of any system; however, they can only be defined if the ontology of a knowledge resource is made explicit.

[4] Currently, it is assumed that the ontologies used to describe services are available to all. Furthermore, in this discussion, we ignore all issues of access privileges, service costs, resource management and so on that are pertinent to systems of this sort.

[5] The use of bridges here is analogous to the use of bridges in UPML[6].

4.1 Ontology Extraction and the Broker

It can be seen, then, that ontologies are fundamental to any approach to brokering of this sort: they enable queries to be posed to appropriate brokers, and semantic checks to be made and bridges to be built. Unfortunately, it is not realistic to expect every potential knowledge resource to be equipped with its ontology; but nor is it desirable to simply ignore those without ontologies, given the intrinsic value of knowledge resources. In this context, EXTRACTexP offers a means by which resources lacking ontological definitions can be made accessible to brokers.

4.2 Knowledge Base Characterisation and the Broker

While this should lead to more flexible and intelligent environment, the language available for expressing queries to the broker is still relatively impoverished, and perhaps not best suited to the sort of queries that will arise in a knowledge-centred system. In particular, while a certain knowledge resource may conform to a particular ontology and satisfy stated goals consistent with that ontology, this gives little indication of the range of the knowledge base, and the structure of the inferences it can make. To address this problem, we can call upon the knowledge base characterisation services outlined above in Section 3.

Consider the case where a query to the broker is now in the form of a request for a knowledge resource that addresses a topic T, and which conforms to a set of constraints C. The technique outlined in Section 3.1 allows a description of the topic that the resource addresses to be extracted. This description is in terms of some pre-defined ontologies that could be supplied by the querying agent. Alternatively (and perhaps more appropriately) these ontologies could be managed by the broker itself, along with any known characterisations of available knowledge sources. The topics of potential candidate resources known to the environment could be extracted by the tool (again acting as a knowledge-service provider in this environment) at the instigation of the broker.

Assuming that a number of knowledge resources are found that cover the desired topic area, the next step would be to apply the constraint set C to these candidate resources, through the invocation (in a similar fashion) of the appropriate filters. The result of this would be to locate those resources (if any) that match the initial query. While providing no guarantee that these will fulfil the querying agent's needs, it would seem to offer an approach that goes beyond the simple syntactic matching often adopted, and a move towards richer, semantic modes of transaction.

5 Related Work

The aim of this section is to summarise the related work in the fields of ontology extraction and knowledge base characterisation and, as a result, set the knowledge services described in this paper in their proper context.

5.1 Ontology Extraction

In recent years there has been an increasing awareness of the potential value of ontologies — an awareness accompanied by a growing realisation of the effort required to develop them manually. As a consequence, there has been a certain amount of research into techniques by which ontological knowledge might be extracted from existing promising sources in which it is considered to be implicit. The aim of this section is to summarise this research, and its relationship with the ontology extraction tool described in preceding sections.

One related research area in which there has been a lot of interest, probably due to the amount of available source material, is that of ontology extraction from natural language texts. Typically, this involves identifying within a text certain linguistic or grammatical cues or patterns that suggest a certain ontological relationship between the concepts instantiating that pattern (for examples see [14,15,16]). Some researchers have attempted to increase the inferential power of these techniques by invoking machine learning algorithms to try to generalise the relationships that are found [17,18]. Thus far, the successes of these text-centred approaches have been limited, with unresolved questions surrounding the extent of the background knowledge that is required for such techniques (which often try to extend an existing ontology), the amount of linguistic processing of the texts that is necessary, and, indeed, the extent and range of the ontological knowledge that it is possible to infer from texts.

Similarities can also be found to the discipline of data mining, the application of machine learning and statistical learners to large databases. As for texts, the vast numbers of data often held by organisations — and the desire to exploit these — make this an appealing approach. Applications of data mining are focused not only upon extracting ontological information, but also upon finding more ‘actionable’ knowledge implicit in the data. However, the limiting factor is often the data themselves: there is no guarantee that these contain any useful knowledge of any sort, but rather they are merely a collection of arbitrary or inconclusive facts. Indeed, it is often the case that the sole outcome of a data mining exercise is a confirmation of the limitations of the data in question.

The work reported here has certain parallels with the work of the software reverse-engineering community, whose members are concerned with the extraction of information from legacy software systems. There is a relationship with the *concept assignment problem* [19], the (often very difficult) task of relating program terms and constructs to the real-world entities with which they correspond. Some techniques which attempt to extract ontological knowledge from code, and which give, perhaps unsurprisingly, often mixed results, have emerged from this discipline [20,21].

However, while the EXTRACTexP tool undoubtedly has similar intentions and shares certain concerns with the work outlined above, it is distinguished from them by the choice of an existing knowledge base as the source of ontological knowledge. In some respects, it is surprising that hitherto there has been little research into the possibilities for extracting ontologies from such sources. In constructing a knowledge base, its developers make conscious decisions to express knowledge at a conceptual level. Consequently, it would seem to be a more immediate and more fertile ground for ontological extraction than text, data or conventional code.

5.2 Knowledge Base Characterisation

The characterisation of knowledge bases (and, more generally, knowledge-based systems) has been a concern of AI research for many years, principally for the purposes of construction and analysis. As one example, the KADS [22] methodology involves characterising the knowledge surrounding a particular task — and hence, the knowledge base to address that task — at *task*, *inference* and *domain* levels according to its nature and content, and the role that it plays in problem-solving. This sort of characterisation can promote the re-use of knowledge components (of, for example, problem-solving methods at the inference level). More recently, projects such as that to develop UPML [6] have extended some of these ideas with the express purpose of modelling distributed environments of knowledge-based components.

Here, we are interested specifically in characterising a knowledge base from the perspective of the potential re-user, and the nature of the requests for knowledge that are made. However, a feature essential to the work reported here is that, if it is to be of use, this characterisation should be (at least) semi-automatic; there has been little work published regarding this aspect, and, as such, we believe that there is a contribution to be made in this area.

6 Conclusions

The success of initiatives such as the semantic web effort will be increased if existing resources can be brought within its compass without the need for extensive re-engineering. Indeed, this might even be thought a necessary feature if these initiatives are to gain the widespread support that they require to succeed. This paper has introduced two techniques that, in a relatively simple, low-cost manner, extract latent information from knowledge bases, namely implicit ontological constraints and characterisation information. This information is of the sort that enables and facilitates the future reuse and transformation of these knowledge bases within distributed environments and, as a consequence, serves to increase the scope and potential of those environments.

Acknowledgements

This work is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

1. M. Crubezy, W. Lu, E. Motta, and M. A. Musen. The internet reasoning service: delivering configurable problem-solving components to web users. In *Proc. Workshop on Interactive Tools for Knowledge Capture at the First International Conference on Knowledge Capture (K-CAP 2001), Victoria, Canada*, pp. 15–22, 2001.

2. M.F. Lopez, A. Gomez-Perez, and M.D. Rojas-Amaya. Ontology's crossed life cycle. *Proc. EKAW-2000 Conference, Juan-les-Pins, France*, Springer, pp. 65–79, 2000.
3. G. Lei, D. Sleeman, and A. Preece. N MARKUP: a system which supports text extraction and the development of associated ontologies. Technical Report, Computing Science Department, University of Aberdeen, UK (in preparation).
4. K.L. Clark. Negation as failure. In H. Gallaire, and J. Minker (eds.), *Logic and Databases*, pp.293–322. Plenum Press, 1978.
5. Y. Kalfoglou and D. Robertson. Use of formal ontologies to support error checking in specifications. In *Proc. 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW-99), Germany*, pages 207–221. Springer Verlag (Lecture Notes in Computer Science 1621), 1999.
6. D. Fensel, V.R. Benjamins, E. Motta and B. Wielinga. UPML: a framework for knowledge system reuse. In *Proc. International Joint Conference on AI (IJCAI-99), Stockholm, Sweden, July 31–August 5, 1999*, Morgan Kaufmann, pp. 16–23, 1999.
7. D.J. Reifer. *Practical Software Reuse*. John Wiley, New York, 1997.
8. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. In *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*, A. Ouksel, A. Sheth (Eds.), **28**(1), March 1999, pp. 47–53, 1999.
9. A.M. Collins, and E.F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, **82**, pp. 407–428, 1975.
10. S. White, and D. Sleeman. A grammar-driven knowledge acquisition tool that incorporates constraint propagation. In *Proc. First Int Conf on Knowledge Capture (KCAP-01), October 21–23, Victoria, Canada*, ACM Press, pp. 187–193, 2001.
11. W. M. Schorlemmer, S. Potter, D. Robertson, and D. Sleeman. Formal knowledge management in distributed environments. In Workshop on Knowledge Transformation for the Semantic Web, 15th European Conference on Artificial Intelligence *ECAI-2002*, Lyon, France, 2002.
12. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross and V.S. Subrahmanian. IMPACT: interactive Maryland platform for agents collaborating together, *IEEE Intelligent Systems magazine*, **14**(2), pp. 64–72, 2000.
13. M. Nodine, and A. Unruh. Facilitating open communication in agent systems. In *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, M. Singh, A. Rao and M. Wooldridge (Eds.), pp. 281–296. Springer-Verlag (Lecture Notes in AI V. 1365), 1998.
14. P.R. Bowden, P. Halstead, and T.G. Rose. Extracting conceptual knowledge from text using explicit relation markers. In N. Shadbolt, K. O'Hara, and G. Schreiber (eds.), *Proc. Ninth European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996*, Springer-Verlag, Berlin, pp. 147–162, 1996.
15. D. Faure, and C. Nédellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system ASIUM. In D. Fensel, R. Studer (eds.), *Knowledge Acquisition, Modeling and Management, Proc. Eleventh European Workshop, EKAW '99, Dagstuhl Castle, Germany, May 26-29, 1999* Lecture Notes in Computer Science, Vol. 1621, Springer, Berlin. pp. 329–334, 1999.
16. U. Hahn, M. Klenner, and K. Schnattinger. Automated knowledge acquisition meets metareasoning: incremental quality assessment of concept hypotheses during text understanding. In N. Shadbolt, K. O'Hara, and G. Schreiber, (eds.), *Proc. Ninth European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996*, Springer-Verlag, Berlin, pp. 131–146, 1996.
17. A. Mädche, and S. Staab. Discovering conceptual relations from text. In W. Horn (ed.), *Proc. Fourteenth European Conference on Artificial Intelligence (ECAI 2000), August 20-25 2000, Berlin, Germany*, IOS Press, Amsterdam, pp. 321–325, 2000.
18. U. Reimer. Automatic acquisition of terminological knowledge from texts. In L. C. Aiello (ed.), *Proc. Ninth European Conference on Artificial Intelligence (ECAI-90), Stockholm, August 6-10, 1990*, Pitman, London, pp. 547–549, 1990.
19. T.J. Biggerstaff, B.G. Mitbander, and D.E. Webster, Program understanding and the concept assignment problem," *Comm. ACM*, **37**(5), pp. 72–83, 1994.
20. Y. Li, H. Yang, and W. Chu. Clarity guided belief revision for domain knowledge recovery in legacy systems. In *Proc. 12th International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, USA*, Springer, 2000.
21. H. Yang, Z. Cui, and P. O'Brien. Extracting ontologies from legacy systems for understanding and re-engineering. In *Proc. IEEE 23rd International Conference on Computer Software and Applications (COMPSAC '99), October 1999*, IEEE Press, 1999.
22. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N.R. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management*, MIT Press, 2000.

The ‘Family of Languages’ Approach to Semantic Interoperability

Jérôme Euzenat[1] and Heiner Stuckenschmidt[2]

[1] INRIA Rhône-Alpes,
655 avenue de l'Europe, 38330 Montbonnot Saint-Martin, France
`Jerome.Euzenat@inrialpes.fr`
[2] Vrije Universiteit Amsterdam,
AI Department, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
`heiner@cs.vu.nl`

Abstract. Exchanging knowledge via the web might lead to the use of different representation languages because different applications could take advantage of this knowledge. In order to function properly, the interoperability of these languages must be established on a semantic ground (i.e., based on the models of the representations). Several solutions can be used for ensuring this interoperability.
We present a new approach based on a set of knowledge representation languages partially ordered with regard to the transformability from one language to another by preserving a particular property. The advantages of the family of languages approach are the opportunity to choose the language in which a representation will be imported and the possibility to compose the transformations available between the members of the family. For the same set of languages, there can be several structures depending on the property used for structuring the family. We focus here on semantic properties of different strength that allow us to perform practicable but well founded transformations.

1 Motivation

The World Wide Web is the largest information system ever. Its size and heterogeneity makes ontology-based search and integration even more important than in other information systems. The "semantic web" [5] is supported by the annotation of web pages, containing informal knowledge as we know it now, with formal knowledge. These documents can reference each other and depend on ontologies and background knowledge. Taking advantage of the semantic web requires to be able to gather, compare, transform and compose these annotations. For several reasons (legacy knowledge, ease of use, heterogeneity of devices and adaptability, timelessness), it is unlikely that this formal knowledge will be encoded in the very same language. The interoperability of formal knowledge languages must then be studied in order to interpret the knowledge acquired through the semantic web. The problem of comparing languages is well known from the field of formal logic, but it takes a greater importance in the context of the semantic web.

We refer to the problem of comparing and interpreting the annotations at the semantic level, i.e., to ascribe to each imported piece of knowledge the correct interpretation, or set of models, as *semantic interoperability*. It will be further characterized below. There are several reasons to non interoperability and several approaches to semantic interoperability [17,8,19] using different techniques. In this paper, the emphasis is on the mismatch between knowledge representation languages, leaving aside other important problems (e.g., axiomatization mismatches).

Consider a company developing applications involving printer maintenance that is neither a printer specialist nor a technical support specialist, it might have great interest in taking advantage of readily available and acknowledged ontologies. There is not a printer support ontology available so the company will have to merge different knowledge sources. Fortunately, the library of DAML (DARPA Agent Markup Language) contains an ontology describing a technical support application (http://www.daml.org/ontologies/69) and a printer ontology can be found at http://www.ontoknowledge.org/oil/case-studies/. However, the first ontology is encoded in DAML-ONT [18] and the second one in the OIL language [12].

The company wants to merge both representations for its own business but it also wants to check the consistency of the result. It thus requires an integration process through transformations that preserve the consequences and a path from that representation to a consistency checker that preserves consistency (so that, if the target representation is found inconsistent, then the source representation was too).

We discuss an approach that helps achieving semantic interoperability through a structured set of knowledge representation languages for which the properties of transformations from one language to another are known. The transformation of representations from one language to another (e.g., the initial languages in which the ontologies were formulated to the language used by the consistency checker) can take advantage of these characterized transformations in the family, minimizing the effort.

This paper first contrasts the family of languages approach with other known approaches (§2). It then puts forth several structures for a family of languages based on different properties (§3). We show that all these properties concur to semantic interoperability. Then, we show what concrete implementation of this approach can be realized (§4).

2 Approaches to language interoperability

We first give a few definitions of the kind of languages that will be considered in this paper. Then, several approaches for importing from one language to another are presented.

2.1 Languages

For the simple purpose of the present paper, a language L will be a set of expressions. A representation (r) is a set of expressions in L.

However, a language can be generated from a set of atomic terms and a set of constructors. A knowledge representations language mainly consists of operators that can be used to form complex terms (or formulas or classes) from simple ones.

For the sake of concreteness, this paper will take advantage of the results obtained in the field of description logics to illustrate the family of languages approach. This does not mean that the approach only applies to description logics, it can be applied as well to first-order logic [9] or conceptual graphs [3]. In the following we give an abstract definition of such a language:

Example 1 (Abstract description language [2]). An abstract description language L is the set of L-expressions δ, over a set T of atomic terms (name of atomic classes) and a set F of operators, where L-expressions are recursively defined as follows:

– every $t \in T$ is a L-expression
– if δ is a L-expression, then $\neg t$ is also a L-expression
– it δ_1 and δ_2 are L-expressions, then $\delta_1 \wedge \delta_2$ and $\delta_1 \vee \delta_2$ are L-expressions
– if $f \in F_L$ in an n-ary operator and $\delta_1, \cdots, \delta_n$ are L-expressions then $f(\delta_1, \cdots, \delta_n)$ is a L-expression

Note that the set T of atomic terms is independent of a specific language.

The concepts in an ontology can be intentionally described by L-expressions. Knowledge representation formalisms are subject to a well-known trade-off between expressiveness of representation and complexity of reasoning [16]. This trade-off leads to a situation that different formalisms are suited for different application scenarios. This also holds for ontology language: there is not one language that fits all situations.

Several approaches have been proposed for ensuring semantic interoperability. We present them under the standpoint of the transformation ($\tau : 2^L \longrightarrow 2^{L'}$) from one knowledge representation language (L) to another (L').

2.2 The Mapping Approach

The most direct and often used approach maps certain types of expressions in the source language and create corresponding expressions in the target language. The formal nature of these mappings vary from purely syntactic matches to “theory interpretations” [9] with well defined properties. Therefore we characterize the mapping approach solely by the existence of a function that maps expressions from one language to another.

(1) $$\exists \tau, (\forall \delta \subseteq L, \tau(\delta) \subseteq L')$$

This approach has the drawback of requiring transformations from any language to any other. It is thus not very reusable and requires to check individually the properties of the transformations. The existence of a transformation τ from L to L' is denoted by by $L \prec L'$. A current example of the mapping approach is described in [7].

2.3 The Pivot Approach

In order to reduce the number of transformations necessary to integrate a certain number of languages, a special transformation architecture can be used. One of the most common is the use of a single pivot language P all other languages are translated to. In order to be able to preserve semantics, this pivot language has to cover all other languages. More formally, the pivot approach is characterized by the following assumption:

(2) $$\exists ! P, \forall L, (L \prec P)$$

Probably the most prominent example of a pivot architecture is Ontolingua [13]. In this approach the Ontolingua language serves as a pivot language. However, translations are also performed from Ontolingua into less expressive languages leading to a loss of information the approach has often been criticized for.

2.4 The Layered Approach

A third approach to deal with semantic interoperability is the use of a layered architecture containing languages with increasing expressiveness. This approach has been proposed in order to avoid the problems arising from the need of using a very expressive language and to ensure tractable reasoning with the integrated languages. In such a layered architecture, representations can be translated into languages higher in the hierarchy without semantic mismatch. Formally speaking, the languages form a total order induced by the coverage relation.

(3) $$\forall i, j, (i \leq j \Rightarrow L_i \prec L_j)$$

A recent example of a layered architecture is the ontology language OIL [12] that has been built onto existing web standards. The idea is to use the W3C Standard RDF Schema as the language on the lowest layer and build additional language features on top of it. Doing this, it is possible to translate RDF schema definitions into languages of the higher levels in order to enrich it.

2.5 The Family of Languages Approach

The family of languages approach, presented in this paper, considers a set of languages structured by a partial order ($\prec$). This is more general than a total order, difficult to choose a priori, and more convenient for the users who can find languages closer to their needs (or, for an intermediate language, languages closer to their own languages).

For every two languages in the family a third language should exist that covers both of them.

(4) $$\forall L, L', \exists L'', (L \prec L'' \wedge L' \prec L'')$$

This equation is different from equation 3 because L'' is dependent on L and L'. In fact, the family of languages approach is a generalization of the pivot and the layered approach that further increases the flexibility of the transformation process.

Consequence. *The family of languages property generalizes the pivot and the layered approach to language integration, i.e.,* $(2) \Rightarrow (4)$ *and* $(3) \Rightarrow (4)$.

The advantage of this approach are the ability to choose an entry (resp. exit) point into the family that is close to the input (resp. output) language. This enables the use of existing results on the family of languages for finding the best path from one language to another (at least by not choosing a very general pivot language). This path can be found with the help of the coverage relation, i.e. by finding some least upper language.

The approach generalizes the pivot approach insofar as the pivot approach fulfills the family of languages property, because the pivot language P can always be used as integration language. It also generalizes the layered approach, because in the layered framework the language that is higher in the hierarchy can be used as the integration language in the sense of the family of languages property. However, the family of languages approach is more flexible, because it does not require a fixed pivot language nor a fixed layering of language. On the contrary, any language that fulfills certain formal criteria can be used as integration language. We discuss these formal criteria in the following section.

3 The Semantic Structure of a Family

A family of languages is a set $\mathcal{L}$ of languages. The goal of the family is to provide an organization that allows to transform one representation from one language of the family to another. We thus use the notion of a transformation $\tau : 2^L \longrightarrow 2^{L'}$ from one representation into another as the basis of the structure of the family. It will then be easier to use this structure in transformations. The structure of a familly of language is given by ordering this set with regard to available transformations satisfying some constraints (with the covering order $\prec$).

In order to provide a meaningful definition of this ordering, we investigate orders based on the semantics of the languages as provided by model theory. In this framework, an interpretation I is a predicate over the set of assertions of a language. Naturally, this interpretation can be defined by structural rules such as those used for defining first-order logic interpretations or description logics.

A model of a representation $r \subseteq L$, is an interpretation I satisfying all the assertions in r. The set of all models of a representation r of L is denoted by $\mathcal{M}_L(r)$. An expression δ is said to be a consequence of a set of expression r if it is satisfied by all models of r (this is noted $r \models_L \delta$). The considerations below apply to first-order semantics but they can be extended.

The languages of a family $\mathcal{L}$ are interpreted homogeneously. This means that the constraints that apply to the definition of the interpretations are the same across languages of the family (and thus, if languages share constructs, like $\vee$, $\neg$, $\wedge$, they are interpreted in the same way across languages). We generally consider languages defined by a grammar with an interpretation function defined by induction over the structure of formulas (like description logics, first order logic or conceptual graphs). In this case, the homogeneity is provided by having only one interpretation rule per formula constructor.

Again, this can be illustrated in the description logics framework.

Example 2 (Abstract Description Model [2]). An Abstract description model is of the form:

$$\Im = \langle W, F^{\Im} = (f_i^{\Im})_{i \in I} \rangle$$

where W is a nonempty set and $f_i^{\Im}$ are functions mapping every sequence $\langle X_1, \cdots, X_{n_i} \rangle$ of subsets of W to a subset of W.

We can define the interpretation mapping in two steps. First we assume an assignment $\mathcal{A}$ mapping every $t \in T$ to a subset of W, then we define the interpretation mapping recursively as follows:

Example 3 (Semantics [2]). Let L be a language and $\Im = \langle W, F^{\Im} \rangle$ an abstract description model. An *assignment* $\mathcal{A}$ is a mapping from the set of atomic term T to 2^W. The assignment of a subset of W to a term t is denoted by $t^{\mathcal{A}}$. The extension $\delta^{\Im,\mathcal{A}}$ of a L-expression is now defined by:

1. $t^{\Im,\mathcal{A}} := t^{\mathcal{A}}$ for every $t \in T$
2. $(\neg\delta)^{\Im,\mathcal{A}} := W - \delta^{\Im,\mathcal{A}}$
3. $(\delta_1 \wedge \delta_2)^{\Im,\mathcal{A}} := \delta_1^{\Im,\mathcal{A}} \cap \delta_2^{\Im,\mathcal{A}}$
4. $(\delta_1 \vee \delta_2)^{\Im,\mathcal{A}} := \delta_1^{\Im,\mathcal{A}} \cup \delta_2^{\Im,\mathcal{A}}$
5. $f(\delta_1, \cdots, \delta_n)^{\Im,\mathcal{A}} := f^{\Im}(\delta_1^{\Im,\mathcal{A}}, \cdots, \delta_n^{\Im,\mathcal{A}})$ for every $f \in F$

The semantics definition given above is the basis for deciding whether an expression δ is satisfiable and whether an expression δ_1 follows from another expression δ_2. More specifically, the L-expression δ is satisfiable if $\delta^{\Im,\mathcal{A}} \neq \emptyset$, an L-expression δ_1 is implied by δ_2 if $\delta_1^{\Im,\mathcal{A}} \subseteq \delta_2^{\Im,\mathcal{A}}$. The definition is general enough to capture description logics as well as modal and first-order predicate logic.

This section will provide tools for defining the structure of a family of languages. It will focus on a semantic structure that is prone to provide semantic interoperability. The structure is given by the coverage relation ($\prec$ above) that can be established between two languages when there exists a transformation from one to the other. In this section, the coverage relation will be characterized in function of a property that it satisfies. The ultimate goal of these properties are to ensure the possible preservation of the consequences while transforming from a language to another.

3.1 Language inclusion

The simplest transformation is the transformation from a language to another syntactically more expressive one (i.e., which adds new constructors).

Definition 1 (Language inclusion). *A language L is included in another language L' iff $\forall \delta \in L, \delta \in L'$.*

The transformation is then trivial: it is thus identity. This trivial interpretation of semantic interoperability is one strength of the "family of languages" approach because, in the present situation, nothing have to be done for gathering knowledge. This first property provides a first relation for structuring a family:

Definition 2 (Language-based Coverage).

$$L\breve{\preceq}L' \Leftrightarrow_{def} (L \subseteq L')$$

Language inclusion can be defined in a more specific way on languages defined as a term algebra where the inclusion of languages can be reduced to the inclusion of the sets of term constructors.

Example 4 (The FaCT Reasoner). The FaCT description logic reasoner implements two reasoning modules one for the language $\mathcal{SHF}$ and one for the language $\mathcal{SHIQ}$ which simply extends $\mathcal{SHF}$ with inverse roles and qualified number restrictions. As a consequence, $\mathcal{SHF}$ models can be handled by the $\mathcal{SHIQ}$ reasoner without change.

3.2 Interpretation preservation

The previous proposal is restricted in the sense that it only allows, in the target language, expressions expressible in the source language, while there are equivalent non-syntactically comparable languages. This is the case of the description logic languages $\mathcal{ALC}$ and $\mathcal{ALUE}$ which are known to be equivalent while none has all the constructors of the other[3]. This can be described as the equality of the Tarskian style interpretation for all the expressions of the language.

Definition 3 (Interpretation preservation). *A transformation τ preserves the interpretations iff*

$$\forall \delta \in L, \forall I, I(\tau(\delta)) = I(\delta)$$

Example 5 (Reasoning in Core-OIL). The lowest layer of the ontology language OIL which has gained significant attention in connection with the semantic web is Core -OIL which provides a formal semantics for a part of RDF schema. In order to provide reasoning services, the language is translated into the logic $\mathcal{SHIQ}$ and the FaCT Reasoner is used to provide the reasoning services [14]. Core-OIL can contain

assertions restricting the applicability of a particular role ($R \leq (\texttt{domain}\, C)$. These assertions must be expressed in $\mathcal{SHIQ}$ which does not offer the domain constructor. It is thus translated into an assertion stating that for any term under $\top$, the range of the inverse of this relation is this particular domain. The translation contains the following interpretation-preserving mapping[4]:

$$\tau(R \leq (\texttt{domain}\, C)) = \top \leq (\texttt{all}\, (\texttt{inv}\, R)\, C)$$

For that purpose, one can define $L\overline{\overline{\ll}}L'$ if and only if there exists a transformation from L to L' that preserves the interpretations of the expressions.

[3] This is true if we consider that the languages here are those described by their names: $\mathcal{AL}$+negation vs. $\mathcal{AL}$+disjunction+qualified existentials. Of course, because they have the same expressivity all the constructors of each language can be defined in the other. But this equivalence must be proved first.

[4] This is not sufficient for eliminating all occurrences of domain. For instance, `(all (domain C) C′)` has to be transformed into `(or (not C) (all anyrelation C′))`. This does not work for *concrete domains* either.

Definition 4 (Interpretation-based coverage).

$$L\overline{\overline{\ll}}L' \Leftrightarrow_{def}$$
$$\exists \textit{ an interpretation preserving tranformation } \overline{\overline{\tau}} : L \rightarrow L'$$

Obviously, language inclusion is stronger than interpretation preservation because the languages are homogeneous and the transformation is then reduced to identity.

Proposition 1 (Language-based coverage entails interpretation-based coverage). *If $L'\breve{\preceq}L$ then $L'\overline{\overline{\ll}}L$.*

The $\overline{\overline{\tau}}$ transformation is, in general, not easy to produce (and it can generally be computationally expensive) but we show, in [11], how this could be practically achieved.

3.3 Expressiveness

The previous property was subordinated to the coincidence of interpretation. In particular, the domain of interpretation has to be the same and the way entities are interpreted must coincide.

Franz Baader [1] has provided a definition of expressiveness of a first-order knowledge representation language into another by considering that a language can be expressed into another if there exists a way to transform any theory of the first into a theory of the second which preserves models up to predicate renaming.

His definitions is based on the idea of "abstract models" in which a language is a couple made of a language L and a model selection function Mod_L which filters the acceptable models for the language (which are not all the first order models). Here, we consider as acceptable all the first-order models.

Definition 5 (Expressibility modulo renaming [1]). *A language L is expressible in a language L' if and only if $\forall r \in L$, $\exists$ a transformation $\tau : L \rightarrow L'$, $\exists \nu : Pred(r) \rightarrow Pred(\tau(r))$ such that $\forall m \in M_L(r), \exists m' \in M_{L'}(\tau(r)); \forall \delta \in L, m(\delta) = m'(\nu(\delta))$ and $\forall m' \in M_{L'}(\tau(r)), \exists m \in M_L(r); \forall \delta \in L, m(\delta) = m'(\nu(\delta))$. $Pred(r)$ is the set of atomic terms T found in the expression r.*

Example 6 (Eliminating undefined concepts axioms in $\mathcal{TF}$). Bernhard Nebel has shown that the transformation from a T-Box with the introduction of undefined (primitive) concepts can be translated into T-box with additional concepts (primitive component concepts). So, each undefined concept $\dot{\leq}$, is introduced by a definition $\doteq$ as the conjunction (`and`) of its known subsumers and an undefined part (expressed with an overline here):

$$\tau(Man\dot{\leq}Human) = Man\doteq(\texttt{and}\; Human\; \overline{Man})$$

This transformation preserves expressiveness [1].

We do not want to consider renaming here (it involves knowing what to rename and using the $Pred$ function which denotes the set of predicates used in an expression). So, expressibility is refined by simply using the transformation $\widehat{\tau}$ instead of ν.

Definition 6 (Expressibility modulo transformation). *A language L is expressible in a language L' if and only if $\forall r \in L$, $\exists$ a transformation $\widehat{\tau} : L \rightarrow L'$, such that $\forall m \in M_L(r), \exists m' \in M_{L'}(\widehat{\tau}(r)); \forall \delta \in L, m(\delta) = m'(\widehat{\tau}(\delta))$ and $\forall m' \in M_{L'}(\widehat{\tau}(r)), \exists m \in M_L(r); \forall \delta \in L, m(\delta) = m'(\widehat{\tau}(\delta))$*

Naturally, expressibility modulo transformation entails expressibility modulo renaming.

Definition 7 (Expressibility-based coverage).

$L\widehat{\lll}L' \Leftrightarrow_{def}$ L is expressible (modulo transformation) in L'

The following proposition is easily obtained by noting that a interpretation-preserving transformation is also a model-preserving transformation. So the corresponding model, can be the model itself (or an extension of itself to formulas missing from the initial language).

Proposition 2 (Interpretation-based coverage entails expressivity-based coverage). *If $L\overline{\overline{\lll}}L'$, then $L\widehat{\lll}L'$.*

3.4 Epimorphic transformations

Full isomorphism between the models of a representation and its transformations is prone to preserve a major part of the meaning. However, an isomorphism would constrain the two sets of models to have the same cardinality. This is relatively artificial. We relax this constraint by asking each model of the transformed representation to be closely related to one model of the source representation. This can be useful when one does want to consider axiomatizations of different natures. This can be used when objects are taken as relations and vice versa (dual representation of graphs is an example).

Definition 8 (Model epimorphism). *A model epimorphism $\pi : M \rightarrow M'$ is a surjective map from a set of model M to another set of models M'.*

Model epimorphisms ensure that all models of the transformed representation are comparable to some model of the source representation.

Definition 9 (Epimorphic transformation). *A transformation τ is epimorphic iff there exists a model epimorphism $\pi : \mathcal{M}_{L'}(\tau(r)) \rightarrow \mathcal{M}_L(r)$ such that $\forall r \subseteq L, \forall m' \in \mathcal{M}_{L'}(\tau(r))$ and $\forall \delta \in L, \pi(m') \models \delta \Rightarrow m' \models \tau(\delta)$*

This kind of transformation allows the generated representation to have many more very different models than the initial representation, but constraint each of these models to preserve all the consequences of one of the models of the initial representation.

Definition 10 (Correspondance-based coverage).

$L\widetilde{\lll}L' \Leftrightarrow_{def} \exists$ an epimorphic transformation $\widetilde{\tau} : L \rightarrow L'$

This basically ensures that the transformation does not loose information (i.e., does not generate unrelated models). The following proposition is obtained by building the epimosphism from the corresponding models in the second equation of definition 6.

Proposition 3 (Expressibility-based coverage entails correspondance-based coverage). *If $L\widehat{\lll}L'$, then $L\widetilde{\lll}L'$.*

3.5 Consequence preservation

Consequence preservation can be considered the ultimate goal of semantic interoperability: it denotes the fact that the consequences (what are satisfied by all models) of the source and the target representations are the same (modulo transformation).

Definition 11 (Consequence preservation). *A transformation τ is said consequence-preserving iff $\forall r \subseteq L, \forall \delta \in L, r \models_L \delta \Rightarrow \tau(r) \models_{L'} \tau(\delta)$*

If τ is a consequence-preserving transformation, then for any $r \subseteq L$, it is said that $\tau(r)$ is a conservative extension of r modulo τ.

Example 7 (Translating from $\mathcal{DLR}$ to $\mathcal{SHIF}$). In order to decide query containement in $\mathcal{DLR}$, [6] displays a mapping from the $\mathcal{DLR}$ logic (which introduces n-ary relations) to $\mathcal{CPDL}$. If one considers the restriction introduced in [15] where $\mathcal{DLR}$ does not contain regular path expressions. These relations are represented by concepts with exactly n features to the components of the relation. This transformation is a consequence preserving transformation.

This definition allows to define a coverage based on consequence as usual:

Definition 12 (Consequence-based coverage).

*$L\overline{\lll}L' \Leftrightarrow_{def}$
$\exists$ a consequence preserving transformation $\overline{\tau} : L \rightarrow L'$*

Model-based coverage is stronger than consequence-based because it already included the notion of consequence-preservation. The point is that there can be “more” models in L' than in L, but they satisfy the same assertions as one model in L, they thus cannot inhibit any consequence.

Proposition 4 (Correspondance-based coverage entails consequence-based coverage). *If $L\widetilde{\lll}L'$, then $L\overline{\lll}L'$.*

It is known that expressivity modulo renaming alone does not necessarily entail consequence preservation [1].

3.6 Consistency preservation

Preserving consistency is a very weak property (it is true of any transformation that forgets knowledge). However, transformations that preserve consistency can be used for checking the inconsistency of a knowledge base: if the target knowledge base is inconsistent, then the source was too.

Definition 13 (Consistency preservation). *A transformation τ is said to be consistency-preserving iff $\forall r \subseteq L, \mathcal{M}_L(r) \neq \emptyset \Rightarrow \mathcal{M}_{L'}(\tau(r)) \neq \emptyset$*

Example 8 (Reasoning in Standard-OIL). The second layer of the OIL language called standard OIL provides an expressive language for building ontologies. Again, the language is translated into $\mathcal{SHIQ}$ in order to provide inference services. Standard OIL also includes capabilities for expressing assertional knowledge and instances in concept definitions. As the FaCT reasoner does not support instance reasoning, the translation from Standard OIL to $\mathcal{SHIQ}$ includes some mappings that do not preserve the complete semantics, but preserve satisfiability [14].

$$\tau((\texttt{one}-\texttt{of}\; i_1\; i_2)) = (\texttt{or}\; I_1\; I_2)$$

This transformation replaces the enumeration of instances by a disjunction of concepts with the same name.

Consistency-based coverage is defined as usual.

Definition 14 (Consistency-based coverage).

$L\dot{\leq}L' \Leftrightarrow_{def}$
$\exists$ *a consistency-preserving transformation* $\dot{\tau} : L \to L'$

Proposition 5 (Expressivity-based coverage entails consistency-based coverage). *If* $L\widehat{\ll}L'$*, then* $L\dot{\leq}L'$*.*

3.7 Composition of properties

As a consequence, all the coverage relations concur to providing the families of language with a structure which enriches the basic syntactic structure usually proposed for these languages.

This defines a hierarchy of more and more constrained structure for the family of language. Establishing this structure can be more or less easy, so it is useful to be able to have several of them that can be used only if necessary. This permits to have the best effort in looking for a path from one language of the family to another.

There can be other useful properties (and thus other structures) that anyone can integrate in the structure of a family. These properties do not have to be totally ordered from the strongest to the weakest. However, for being useful to semantic interoperability, new properties should entail some of the properties above.

These structures enable the composition of transformations while knowing their properties. The following table provides the minimal property satisfied by the composition of two transformations given their properties.

	$\breve{\leq}$	$\overline{\overline{\ll}}$	$\widehat{\ll}$	$\dot{\leq}$	$\widetilde{\ll}$	$\overline{\ll}$
$\breve{\leq}$	$\breve{\leq}$	$\overline{\overline{\ll}}$	$\widehat{\ll}$	$\dot{\leq}$	$\widetilde{\ll}$	$\overline{\ll}$
$\overline{\overline{\ll}}$	$\overline{\overline{\ll}}$	$\overline{\overline{\ll}}$	$\widehat{\ll}$	$\dot{\leq}$	$\widetilde{\ll}$	$\overline{\ll}$
$\widehat{\ll}$	$\widehat{\ll}$	$\widehat{\ll}$	$\widehat{\ll}$	$\dot{\leq}$	$\widetilde{\ll}$	$\overline{\ll}$
$\dot{\leq}$	$\dot{\leq}$	$\dot{\leq}$	$\dot{\leq}$	$\dot{\leq}$	$\emptyset$	$\emptyset$
$\widetilde{\ll}$	$\widetilde{\ll}$	$\widetilde{\ll}$	$\widetilde{\ll}$	$\emptyset$	$\widetilde{\ll}$	$\overline{\ll}$
$\overline{\ll}$	$\overline{\ll}$	$\overline{\ll}$	$\overline{\ll}$	$\emptyset$	$\overline{\ll}$	$\overline{\ll}$

In summary, the semantic structure of a family of languages provides us with different criteria for coverages all based on the notion of transformability. These notions of coverage do not only give us the possibility to identify and prove coverage, they also specify a mechanisms for transforming the covered into the covering language. Therefore we can show that a suitable language can be generated and how the generation is being performed. In the next section we present an instanciation of this approach.

4 Implementing the Approach

The family of language approach can take advantage of many knowledge representation formalisms that have been designed in a modular way. A concrete example of a family is presented below through an example (§4.2) using the DLML encoding of description logics supplied with transformations (§4.1).

4.1 A concrete family of languages

DLML [10] is a modular system of document type descriptions (DTD) encoding the syntax of many description logics in XML. It takes advantage of the modular design of description logics by describing individual constructors separately. The specification of a particular logic is achieved by declaring the set of possible constructors and the logic's DTD is automatically build up by just assembling those of elementary constructors. The actual system contains the description of more than 40 constructors and 25 logics. To DLML is a associated a set of transformations (written in XSLT) allowing to convert a representation from a logic to another.

The first application is the import and export of terminologies from a DL system. The FaCT system [4] has already developed that aspect by using such an encoding. We also developed, for the purpose of the examples presented here, the transformations from OIL and DAML-ONT to DLML. These transformation are simple XSLT stylesheets.

4.2 Example

The company which needs a printer support ontology has to merge different knowledge sources the technical support application ontology in DAML-ONT and the printer ontology written in the OIL language [12]. It also wants to translate the merged ontology into the $\mathcal{SHIQ}$ language in order to check consistency of the result. The transformation must be consistency preserving.

The translation methodology, from one language to another, consists in choosing the input and output languages within the family. The source representation will be translated in the input language and the target representation will be imported from the output language. The input languages are obviously DLML counterparts of OIL and DAML-ONT and the translation is easily carried out because both language have been inspired by description logics. The target language will be the DLML language corresponding to $\mathcal{SHIQ}$, supported by the FaCT reasoner.

Then, a path from the input language to the output language which satisfies required properties has to be found in the family of languages used. This path is presented below.

The first goal will be achieved by translating the DAML-ONT and OIL representations in a representation language (called G) which encompasses all the constructs of the initial languages. The transformations depend only on the language inclusion property between the two input languages and G.

The second goal will be achieved by composing three DLML transformations that rewrite some representations with a particular construct to representations without it, suitable to be checked for consistency by the FaCT reasoner. This implements transformations already at work in the OIL-based tools

[14]. It thus chain the following transformations (summarized by figure 1):

domain2allinv which replaces `domain` restrictions on role definitions by a general constraint applying to the restricted terms (through the restriction of the inverse role codomain): this transformation is interpretation-preserving;
oneof2ornot which replaces enumerated domains (`oneof`) by disjunctive concepts whose disjuncts represents the elements of this domain: this transformation is only consistency preserving;
cexcl2not which replaces concept exclusion (introduced by the previous transformation) by conjunction with the negated concept. This transformation is also interpretation preserving.

$$\begin{tikzcd}
DAML \arrow[r] & L_{DAML} \arrow[dr, "\check{\leq}"'] & \texttt{merge} & L_{OIL} \arrow[dl, "\check{\leq}"] & OIL \arrow[l] \\
& & G = L_{DAML} \check{\vee} L_{OIL} \arrow[d, "\texttt{domain2allinv}"', "\overline{\overline{\lll}}"] & & \\
& & L_{int} \arrow[d, "\texttt{oneof2ornot}"', "\dot{\leq}"] & & \\
& & L'_{int} \arrow[d, "\texttt{cexcl2not}"', "\overline{\overline{\lll}}"] & & \\
& & L_{\mathcal{SHIQ}} \arrow[rr] & & \mathcal{SHIQ}
\end{tikzcd}$$

Fig. 1. The transformation flow involved in importing the two ontologies to $\mathcal{SHIQ}$.

Thus the import of OIL and DAML-ONT into $\mathcal{SHIQ}$ described above is consistency preserving.

5 Conclusion

The ’family of languages’ approach is one approach for facilitating the exchange of formally expressed knowledge in a characterized way. It is not exclusive to other approaches like direct translation or pivot approaches. It has several advantages over other solutions to the semantic interoperability problem because it allows users:

- to translate to closer languages among many of them;
- to share and compose many simple transformations for which the property are known and the transformations available;
- to select the transformations to be used with regard to the kind of properties that are required by the transformation.

This approach is thus a tool for better ‘ontology engineering’.

The approach generalizes previous proposals for translation architectures and provides a greater flexibility in terms of languages that can be used for the integrated models. We have presented here this approach in a unified framework and proposed a first tower of structure for the family of languages based on the properties that are satisfied by the transformations. Different semantic relations can be used to establish the structure of a family of languages and ensure formal properties of transformations between languages. We concluded with a brief description of an existing implementation of the approach.

The approach can easily be implemented using existing web technologies such as XML and XSLT, but also provides an infrastructure for ensuring formal properties by proving the formal properties of transformations between concrete languages. It is even possible to annotate transformations with these proof and use them for justifying or explaining the transformations.

Acknowledgements

This work has benefited from the support of the OntoWeb thematic network (IST-2000-29243) and the PAI Procope program of the French Ministry of Foreign Affairs and DAAD.

References

1. Franz Baader. A formal definition of the expresive power of terminological knowledge representation languages. *Journal of logic and computation*, 6(1):33–54, 1996.
2. Franz Baader, Carsten Lutz, Holger Sturm, and Frank Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research*, 16:1–58, 2002.
3. Jean-François Baget and Marie-Laure Mugnier. The $\mathcal{SG}$ family: extensions of simple conceptual graphs. In *Proc. 17th IJCAI, Seattle (WA US)*, pages 205–210, 2001.
4. Sean Bechhofer, Ian Horrocks, Peter Patel-Schneider, and Sergio Tessaris. A proposal for a description logic interface. In *Proc. int. workshop on description logics, Linkping (SE)*, number CEUR-WS-22, 1999. http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-22/bechhofer.ps.
5. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 279(5):35–43, 2001. http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.
6. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS’98)*, pages 149–158, 1998.
7. Hans Chalupsky. OntoMorph: a translation system for symbolic knowledge. In *Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US)*, pages 471–482, 2000.
8. Mihai Ciocoiu and Dana Nau. Ontology-based semantics. In *Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US)*, pages 539–546, 2000. http://www.cs.umd.edu/ nau/papers/KR-2000.pdf.
9. Herbert Enderton. *A mathematical introduction to logic*. Academic press, 1972. revised 2001.
10. Jérôme Euzenat. Preserving modularity in XML encoding of description logics. In Deborah McGuinness, Peter Patel-Schneider, Carole Goble, and Ralph Möller, editors, *Proc. 14th workshop on description logics (DL), Stanford (CA US)*, pages 20–29, 2001.

11. Jérôme Euzenat. Towards a principled approach to semantic interoperability. In Asuncion Gomez Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, *Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US)*, pages 19–25, 2001.
12. Dieter Fensel, Ian Horrocks, Frank Van Harmelen, Stefan Decker, Michael Erdmann, and Michel Klein. Oil in a nutshell. In *12th International Conference on Knowledge Engineering and Knowledge Management EKAW 2000*, Juan-les-Pins, France, 2000.
13. Thomas Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisitio*, 5(2):199–220, 1993.
14. Ian Horrocks. A denotational semantics for Standard OIL and Instance OIL, november 2000. http://www.ontoknowledge.org/oil/downl/semantics.pdf.
15. Ian Horrocks, Ulrike Sattler, Sergio Tessaries, and Stephan Tobies. Query containment using a $\mathcal{DLR}$ abox - preliminary version. LCTS Report 99-15, RWTH Aachen, 1999.
16. Hector Levesque and Ronald Brachmann. *Readings in Knowledge Representation*, chapter A Fundamental Tradeoff in Knowledge Representation and Reasoning (Revised Version), pages 31–40. Morgan Kaufmann Publishers, San Mateo, 1985.
17. Claudio Masolo. *Criteri di confronto e costruzione di teorie assiomatiche per la rappresentazione della conoscenza: ontologie dello spazio e del tempo*. Tesi di dottorato, Universit di Padova, Padova (IT), 2000.
18. Deborah McGuinness, Richard Fikes, Lynn Andrea Stein, and Hendler James. DAML-ONT: An ontology language for the semantic web. In Dieter Fensel, James Hendler, Henri Lieberman, and Wolfgang Walhster, editors, *The semantic web: why, what, and how?* The MIT press, 2002. to appear.
19. Heiner Stuckenschmidt and Ubbo Visser. Semantic translation based on approximate re-classification. In *Proceedings of the KR workshop on semantic approximation granularity and vagueness, Breckenridge, (CO US)*, 2000. http://www.tzi.de/ heiner/public/ApproximateReClassification.ps.gz.

A Logic Programming Approach To RDF Document And Query Transformation

Joachim Peer[1]

Institute for Media and Communications Management,
University of St.Gallen, Switzerland
joachim.peer@unisg.ch

Abstract. The Resource Description Framework (RDF) provides a means to explicitly express semantics of data. This makes it an attractive tool for managing any kind of data, especially if multiple heterogenous vocabularies are involved. However, there exists neither a standard way of transforming RDF documents nor of transforming RDF queries yet. Several concepts have been proposed so far, each concentrating on different goals. This paper presents a concept following a Logic Programming approach, providing the amount of expressivity needed to build generic services for management of RDF data. In this paper we will focus on several transformation problems not satisfactory discussed yet and we will provide an overview of the algorithms and data structures needed to cope with these problems.

1 Introduction

Technical advancements and the success of the Internet and the World Wide Web in the last years lead to an explosion of the amount of data available. Today, large parts of the economic and social world directly depend on the availability and processing of this data. However, the usefulness of data can be greatly enhanced if it is described in an explicit, machine interpretable manner. A first step towards such a machine interpretable data format was provided by the W3C with the eXtensible Markup Language (XML), which has already changed the way electronic business is conducted today. However, XML basically only provides a means for parser-reuse and a means for machines to automatically check syntactical and structural document correctness. Although standardised XML DTDs and XML Schemas partially provide semantics for certain defined XML elements, the language concept of XML itself does not provide any sufficient means to allow the semantics of a document be expressed explicitly. Therefore, the W3C developed the Resource Description Framework (RDF) [1], which is from ground up designed to be capable of explicitly expressing semantics of data. RDF does so by incorporating a very generic model of knowledge representation: It allows to formulate *statements* by assigning *properties* combined with property *values* to *resources*, whereby a property value may be either a literal or another resource.

An important cornerstone of our work is that RDF can be seen as a language of First Order Predicate Logic, with resources being subjects, properties being predicates and property values either being subjects or constants. Direct quotation of other statements is - just as in First Order Predicate Calculus - not permitted. Instead, a syntactical tool called *reification* is provided for this purpose. Therefore, desirable computational properties (e.g. efficiency, tractability) are generally preserved for RDF, in opposition to Higher Order-Predicate Calculi. RDF contains a reserved property *rdf:type* which provides a standardised way of typing resources. Typing of resources provides a basic means to distinguish between schema definitions and instance data which makes use of that schema definition. Several additional constructs (like property-range and -domain restrictions, subClass- and subProperty-relationships) were introduced in the RDF Schema specification [2], which is used to define schemas for RDF instance documents. As discussed above, RDF is suitable to present any kind of information in any complexity, without necessarily giving up computational tractability. Another useful characteristic of RDF is that it is - in opposition to XML - syntax agnostic. As described in [3] and [4] XML provides an arbitrary big number of possible valid XML trees to represent *one* logical model. RDF on the other hand, provides exactly one tree for one logical model. Thus, RDF can be seen as a means to reduce ambiguity and complexity of models to a minimum. Obviously, this characteristics of RDF is very useful for any task of schema integration, because in general the complexity of any transformation scheme tends to form a polynomial function of the complexity of the source and the destination schema. Therefore, the reduction of complexity of both sides can lead to more efficient transformations.

However, XML is much more in use than RDF today, especially in business-to-business applications. For transformation between XML files mainly XML based techniques like XSLT [5] are currently used to map XML files directly into each other, leading to complex transformation and preparation tasks. As a solution to this problem a multi-layered approach to ontology[1] integration was proposed in [7]. It distinguishes three separate layers of information: (1) the syntax layer, which presents the XML surface syntax of data, (2) the data model layer, which represents the logical model of the document and may be represented using RDF, (3) the ontology layer, which relates elements from the data model layer to establish (domain-) specific conceptualisations. According to this approach, XML data can be transformed to RDF, which in turn can be more easily processed and transformed by machines. Applying this framework to our work, we can use both XML for exchange with legacy systems and RDF for intelligent processing and querying of data.

This paper is organised as follows: in section 2 we present an overview of selected RDF transformation approaches and concepts, and we position our approach in relation to these

[1] This paper uses the definition provided by Gruber [6], who defines an ontology as a *formalisation of a conceptualisation*.

concepts. In section 3 we then present our concept of RDF and RDF query transformations and we briefly describe the context of its application. In section 4 we present the process model of our concept, concrete algorithms for transforming RDF documents and RDF queries. We conclude the paper with a short summary of findings and an outline of future work.

2 Related Work

Currently there exists no standard way of transforming RDF-encoded documents or RDF queries, although we can identify several approaches with different characteristics:

2.1 DAML+OIL sameAs Properties

The DAML+OIL language [8] provides a means to link ontologies together in a very Web-like fashion. It defines a property *equivalentTo* which can be assigned to any resource, stating that the resource is semantically equivalent to another resource specified. Derived from that basic property three specialised properties *sameClassAs*, *samePropertyAs* and *sameIndividualAs* are defined, providing the ability to associate classes with classes, properties with properties and instances with instances. This kind of mapping definition works well for integrating ontologies with minor structural differences. Especially synonym relations can be marked up and resolved in an inexpensive way. Another advantage of this concept is that queries can be automatically propagated to the related ontologies: Let us consider an ontology A, which contains several sameAs-properties referring to constructs of ontology B. If an agent submits a query using constructs of vocabulary A to a semantic search engine and if the semantic search engine is aware of the DAML+OIL based definitions of A and B, it should be able to expand the query to the equivalent concepts of Vocabulary B. The drawback of the sameAs-properties of DAML+OIL is that complex relationships between properties and classes need to be expressed using class expressions like *Restriction*-elements, which can be used in DAML+OIL ontologies only. Therefore, a full fledged DAML+OIL reasoner is needed to resolve such relationships.

2.2 XSLT Inspired Approach: RDFPath/RDFT

Given the fact that RDF can be expressed using an XML based notation called RDF/XML [1] and given the fact that the XML world provides specifications with production ready implementations for transforming XML documents - namely XPath [9] and XSLT - it is tempting to use XPath and XSLT to perform transformations on RDF/XML documents. However, given the fact, that there exist several different variants of RDF/XML (i.e. abbreviated serialisations) it turns out that using pure XPath/XSLT for transformations is a tedious and improper approach.

An approach trying to overcome this problem and still being modelled very closely after the XML transformation tools XPath and XSLT is the RDFPath/RDFT approach presented in [10]. Just as in the XML-specific versions of these languages, RDFPath is used to localise information in an RDF model and to specify the path between two nodes in a model. RDFT is, similar to XSLT, used to relate a source document with a destination document, using the expressions defined using RDFPath. The advantage of using RDFPath/RDFT over XPath/XSLT is that it abstracts from the actual RDF/XML serialisation syntax, making RDF-to-RDF transformations possible independent from the serialisation format used. However, RDFT still focuses more on structure than on the meaning of RDF documents. Characteristically, it does not allow custom rules to be defined, making it difficult to resolve transitive relationships defined in custom RDF schemata. Although RDFT is therefore not well suited for complex RDF-RDF transformations, it may provide a good concept for mapping XML to RDF and vice versa.

2.3 Performance Optimised Approach: RDF-T

A concept explicitly designed for the task of RDF transformation is RDF-T [11]. It provides constructs (called *Bridges*) to define relations between RDF schema classes (using *ClassBridges*) and RDF property definitions (using *PropertyBridges*) respectively. To define the characteristics of an association drawn by a certain Bridge RDF-T provides two properties *rdft:Relation* and *rdft:Condition*. Equivalence relations, which we are particularly interested in, are represented by Bridges with a property ”Relation” of value ”Map” and with a property ”Condition” containing the values ”Necessary” and ”Sufficient”. Both ClassBrigdes and PropertyBridges are capable of representing one-to-one, one-to-many and many-to-many relationships.

This goal can also be achieved using the sameAs-properties of DAML+OIL presented above: To specify 1:n mappings using sameAs-properties DAML+OIL class expressions like *daml:Union* can be used. However, as already mentioned above, this would require the full machinery of a DAML+OIL reasoner.

The philosophy behind RDF-T is a different one: One of the core requirements of RDF-T is performance and it is reasonable that specialised constructs as provided by RDF-T can be processed much faster than general multi-purpose constructs as represented by DAML+OIL. However, a potential disadvantage of RDF-T is that it permits only homogeneous mappings between classes and classes and between properties and properties. Cross-mappings between classes and properties (or vice versa) are not permitted. For certain areas and for certain controlled vocabularies this may be a valid assumption and the performance advantage gained by this simplification will justify the loss of mapping expressiveness in many cases. However, in a Semantic Web context this simplification can not be uphold. Though it is true that RDF provides exactly one representation of one interpretation (model) of a world, we must consider that there are, of course, several *different* models of that world which can follow very different modelling paradigms.

2.4 KR-Transformation Approach

An approach with very high expressiveness is represented by OntoMorph [12]. OntoMorph is designed to transform instance data between different knowledge bases. OntoMorph allows to transfer meaning even between knowledge bases using different logical and modelling paradigms, and using different notations. OntoMorph achieves the transformation using so-called *rewrite rules*. It defines two levels of transformation - the syntactical and the semantical level - and for each of these two levels it provides a special kind of rewrite rules: *Syntactical* rewrite rules are used to transfrom different sentences from one knowledge respresentation (KR) - syntax to another; syntactical rewrite rules have the form *pattern* ⇒ *result*, whereby pattern and result are Lisp-style representations of the parse trees of terms to be translated. Because syntactical rewrite rules provide no means for inference, OntoMorph also provides *semantical* rewrite rules: On the left side of a semantical rewrite rule is a PowerLoom-expression (which is a typed variant of a KIF expression) which determines and selects certain instances from the knowledge base. On the right side of the semantical rewrite rule is another PowerLoom expression for reclassifying the translated instances.

Although adapting OntoMorph to the domain of RDF transformations would be conceivable, we think that doing so may give away many opportunities for simplifications: Because our transformation approach is restricted to a single metamodel - the RDF model - we are able to dispense some of the heavy weight expressive features of OntoMorph, including the necessity of Full First Order inference systems like PowerLoom.

2.5 Logic Programming Approach

An approach closely related to the RDF data model is presented in [13], which facilitates a Logic Programming[2] approach.

The concept presented in [13] presents a meta-data architecture which distinguishes between three levels of data: instance level data (e.g. RDF triples, Topic Map Instances, XML elements), schema level data (e.g. RDF- or Topic Map Schemas, XML DTDs), and model constructs (RDF model, Topic Map model, XML model). The idea behind that concept is that the definition of a common - RDF based - metamodel on top of the data model, similarly to the metamodelling concepts presented in [14] and [15] may provide a means of universal translation between these models. This claim is based on the premise that each level of the architecture can be viewed as an instantiation of the levels above it. More specifically, model constructs are seen as particular instantiations of the abstractions defined by the metamodel, schema-level data as particular instantiations of the model's schema constructs, and instance-level data are interpreted as instantiations of the model's instance constructs. However, since the metamodel described in [13] is expressed using RDF, the findings of this paper are interesting to us independently of the metamodel-concept itself. From our perspective it is irrelevant if the RDF triples to be transformed represent instance-, schema-, metaschema-data or other superimposed information.

A transformation between two documents is defined by a mapping definition M: M consists of a set of mapping rules T ⇒ T', where T and T' are sets of S predicates (definition given below). The rule can be read as follows: the left-hand side S-predicates must be true in order to generate the right-hand side S-predicates (i.e., the right-hand side S-predicates are *produced* by the rule). S, also called a *production rule*, is a predicate of the form S(L, t) that is true if t ∈ L, which is a database of triples for a layer of information. Finally, the predicate t(subject, predicate, object) presents an RDF triple. As briefly sketched in the introduction section, RDF can be interpreted as an application of First Order Predicate Calculus. Thus, the approach presented in [13] promises to fit the characteristics of RDF very well. However, some problems we are concerned with in RDF transformation are not treated by this concept: One of these problems is the mapping of RDF graphs with major structural differences, which poses certain problems not discussed yet. Another open issue not satisfyingly treated yet is the transformation of queries.

[2] This paper uses the classical definition of this term, primarily referring to Horn Logic based languages like Prolog and Datalog.

2.6 Our Position

We presented several approaches for translating RDF information. Promising approaches include RDF-T, which emphasises performance rather than expressive power. At the other end of the spectrum is the Full First Order Logic-based approach presented by OntoMorph. Our approach is positioned in the range between these two concepts, building on the findings presented in [13]. Additionally to the concept described in [13] we focus on several questions important to us, including the correct introduction and assignment of anonymous nodes and the translation of queries.

3 The Proposed Concept

The concept presented serves as foundation for a part of a larger system for storage and intelligent retrieval of RDF encoded data. This system is designed to be used as an infrastructural component in distributed systems, thus proving a generic service according to [16].

Beside the ability to develop and facilitate domain specific queries for intelligent data retrieval one of the core requirements (and reasons for facilitating RDF) is the support of interoperability between multiple heterogeneous vocabularies. We identified two requirements for interoperability between vocabularies: The first requirement is the ability to transform RDF instance data from one vocabulary into another (e.g. to convert e-business documents). The second requirement is the ability to transform queries executed against an RDF triple store. The latter requirement is needed particulary if the RDF documents queried use multiple different vocabularies (e.g. RDF Schemas) and if queries should still be able to reach most of the data.

3.1 Representing RDF Data and RDF Queries

At first we need to define the objects of interest for our transformation concept and how they are represented.

Representing RDF Data As already described on several places (e.g. in [17] and [18]) RDF statements can be represented as logical expressions. However, instead of defining a triple (P,S,O) directly as predicate P(S,O), we follow the advise of [19] and define it as a ternary relation (which we will call *triple*, throughout this paper)

```
triple(Subject,Predicate,Object).
```

The reason for choosing ternary relations is that statements about properties (e.g. subProperty-relationships) would lead to unwanted higher order constructs if the statements would be expressed using binary relations.

An RDF document can therefore be represented as a set of ternary predicates *triple*. However, if the content of multiple RDF documents needs to be managed and processed, it is necessary to preserve the URI of the location the triples originally stem from. This information can be stored in a fact *data*, whereby the URI of the documents containing the triple is represented by the variable *Uri*:

```
triple(Subject,Predicate,Object) :-
  data(Uri, Subject,Predicate,Object)
```

Representing RDF Queries Several RDF Query languages and systems have been proposed following approaches like the *database approach* or the *knowledge base approach*, as identified in [20]. The database approach interprets an RDF document primarily as an XML document, which leads to several drawbacks described in [20]. For this reason, all of the recently presented RDF Query languages and systems adhere to the knowledge base approach, explicitly dealing with RDF triples instead of document trees. It is easy to see that the knowledge base approach of RDF querying fits the abilities of Logic Programming well. Queries can be expressed as (Horn-) rules, which may be asserted to and executed by any Logic Programming based engine.

The semantics of RDF query languages like RQL [21], RDFDB-QL [22], SQUISH [23] or Sesame [24] can be expressed using Horn rules, even if the concrete implementations use imperative languages instead of Logic Programming. In fact, these implementations can be seen as specialised versions of Logic Programming algorithms like SLD resolution, which are (per se) imperative as well. Therefore we suggest that the representation of RDF queries as Horn rules is a very natural one. Several alternative logic based notations to express RDF queries have been proposed, including Frame-logic based notation as presented in [25]. However, even these notations can be translated back to Horn rules as shown in [25]. Using Horn rules to represent RDF queries leads to a query format as follows:

```
query(arguments) :-
    rulehead1(arguments),
    ...
    ruleheadN(arguments).
...
query(arguments) :-
    rulehead1(arguments),
    ...
    ruleheadM(arguments).
```

Each query may consist of a set of conjunctions as depicted above. Each conjunction consists of a set of ruleheads containing arguements (atoms and variables).

Representing Rules Formulating RDF statements and queries in the way described above opens the door for Logic Programming-based processing of that data. As shown in [26] and [25], rules may be defined that operate upon these facts. In general we can distinguish two toplevel types of rules for our concept:

– *System rules*, which are needed to provide basic predicates like *triple*.
– *Individual rules*, which build on the set of system rules.

We can distinguish several types of individual rules: They may be defined to derive information according to semantics defined by formalisms like RDF Schema or DAML+OIL. For instance, a *subClassOf* rule may be defined to determine the transitive closure of the subClassOf-relationsship provided by RDF Schema, (further examples for such rules are presented in [27]) Another type of rules may encapsulate *domain specific* knowledge. Individual rules may be used just as system rules in both queries and mapping definitions. The requirements and restrictions for using individual rules in mapping definitions are discussed in section 4.3.

3.2 Transformation Rules

We follow the approach presented by [13] and specify a transformation schema as a set of rules. At first glance, this may also look similar to the concept of rewrite rules as presented in [12]. However, the focus on a specific modelling paradigm and data model (i.e. RDF) allows us to propose several modifications and simplifications. First, we do not need to distinguish between syntactic and semantical rewrite rules. We can offer a single integrated rewrite method, which transforms one RDF graph into another, operating on both the syntactical and the semantical level, whereby the distinctions between these two levels is unimportant in this context. Second, because both source and destination documents adhere to the same metamodel (i.e. RDF), we deal with translations of much less inherent complexity. This allows us to formulate mappings not only as unidirectional implications (denoted as '$\Rightarrow$'), but also as equivalences (denoted as '$\Leftrightarrow$').

A specification of a transformation schema T consists of:

– a disjunction of n equivalences. Each equivalence E consists of two conjunctions of rule-heads, interconnected by an equivalence operator $\Leftrightarrow$. Each rule-head can take any number of arguments. Arguments can be atomic values (literals) or logic variables.
– an optional set of facts about the relationships between variables assigned to the rule-heads. Syntax and semantics of these optional facts are described in section 4.4.

To illustrate this concept, we will provide an example for a tranformation of e-business documents. We will use this example throughout this paper for illustration purposes. The example makes use of two ontologies c and d, derived from XML based e-commerce standards. The RDF schema definitions of these ontologies are available online at http://elektra.mcm.unisg.ch/wm.

One of the differences between the ontologies c and d is that they provide different concepts for the term "supplier". Ontology c provides an explicit class for this term, whereas d treats suppliers as an organisation with a special attribute value associated. The transformation between instance data of c and d therefore needs a mapping statement like "all Resources X which are type c:Supplier are equal to Resources

X of type d:Organization, given that a property d:Role with value ’supplier’ is assigned to that resource”. This statement can be expressed as a logical equivalence as follows:

```
type(?X,'c:Supplier')
<=>
type(?X,'d:Organization'),
triple(?X,'d:role','supplier')
```

According to the nature of logical equivalence, this transformation rule may be read in both directions. Evaluating it from left to right leads to a *branching* of nodes, in the other direction it leads to a *conflation* of nodes. Depending on the translation direction each side of the equivalence serves either as matching rule or as production rule.

Large ontologies may lead to complex mapping expressions. Therefore, we provide ”syntactical sugar” in the form of substitution rules. Consider the following example: Large parts of the complex mapping statement ”all Resources X with a property c:supplierLocation SL with a c:address A (of type c:Address) with a c:postalAddress PA (of type c:PostalAddress) with a c:street S is equivalent to a Ressource X which `c:has_address` PA (of type d:Address) with a d:street S” can be encapsulated in substitution rules (e.g. addressLeft and addressRight), thus enabling readable mappings:

```
addressLeft(?X,?SL,?A,?PA),
triple(?PA,'c:street',?S)
<=>
addressRight(?X,?SL,?A,?PA),
triple(?PA,'d:street',?S)
```

The complex substitution rules themselves may be defined once in a document and can be reused by other rules.

```
addressLeft(?X,?SL,?A,?PA) :-
  triple(?X,'c:supplierLocation',?SL),
  triple(?SL,'c:address',?A),
  type(?A,'c:Address'),
  triple(?A,'c:postalAddress',?PA),
  type(?PA,'c:PostalAddress')

addressRight(?X,?SL,?A,?PA) :-
  triple(?X,'d:has_address',?PA),
  type(?PA,'d:Address')
```

Without substitution rules, the matchings would look as follows:

```
triple(?X,'c:supplierLocation',?SL),
triple(?SL,'c:address',?A),
type(?A,'c:Address'),
triple(?A,'c:postalAddress',?PA),
type(?PA,'c:PostalAddress'),
triple(?PA,'c:street',?S)
<=>
triple(?X,'d:has_address',?PA),
type(?PA,'d:Address'),
triple(?PA,'d:street',?S)
```

4 Execution Model and Algorithms

After presenting the underlying concepts and the static structure of our proposed concept, we will now focus on the algorithmic aspects.

4.1 Transforming Ontologies

We introduce the following pseudocode representation of our algorithm to ease its discussion:

```
(1)  for each equiv. E in
     Mapping-Rulebase {
(2)    extract source rule part RPS
       from E.
(3)    extract destination part RPD
       from E.
(4)    find all solutions for RPS in
       the triple store.
(5)    for each solution S {
(6)      gather values of bound
         variables from the current
         solution.
(7)      for each rule-head R in RPD {
(8)        for each variable argument
           V in R {
(9)          if a binding for V is part
             of the solution {
(10)           unify with that value.
(11)         } else {
(12)           unify with a new id.
(13)         }
(14)       }
(15)     }
(16)     assert equivalent triple in
         destination triple store.
(17)   }
(18) }
```

The basic tranformation algorithm presented above can be described as follows. The transformation engine iterates over a set of defined equivalences (line 1). Each of the equivalences is split up in its left and its right side (lines 2, 3). Depending on the direction of the transformation process, one side of each equivalence is used as matching rule and the other side is used as production rule. For each matching rule all solutions can be found using Logic Programming resolution techniques like SLD (line 4). These solutions represent the essential information of the source document that need to be incorporated into the destination document. The algorithm iterates over all found solutions (line 5, 6). For each rulehead (line 7) and each variable argument (line 8) the current solution is searched for values of matching variables. If a value is found, then the resp. variable argument is unified with that value (lines 9, 10). Otherwise a new (random) value is created and assigned to the variable argument (lines 11, 12). In any case, the values assigned to the variable arguments of the production rule will be used to create new triples in the destination document (line 16).

If we apply this algorithm (together with the equivalences defined in section 3.2) to the following RDF fragment

```
<c:Supplier about="http://org1.com" />
```

then it will be transformed to its pendant

```
<d:Organisation about="http://org1.com">
  <d:role>supplier</d:role>
</d:Organisation>
```

However, under certain conditions this basic algorithm produces unwanted results. In section 4.4 we will analyse these conditions and present an extension to this algorithm to eliminate the unwanted behaviour.

4.2 Tranforming Queries

We suggest that a query can be seen as a special instantiation of a document schema. Therefore, most parts of the algorithm for document transformation as described above can be applied to queries.

The first step in a query transformation process is to transform the query into the same form as any other RDF document to be transformed, so that we can apply our transformation algorithm on it. We will illustrate this using the following example query:

```
q(X) :- triple(?X, 'isa', 'c:Supplier'),
        triple(?X, 'c:name', ?N),
        substring(?N, 'computer')
q(X) :- triple(?X, 'isa', 'c:Supplier'),
        triple(?X, 'c:name', ?N),
        substring(?N, 'hardware')
```

This query will be transformed to sets of ground rules, which we will call *query stores*. A query store contains all elements (conjunctions) of a disjunction of a query. If a query consists of N disjunctive parts, then N query stores will be generated. The query above consists of 2 disjunctive parts (i.e. N=2), therefore 2 query stores will be generated.

All variables in each query store are substituted by literals. After substitution, the query stores for the query above may look as follows:

```
triple('varX', 'isa', 'c:Supplier'),
triple('varX', 'c:name', 'varN'),
substring('varN','computer')
<=>
triple('varX', 'isa', 'c:Supplier'),
triple('varX, 'c:name', 'varN'),
substring('varN','hardware').
```

Now, after the logic variables are treated as simple atoms, we can apply the core algorithm provided above. All we need to do is to wrap an additional iteration around the algorithm as a whole (we introduce new lines 0 and 19). The adapted algorithm may be expressed in pseudocode as follows:

```
(0)  for each query store QS in the sets
     of querystores {
(1)    for each equiv. E in
       Mapping-Rulebase {
(2)      extract source rule part RPS
         from R.
              /** equal to
                 algorithm above **/
(18)   }
(19) }
```

In the example case above, both query stores are processed according to our algorihm, therefore two *destination conjunctions* are constructed:

```
Conj. #1:
triple('varX','type','d:Organisation'),
triple('varX','d:role','d:Supplier'),
triple('varX','d:org_name','varN'),
substring('varN','computer').

Conj. #2:
triple('varX','type','d:Organisation'),
triple('varX','d:role','d:Supplier'),
triple('varX','d:org_name','varN'),
substring(varN,'hardware').
```

After this transformation step, a simple substitution of the masked variable names (varX, varN) will finish the conversion of the query from vocabulary c to d:

```
q1'(X) :-
    triple(?X,'type','d:Organisation'),
    triple(?X,'d:role','d:Supplier'),
    triple(?X,'d:org_name', ?N),
    substring(?N, 'computer').
q1'(X) :-
    triple(?X, 'type','d:Organisation'),
    triple(?X, 'd:role','d:Supplier'),
    triple(?X, 'd:org_name', ?N),
    substring(?N, 'hardware').
```

4.3 Resolution of Production Rules

If a matching rule of a side of an equivalence evaluates to true and binds free logical variables, we suppose the system to trigger actions to produce the equivalent values on the other side. We have to distinguish three kind of production rules:

– triples
– rules that can be reduced to a single conjunction of triples
– rules that cannot be reduced to a single conjunctions of triples

The first kind of rules, i.e. triples, are very easy to handle. The algorithm only needs to substitute variable names with their concrete bindings; the same is true for the second kind of rules. However, there are rules that cannot be directly translated to conjunctions of triples. These are rules containing native rules (e.g. Prolog library predicates) or recursion. Such rules can not be used as production rules directly, because the system would not be able to determine its reciprocal semantics. Therefore it is necessary to define this semantics individually.

As an example let us consider the recursive defined rule `type` which reflects semantics of RDF Schema, namely the transitive relationship between types and their subclasses.

```
type(?I,?C) :-
    triple(?I, 'rdf:type', ?C).
type(?A, ?CSuper) :-
    subClassOf(?Super, ?CSuper),
    type(?A, ?Super).
subClassOf(?C1,?C2) :-
    triple(?C1, 'rdf:subClassOf', ?C2).
```

In the case of ontology transformation, an alternative production rule triple(I, 'rdf:type', C) may be individually assigned for the rule type(I,C). This will ensure that the knowledge about the class membership of a certain resource will be translated safely. However, in the case of query translation, this substitution is not always applicable, because in a context of a query the modified semantics of the construct can lead to unwanted consequences like too narrow or too wide search spaces. For instance, if type(I,C) is a conjunctive part of a query and if it is transformed to triple(I, 'rdf:type', C), the search space of the query is narrowed. Therefore, in the case of query transformations many rules may be transferred unmodified. This was the case for the `substring`-predicate, which was involved in the transformation example above.

4.4 Dealing with Structural Differences

In many cases ontologies to be mapped show significant structural differences that need to be captured by a proper mapping mechanism. Consider the example ontologies c and d, which model the concept of *addresses* in a very different way: Ontology c is more generic and distinguishes multiple `SupplierLocations`, which may consist of several abstract `Addresses` with multiple `PostalAddresses`. In contrast, ontology d defines the postal addresses of an organisation directly using a property `has_address`. This is illustrated in figure 1.

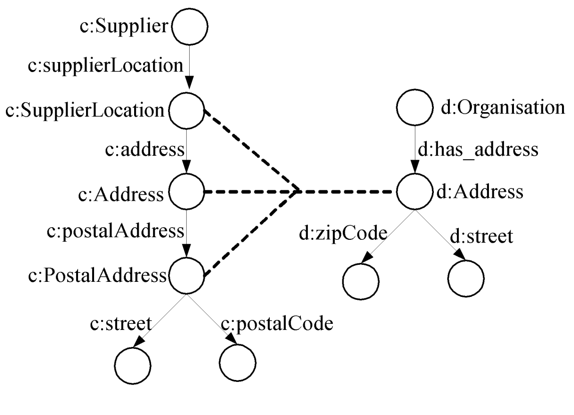


Fig. 1. Example of structural differences

If we need to map the concepts `c:street` and `c:postalCode` with their respective pendants `d:zipCode` and `d:street`, we can, based on the findings above, formulate two equivalences as follows (note that these equivalences can be written down significantly shorter using substitution rules as described above, but the notation below reflects the problem depicted in figure 1 more transparently):

```
triple(?X,'c:supplierLocation',?SL),
triple(?SL,'c:address',?A),
type(?A,'c:Address'),
triple(?A,'c:postalAddress',?PA),
type(?PA,'c:PostalAddress'),
triple(?PA,'c:street',?S)
<=>
triple(?X,'d:has_address',?PA),
type(?PA,'d:Address'),
triple(?PA,'d:street',?S)
```

```
triple(?X,'c:supplierLocation',?SL),
triple(SL,'c:address',?A),
type(A,'c:Address'),
triple(?A,'c:postalAddress',?PA),
type(?PA,'c:PostalAddress'),
triple(?PA,'c:postalCode',?Z)
<=>
triple(?X,'d:has_address',?PA),
type(?PA,'d:Address'),
triple(?PA,'d:zipCode',?Z)
```

Before we will present our solution, we will shortly discuss the problems sketched above in more detail. In our discussion we distinguish two common problems:

Problem of Branching If the equivalencies above are evaluated from left to right, i.e. from the simpler to the more complex structure, the problem arises that there are no bound values for the variables SL and A, because these variables do not occur on the right side of the equivalence. The concept described in [13] partially solves this problem by introducing a 0-ary Skolem function *guid()* that delivers a unique identifier to be unified with an unbound variable. However, this approach only works if it is admitted that for each occurrence of a production rule containing an unbound variable and its associated guid() function a new resource is generated. In many situations, including the situation above, this is not the case: A translation algorithm successfully transforming the example in figure 1 needs to *re-use* anonymous resource already generated, to preserve the semantic structure of the transformed document.

Problem of Conflation The case of conflation may look less error prone at the first glance: If the equivalences above are evaluated from left to right, i.e. from the more complex to the simpler structure, the bindings of the variable PA are used to construct the simpler structure on the right side. Because all nodes already exist, there is no problem of generating (unwanted) new anonymous ressources. However, if somewhere in the mapping definition another equivalence dealing with the address-objects exists, we run into a similar problem. Consider the following equivalence:

```
triple(?X,'c:supplierLocation',?X_SL),
triple(?X_SL,'c:address',?A),
type(?A,'c:Address'),
triple(?A,'c:phone',?P),
triple(?P,'c:name','CustomerCare'),
triple(?P,'c:value',?V)
```

```
<=>
triple(?X,'d:has_address',?A),
type(?A,'d:Address'),
triple(?A,'d:servicePhone',?V)
```

The problem is, that the above equivalence is agnostic of the entity "postalLocation". Therefore a tranformation engine would use the value of *A* to construct the object for the statement `triple(X,'d:has_address',A)` on the right side. However, the rules for translating "street" and "postalCode" objects (introduced in our examples above) use the value of the postalAddress *PA* to create the same construct. If these rules are used together in a single transformation process (which would be desirable) these conflicts may become acute, leading to multiple occurrences of `has_address` properties constructed in the resulting document, which is unwanted.

Solution Each equivalence per se is logically correct (i.e. the relationships of the variables are correctly formulated in each equivalence), but in conjunction problems arise, because the relationships between the variables across multiple equivalences are undefined. Therefore, it is necessary to introduce *additional facts* that provide meta-information about the relationships of the logic variables used in the production rules.

To solve the problem of branching for the example above, we need to define the relationships of the potentially conflicting variables. This can be achieved using a fact as follows:

```
dependence('c', ['X', 'SL', 'A', 'PA']).
```

This fact tells the translation engine that on the side *c* the variable PA depends on the variable A, which depends on the variable SL, which depends on the variable X. For the translation engine this means that, before assigning any value to one of these variables (e.g. A) it has to compare the current solutions for variables it depends on (e.g. X, SL) with solutions of other equivalencies already processed. If such a combination was found (e.g. a solution for X, SL and A, produced while processing some other equivalence) then the existing value of the dependent variable is taken in favour of using the current binding or even creating a new one. This kind of value-tracking can be implemented using data structures like trees.

Therefore we can now complete our algorithm for document- and query transformation by replacing lines 9-13 with lines 9-22 of the pseudocode fragment below:

```
(7)  for each rule-head R in RPD {
(8)    for each variable
       argument V in R {
(9)     if(V is a dependent variable) {
(10)       if solution-tree contains
           matching existing values {
(11)         unify with value found.
(12)         add solution to
             solution-tree.
(13)         continue with next
             variable.
(14)       }
(15)     }
(16)     if a binding for V is part
         of the solution {
(17)       unify with that value.
(18)     } else {
(19)       unify with a new id.
(20)     }
(21)     add solution to solution-tree.
(22)   }
(23) }
```

Each variable argument V is now tested if it is a dependent variable, by looking up the set of defined `dependence`-facts (line 9). If V is a dependent variable, a test for existing bindings is conducted (line 10). For this purpose the algorithm uses a tree-structure which holds all variable values used by production rules and which can be queried for existing bindings. If such a combination of values is found, then the corresponding value is retrieved from the tree structure and is assigned to V (line 11).

If the variable V is not a dependent variable or if no matching combination of existing variable bindings was found, then the algorithm proceeds by consulting the current solutions (lines 16-20), as already introduced by the first version of our algorithm in section 4.1.

In any case, a successful unification must be stored in the solution tree (lines 12 and 21). This ensures that the bindings are available for later iterations.

5 Summary and Outlook

RDF provides a good framework for effectively storing and retrieving data, especially if multiple heterogenous vocabularies are involved. Despite this fact, there exists neither a standard for transforming RDF documents nor for transforming RDF queries yet. However, several concepts have been proposed so far, concentrating on different goals (e.g. expressiveness, performance). Our concept follows the approach of applying Logic Programming as presented already in [13], because it provides the expressivity that we need to build generic transformation services. In this paper we focused on problems not satisfactory discussed yet, e.g. the problems arising when transforming documents with substantial structural differences.

As a proof of concept we have developed a first version of a prototype incorporating the concepts presented in this paper. It can be accessed online via http://elektra.mcm.unisg.ch/wm.

However, there are many other issues related with transforming RDF documents that are not discussed in this paper: among these problems are the correct transformation of RDF collections and reified statements. Another problem not explicitly addressed in this paper is the problem of the semantics of negation ("not") used in transformation rules.

References

1. Lassila, O., Swick, R.: Resource description framework (RDF) model and syntax specification. W3C Recommendation (1999) http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.
2. Brickley, D., Guha, R.: Resource description framework(RDF) schema specification 1.0. W3C Candidate Recommendation (2000) http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

3. Berners-Lee, T.: Why RDF model is different from the XML model. Informal note (Draft) (1998) http://www.w3.org/DesignIssues/RDF-XML.html.
4. Berners-Lee, T.: Using XML for data. Informal note (Draft) (1998) http://www.w3.org/DesignIssues/RDF-XML.html.
5. Clark, J.: XSL transformations (xslt). W3C Recommendation (1999) http://www.w3.org/TR/xslt.
6. Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition **5(2)** (1993) 199–220
7. Omelayenko, B., Fensel, D.: Scalable document integration for b2b electronic commerce. Electronic Commerce Research Journal, Special Issue (submitted) (2002)
8. Horrocks, I., van Harmelen, F., Patel-Schneider, P.: Reference description of the DAML+OIL ontology markup language. Language specification (2001) http://www.daml.org/2001/03/daml+oil-index.html.
9. Clark, J., DeRose, S.: XML Path language (xpath). W3C Recommendation (1999) http://www.w3.org/TR/xpath.
10. Kokkelink, S.: RDFPath: A path language for RDF. In preparation (2001) http://zoe.mathematik.uni-osnabrueck.de/QAT/Transform/RDFTransform/RDFTransform.html.
11. Omelayenko, B., Fensel, D., Klein, M.: RDF-T: An RDF transformation toolkit. WWW 2002 (submitted) (2002) http://www.cs.vu.nl/ borys/papers/RDFT-WWW02.pdf.
12. Chalupsky, H.: OntoMorph: A translation system for symbolic knowledge. In: Principles of Knowledge Representation and Reasoning. (2000) 471–482 citeseer.nj.nec.com/chalupsky00ontomorph.html.
13. Bowers, S., Delcambre, L.: Representing and transforming model based information. In: Proc. of the Workshop of the Semantic Web at the 4th European Conference on Research and Advanced Technology for Digital Libraries (ECDL-2000). (2000)
14. Object Management Group: Meta object facility (MOF) specification. OMB Document ad/99-09-04 (1999) http://www.omg.org/cgi-bin/doc?ad/99-09-04.
15. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language User Guide. Addison Wesley (1999)
16. Schmid, B., Lechner, U.: Communities and media - towards a reconstruction of communities on media. In Sprague, E., ed.: Hawaiian Int. Conf. on System Sciences (HICSS 2000), IEEE Press (2000)
17. Marchiori, M., Saarela, J.: Query + metadata + logic = metalog. QL'98 - The Query Languages Workshop (1998) http://www.w3.org/TandS/QL/QL98/pp/metalog.html.
18. Fikes, R., McGuinness, D.: An axiomatic semantics for RDF, RDF Schema, and DAML+OIL. KSL Technical Report KSL-01-01 (2001)
19. Hayes, P.: RDF model theory. W3C Working Draft (2002) http://www.w3.org/TR/rdf-mt/.
20. Karvounarakis, G.: RDF query languages: A state-of-the-art (1998) http://139.91.183.30:9090/RDF/publications/state.html.
21. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A declarative query language for RDF. In: The Eleventh International World Wide Web Conference (WWW'02). (2002)
22. Guha, R.: RDFDB : An RDF database (2000) http://rdfdb.sourceforge.net/.
23. Miller, L.: RDF query: Squish QL (2001) http://swordfish.rdfweb.org/rdfquery/.
24. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF schema. In Horrocks, I., Hendler, J., eds.: Proc. of the 2002 International Semantic Web Conference (ISWC 2002). Number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) To appear.
25. Decker, S., Sintek, M.: TRIPLE-an RDF query, inference, and transformation language. In Horrocks, I., Hendler, J., eds.: Proc. of the 2002 International Semantic Web Conference (ISWC 2002). Number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) To appear.
26. Conen, W., Klasping, R.: A logical interpretation of RDF. Electronic Transactions on Artificial Intelligence (ETAI) (under review) (2001)
27. Zou, Y.: DAML XSB rule version 0.3. working paper (2001) http://www.cs.umbc.edu/ yzou1/daml/damlxsbrule.pdf.

Information Retrieval System Based on Graph Matching

Takashi Miyata[1] and Kôiti Hasida[1,2]

[1] Core Research for Evolutional Science and Technology,
Japan Science and Technology Corporation
miyata.t@carc.aist.go.jp
[2] Cyber Assist Research Center,
National Institute of Advanced Industrial Science and Technology
hasida.k@aist.go.jp

1 Introduction

The recent increase of online documents demands more and more laborious work to obtain necessary information. Information retrieval (IR) systems play an important role here. An IR task fundamentally relies on matching two different representations: the query and the answer in the database. This matching requires sophisticated and complicated inferences based on background knowledge and/or context in general.

Suppose that the query contains keyword 'build.' In a context related to housing, 'build' is similar to 'construct.' In a context related to career, it would be similar to 'promote.' It is impossible to infer such contexts only from the limited input. Some kind of collaborative *interaction* with the user is indispensable here. This interaction is mediated by the semantic structures of the query and the database. Considering the semantic structures around the retrieved answer candidates, the IR system can better focus on the right answer, and the user can revise her queries efficiently.

Although IR with linguistic structure[2] and IR with interaction[1] have been separately studied by number of researchers, we believe that their combination provides valuable hints both for the user and the system.

2 Interaction in Query Revision

Suppose that the user wants to retrieve documents which report on a house built with lower cost by robots, and inputs the graph in Fig. 1. The system will return thousands of documents, which appear irrelevant. The system, however, also indicates that the input keyword 'house' is often directly related (through syntactic dependency or anaphora) with verb 'construct.' The user then recalls that 'construct' is a synonym of 'build.' She also recalls 'inexpensive' as a synonym of 'cheap.' There are still thousands of seemingly irrelevant candidates. The user notices that the agent of 'build' could be 'robot' and revises her query graph by adding a link between 'robot' and 'build.' The number of documents reported by the system remains the same but the ranking among them changes, and the user notices that the highest ranked document is the one she wants.

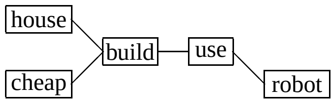


Fig. 1. Graph for "Build a House Cheaply (by) Using Robots."

Similar situations were observed during our preliminary experiment on 15 example retrieval tasks. The subject added 0.8 keywords by himself, picked up 3.73 synonyms from the thesaurus, and adopted 0.53 keywords/synonyms from the list of words often directly semantically related to input keywords. Although the synonyms/keywords of this last type are not very many, these words play an important role in query revisions to raise the rank of the correct answers. Since the thesaurus used in this experiment is of a general nature, its coverage is far from complete. Even if the thesaurus were specific to a particular domain, it could not be specialized to a particular retrieval task. Indicating such words only by a thesaurus would be impossible in practice.

3 Matching and Scoring Algorithm

A query graph can be formalized as an extension of conventional "conjunctive normal form" of keywords, $(k_{1,1} \vee \cdots \vee k_{1,n_1}) \wedge \cdots \wedge (k_{m,1} \vee \cdots \vee k_{m,n_m})$, where each $k_{i,j}$ is a keyword. We regard each conjunct as a vertex, and introduce edges among vertices. A vertex is labeled by one or more keywords.

Our matching algorithm is divided into two phases. The first phase considers only the correspondence and similarity between vertices. The similarity score of each subgraph in the database (also a graph) against the input query graph is calculated. To reduce computational complexity, we have developed an incremental tabulation-based enumeration algorithm based on Chart parsing with sophisticated scheduling[3]. The second phase adjusts the similarity score of each candidate subgraph by checking whether edges in the query graph correspond to those in the candidates. Given the correspondence among vertices, which can be calculated in $O(kN)$ time (k: # keywords, N: the maximum # documents that contain a keyword) by an inverted file, the computational complexity of match finding is $O(nr)$ (n: # vertices in a query, r: # candidates to examine). Exact ranking requires that r be $O(N^n)$ and we set r to a small value.

References

1. Roy Goldman and Jennifer Widom. Interactive query and search in semistructured databases. Technical report, Stanford University, 1998. http://www-db.stanford.edu/lore/.
2. Tomek Strzalkowski. Natural language information retrieval. *Information Processing and Management*, 31(3):397–417, 1995.
3. Henry Thompson. Best-first enumeration of paths through a lattice — an active chart parsing solution. *Computer Speech and Language*, 4:263–274, 1990.

Formal Knowledge Management in Distributed Environments*

W. Marco Schorlemmer[1], Stephen Potter[1], David Robertson[1], and Derek Sleeman[2]

[1] CISA, Division of Informatics, The University of Edinburgh
[2] Department of Computing Science, University of Aberdeen

In order to address problems stemming from the dynamic nature of distributed systems, there is a need to be able to express notions of evolution and change of knowledge components of such systems. This need becomes more pressing when one considers the potential of the Internet for distributed knowledge-based problem solving — and the pragmatic issues surrounding knowledge integrity and trust this raises. We introduce a formal calculus for describing transformations in the 'lifecycles' of knowledge components, along with suggestions about the nature of distributed environments in which the notions underpinning the calculus can be realised. The formality and level of abstraction of this language encourages the analysis of knowledge histories and allows useful properties about this knowledge to be inferred.

Formal Lifecycles in a Brokering Architecture

We take the real-life example of Ecolingua[3], an ontology for ecological meta-data. It was constructed on the Ontolingua Server[4] by reusing classes from other ontologies in the server's library, and then automatically translated into Prolog syntax by the server's translation service.

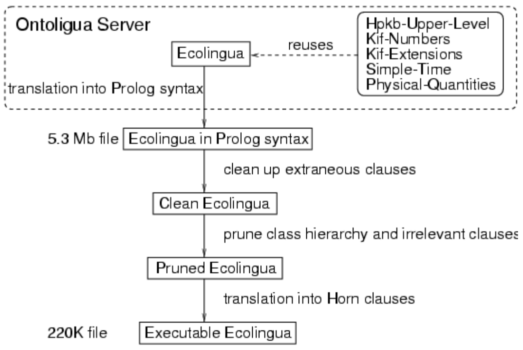
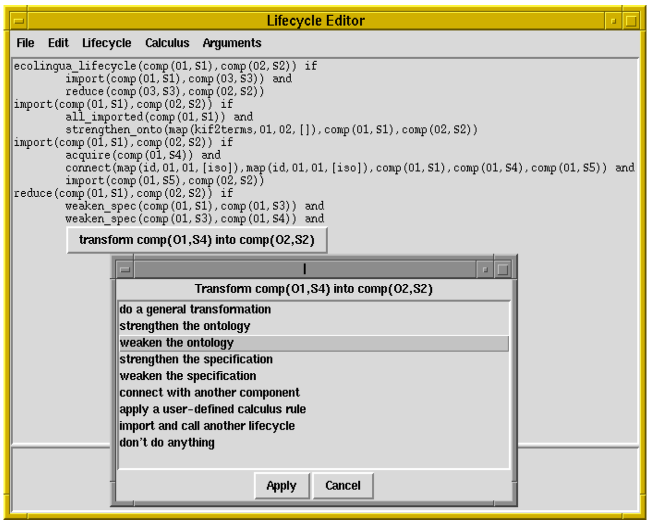


Fig. 1. Ecolingua's lifecycle

Because the outcome of the translation process was an overly large 5.3 Mb file, and in order to get a smaller and more manageable set of axioms, it was necessary to reduce the ontology, by implementing filters that first deleted all extraneous clauses, and then pruned the class hierarchy and removed irrelevant clauses accordingly. Finally, a translation into Horn clauses was performed in order to use the ontology with a Prolog interpreter (see Figure 1).

We postulate that particular sequences of lifecycle steps like those of Figure 1 might be common in particular domains and perhaps with particular forms of knowledge component. The ability to generalise and 'compile' these sequences into *lifecycle patterns* would encourage more efficient behaviour when faced with the need to make similar modifications in the future.



```
ecolingua_lifecycle(comp(01,S1),comp(02,S2)) if
        import(comp(01,S1),comp(03,S3)) and
        reduce(comp(03,S3),comp(02,S2))
import(comp(01,S1),comp(02,S2)) if
        all_imported(comp(01,S1)) and
        strengthen_onto(map(kif2terms,01,02,[]),comp(01,S1),comp(02,S2))
import(comp(01,S1),comp(02,S2)) if
        acquire(comp(01,S4)) and
        connect(map(id,01,01,[iso]),map(id,01,01,[iso]),comp(01,S1),comp(01,S4),comp(01,S5)) and
        import(comp(01,S5),comp(02,S2))
reduce(comp(01,S1),comp(02,S2)) if
        weaken_spec(comp(01,S1),comp(01,S3)) and
        weaken_spec(comp(01,S3),comp(01,S4)) and
```

transform comp(O1,S4) into comp(O2,S2)
Transform comp(O1,S4) into comp(O2,S2)
do a general transformation
strengthen the ontology
weaken the ontology
strengthen the specification
weaken the specification
connect with another component
apply a user-defined calculus rule
import and call another lifecycle
don't do anything
Apply
Cancel

Fig. 2. Editing Ecolingua's lifecycle

For this reason we have implemented a *lifecycle editor and interpreter* that (1) enables a knowledge engineer to analyse the lifecycle of a knowledge component, extract its abstract pattern, and devise a formal representation of it (see Figure 2), using rules of a formal lifecycle calculus that capture abstract lifecycle steps such as *weaken the ontology* by means of a channel-theoretic semantics[5]; and (2) is capable of executing the formal representation of a lifecycle.

Since lifecycle patterns are described at a generic level, we have implemented a *brokering service* that enacts the lifecycle execution in a distributed environment, so that a knowledge engineer can choose among several solvers capable of performing abstract lifecycle steps in a particular domain-specific fashion.

The lifecycle execution generates a *lifecycle history* of the knowledge component. This lifecycle history can later be used to infer properties of the components, not by inspecting the specification of the component itself — since this would require the cumbersome task of reasoning with the axioms that constitute the specification — but by inspecting the structure of its lifecycle.

* Supported under the AKT IRC, which is sponsored by the UK EPSRC under grant GR/N15764/01, and comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

[3] Corrêa da Sliva et al.: On the insufficiency of ontologies: problems in knowledge sharing and alternative solutions. Knowledge-Based Systems **15** (2002) 147–167.

[4] Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: a tool for collaborative ontology construction. International Journal of Human-Computer Studies **46** (1997) 707–727.

[5] Barwise, J., Seligman, J.: Information Flow: The Logic of Distributed Systems. Cambridge University Press (1997).

Distributed Semantic Perspectives

Oliver Hoffmann and Markus Stumptner

University of South Australia
(oliver@hoffmann.org,mst@cs.unisa.edu.au)

1 Introduction

With increased use of domain ontologies arises the task of combining or linking existing ontologies. But in general, existing ontologies are based on different semantic contexts and therefore simple merging of two or more existing ontologies is not feasible: symbols in one ontology might have little or no congruency with symbols in another ontology; therefore semantic context has to be reconstructed from within available data [1] or via intervention from outside the computer system. In this paper, a concept for re-construction of context and re-interpretation of knowledge on the basis of external user input is presented.

2 Non-Implementables

The semantics of a symbol can be described as a triangular relationship between the symbol itself, the entity referred to by the symbol and the concept some individual has of that entity [2].

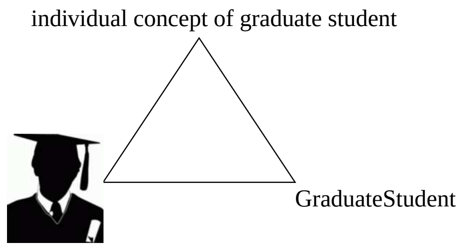


Fig. 1. the semantic triangle

Figure 1 shows a semantic triangle for the symbol GraduateStudent: The symbol is set into relation with a graduate student (real world entity, bottom left) and the concept of graduate student as conceived by some individual (top). Two of the three elements required for a semantic relationship (individual concept and real world entity) cannot be implemented in a computer system: A computer system can hold a reference to real world entities or concepts of real world entities, but such references would have to be symbols and as such only have semantic content when itself set in relation to entities and concepts outside of the computer system. Thus *non-implementables* (*nimps*) are necessary and irreplaceable preconditions for semantic content.

3 Standardized Interpretation

Although nimps are not directly accessible, knowledge engineering can influence nimps indirectly. For instance, individual concepts can be synchronized via the use of shared symbols. Formal semantics of data contained in ontologies facilitate compatible interpretation of new symbols on the basis of symbols already interpreted in a compatible way. Thus a standardized interpretation of symbols is developed, or a "world 3" as described by Karl Popper: In addition to the world of real entities ("world 1") and the concepts some individual has about the world ("world 2"), a third world of shared interpretations is built among some group of individuals with the help of symbols. But according to Popper, this standardization process has to be open ended [3] if the standards used are to serve a purpose like that envisioned in the semantic web initiative [4]. It has to remain possible to re-define standards and to re-interpret standards in different contexts. For example, the term "graduate student" is ill defined in a standard German University context, because of significant differences in University education structure. If the symbol GraduateStudent in some ontology is interpreted as referring to the level of expertise, then the number of semesters of prior study can be the appropriate translation. If GraduateStudent is interpreted as referring to the formal degree, then some adequate degree like "Diplom-Informatiker" might be the corresponding title. Thus implicitly standardized nimps associated with ontologies have to be taken into consideration when combining different ontologies.

4 Semantic Perspectives

The concept of semantic perspectives is a generalization of design perspectives [5]. Distributed semantic perspectives are ontologies loosely linked via plug-and-play translators and constitute different views on a common content., thus creating a distributed system of different perspectives on the common semantic content. In the case of a single semantic perspective without links to other perspectives, the nimp parts of it's semantic content can be assumed to be sufficiently standardized: Either the nimp parts of its semantic content are completely identical among different individuals interacting with the perspective or nimp differences are compatible with the perspective's intended functionality. If perspectives are to be combined, however, nimp differences have to be considered. A specific translator between two given perspectives operates context-less in the sense that it only operates on symbols, which can be interpreted as either ignoring nimp parts of semantic content or implicitly assuming a standardized semantic context for the translation. Therefore any automated choice of translator as in [6] is equivalent to restricting nimp parts of semantic content to one specific semantic context. In order to accomodate different contexts, translators between semantic perspectives can be turned on and off at run time and two given perspectives can be linked by any of a set of alternative translators, with each translator implicitly determining semantic context for both the source and target perspective.

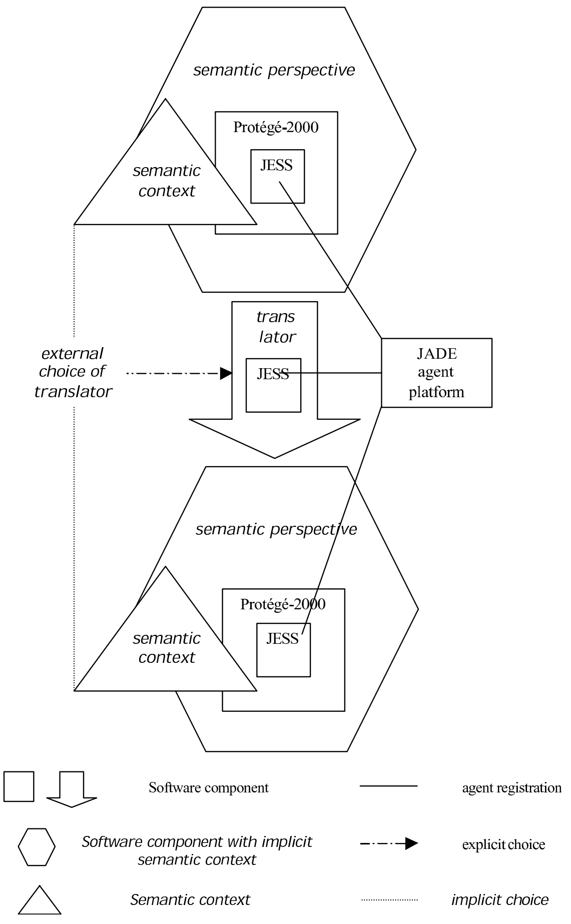


Fig. 2. external choice of translator

This dynamic re-combination of translators and perspectives (Figure 2), allows for re-interpretation and re-contextualization of semantic content, thus accomodating changing contexts. Distributed semantic perspectives can be linked and thus several ontologies can be combined, edited and used in parallel. In contrast, one final merging operation like in [7] would eliminate the option of re-interpreting context for source perspectives and limit interpretation for the merged result perspective.

Semantic perspectives have been implemented via the integration of the Java Agent Development Environment JADE [8] [1], a multi agent system, the Java Expert System Shell JESS [9] [2], a rule based language, Protégé-2000 [10] [3], an ontology editor, the JessTab [11] a plugin for Protégé-2000, and JadeJessProtege [4]. Two types of features are added to standard ontology editor features: First, the ability to extract and integrate ontological information from and into the running ontology editor; second, the ability to transmit and receive such ontological information to and from translators into or from other ontology editors. With these additional features, the ontology editor is turned into an agent. Perspectives and translators, which can be hosted on different computers, register with a JADE platform. Once a semantic perspective has established contact with the JADE agent platform and received a list of relevant translators, parts of the ontology can be extracted and sent to translators via JADE agent messages. Then translators re-contextualize content as illustrated by the following JESS code:

```
(Information (Perspective Ontology1) (Class Student) (Slot Name) (Value ?name))
(Information (Perspective Ontology1) (Class Matura) (Slot Name) (Value ?name))
(Information (Perspective Ontology1) (Class Matura) (Slot Year) (Value ?year))
(test (< ?year 1998))
=>
(assert (Information (Perspective Ontology2) (Class GraduateStudent) (Slot Name)(Value ?name)))
```

In this example, information extracted from the source perspective Ontology1 is translated into information for the target perspective Ontology2. The source perspective contains information that a student with some name has passed Matura (high school graduation) before 1998. This is translated into the information that the student is a graduate student. Here the number of years passed since some person came out of high school is used as an (imperfect) indicator of whether a student should be regarded as “graduate” student in a specific context.

[1] `http://jade.cselt.it`
[2] `http://herzberg.ca.sandia.gov/jess`
[3] `http://protege.stanford.edu`
[4] `http://JadeJessProtege.net`

References

1. Stuckenschmidt, H., Visser, U.: Semantic Translation based on Approximate Re-Classification. In: Proceedings of the Workshop “Semantic Approximation, Granularity and Vagueness”, Proc. KR2000, San Francisco, California, U.S.A., Morgan Kaufmann (2000)
2. Sowa, J.: Knowledge Representation: Logical, Philosophical and Computational Foundations. Brooks/Cole, Pacific Grove, California, U.S.A (2000)
3. Popper, K.: Objective Knowledge: An Evolutionary Approach. Oxford University Press, Oxford, U.K. (1972)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
5. Hoffmann, O., M.Stumptner, Chalabi, T.: Tolerating Inconsistencies: The Distributed Perspectives Model. In: Computer Aided Architectural Design Futures, Dortrecht, N.L., Kluwer (2001) 375–386
6. Campbell, A.E., Shapiro, S.: Algorithms for Ontological Mediation. Technical report, Dep. of CS and Engineering, State Univ. of New York at Buffalo (1998)
7. McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: An Environment for Merging and Testing Large Ontologies. In: Proc. KR2000, Breckenridge, Colorado, U.S.A. (2000)
8. Bellifemine, F., Poggi, A., Rimassa, G.: JADE – a FIPA-compliant agent framework. In: Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology(ATCS). (1999) 97–108
9. Friedmann-Hill, E.: Jess, the java expert system shell. Technical report, Sandia National Laboratories, Livermore, CA, U.S.A. (1998)
10. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., Musen, M.A.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems **2** (2001)
11. Eriksson, H.: The JessTab Approach to Protégé and JESS Integration. In: Intelligent Information Processing IIP-2002, Montreal, Canada (2002)

A framework to solve the ontology translation problem

Oscar Corcho

Facultad de Informática. Universidad Politécnica de Madrid
Campus de Montegancedo, s/n. 28660 Boadilla del Monte. Madrid. Spain
`ocorcho@fi.upm.es`

Abstract. Ontologies are developed with different tools and languages. Reusing an ontology usually requires transforming it from its original format to a target format. However, many problems usually arise in these transformations, related to the compatibility among tools/languages. We propose an ontology reengineering methodology (with its technological support) as a solution to the ontology translation problem.

1 Introduction

Nowadays, different tools exist for developing ontologies: OILEd, OntoEdit, Ontolingua, OntoSaurus, Protégé2000, WebODE, WebOnto, etc. Each tool has their own knowledge model, and usually allows exporting/importing ontologies in their own textual representation.

Several languages are also used for the implementation of ontologies, such as Ontolingua, LOOM, OCML, FLogic, XOL, SHOE, RDF(S), OIL, DAML+OIL, etc. Apart from their lexical and syntactical differences, there are also more significant ones due to the knowledge representation (KR) formalism in which they are based (frames, semantic nets, description logic, etc.) and the semantics of their representation primitives and constructs, which fully determine both their expressiveness and reasoning capabilities.

The ontology translation problem appears when we decide to reuse an ontology (or part of an ontology) using a tool or language that is different from those ones in which the ontology is available. If we force each ontology developer, individually, to commit to the task of translating and incorporating the necessary ontologies to their systems, they will need both a lot of effort and a lot of time to achieve their objectives. Therefore, ontology reuse in different contexts will be highly boosted as long as we provide automatic ontology translation services among those languages and/or tools.

2 Characterisation of the ontology translation problem

The first reference to this problem was presented by Gruber in [2]. He proposed, as a solution, to follow a set of *ontological commitments* when an ontology was created. From all these ontological commitments, the ”minimal encoding bias” deserves special attention: Gruber proposed to conceptualise ontologies in the knowledge level, instead of doing it in the symbolic level, and to implement them using automatic translators. However, this criterion has not been commonly followed in ontology development, forcing ontology developers to translate existing ontologies manually or create ad-hoc translators between languages or knowledge models, which is a time consuming task.

Translation problems can be classified as follows:

Lexical problems. They appear when the terms used for language identifiers, texts and constructs (names of components, sizes of their textual descriptions, etc.) follow different conventions in the different languages and/or tools. For instance, concept National Park in Ontolingua is usually written as *National-Park*, while in FLogic hyphens are not allowed inside identifiers (hence, it is written as *NationalPark*).

Syntax problems. Different languages/tools use different grammars to represent their components. Some languages/tools also allow defining the same component in different ways. When performing translations, both situations must be taken into account.

Expressiveness problems. These problems are caused because different languages/tools are based on different KR paradigms. First, not all the languages allow expressing the same knowledge: we must analyse what components can be translated directly from a language to another one, what components can be expressed using other components from the target language, what components cannot be expressed in the target language, and what components can be expressed, although with losses of expressiveness.

Reasoning problems. The existence or not of an inference engine for a language, and the characteristics of this inference engine, usually bias the implementation of an ontology.

3 The framework: WebODE and OntoDialect

We propose to solve the translation problem in the context of a methodology for ontological reengineering. We distinguish three main phases: *reverse engineering* (we transform automatically an ontology that has been coded in a language to a knowledge model that is independent from the implementation); *reestructuration* (performed in the knowledge level, in accordance to the future uses of the ontology in an application); and *implementation* (we transform automatically the reestructured ontology into the target language).

This methodology is technologically supported by the WebODE ontology-engineering workbench [1]. Reestructuration is currently performed manually, with the WebODE ontology editor. Translators from WebODE to ontology languages/tools and vice versa can be created with the OntoDialect system, integrated in the workbench.

References

1. Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A. WebODE: a scalable workbench for ontological engineering. First International Conference on Knowledge Capture (KCAP2001). Victoria. Canada. October, 2001.
2. Gruber R. A translation approach to portable ontology specification Knowledge Acquisition. #5: 199-220. 1993

Author Index

Bowers, Shawn 19

Casati, Fabio 1
Castano, Silvana 51
Chu, Wesley W. 3
Corcho, Oscar 114

Delcambre, Lois 19
Dell’Erba, Mirella 69

Euzenat, Jérôme 93

Fan, Hao 36
Felfernig, Alexander 11
Ferrara, Alfio 51
Fodor, Oliver 69
Friedrich, Gerhard 11

Hasida, Kôiti 110
Hoffmann, Oliver 112

Jannach, Dietmar 11

Kiryakov, Atanas 27

Lee, Dongwon 3

Maedche, Alexander 60
Mani, Murali 3
Mitra, Prasenjit 43
Miyata, Takashi 110
Motik, Boris 60

Ognyanov, Damyan 27
Omelayenko, Borys 77

Peer, Joachim 101
Potter, Stephen 85, 111
Poulovassilis, Alexandra 36

Ricci, Francesco 69
Robertson, Dave 85
Robertson, David 111

Schorlemmer, W. Marco 85, 111
Silva, Nuno 60
Sleeman, Derek 85, 111
Spada, Antonella 69
Stuckenschmidt, Heiner 93
Stumptner, Markus 11, 112

Volz, Raphael 60

Werthner, Hannes 69
Wiederhold, Gio 43

Zanker, Markus 11