Frontiers in Artificial Intelligence and Applications

KNOWLEDGE TRANSFORMATION FOR THE SEMANTIC WEB

Edited by
Borys Omelayenko
Michel Klein

IOS Press
Ohmsha

KNOWLEDGE TRANSFORMATION FOR THE SEMANTIC WEB

Frontiers in Artificial Intelligence and Applications

Series Editors: J. Breuker, R. López de Mántaras, M. Mohammadian, S. Ohsuga and W. Swartout

Volume 95

Recently published in this series:

Vol. 94. H. Jaakkola et al. (Eds.), Information Modelling and Knowledge Bases XIV
Vol. 93. K. Wang, Intelligent Condition Monitoring and Diagnosis Systems – A Computational Intelligence
Vol. 92. V. Kashyap and L. Shklar (Eds.), Real World Semantic Web Applications
Vol. 91. F. Azevedo, Constraint Solving over Multi-valued Logics – Application to Digital Circuits
Vol. 90. In preparation
Vol. 89. T. Bench-Capon et al. (Eds.), Legal Knowledge and Information Systems – JURIX 2002: The Fifteenth Annual Conference
Vol. 88. In preparation
Vol. 87. A. Abraham et al. (Eds.), Soft Computing Systems – Design, Management and Applications
Vol. 86. In preparation
Vol. 85. J.M. Abe and J.I. da Silva Filho (Eds), Advances in Logic, Artificial Intelligence and Robotics – LAPTEC 2002
Vol. 84. H. Fujita and P. Johannesson (Eds.), New Trends in Software Methodologies, Tools and Techniques – Proceedings of Lyee_W02
Vol. 83. V. Loia (Ed.), Soft Computing Agents – A New Perspective for Dynamic Information Systems
Vol. 82. E. Damiani et al. (Eds.), Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies – KES 2002
Vol. 81. J.A. Leite, Evolving Knowledge Bases – Specification and Semantics
Vol. 80. T. Welzer et al. (Eds.), Knowledge-based Software Engineering – Proceedings of the Fifth Joint Conference on Knowledge-based Software Engineering
Vol. 79. H. Motoda (Ed.), Active Mining – New Directions of Data Mining
Vol. 78. T. Vidal and P. Liberatore (Eds.), STAIRS 2002 – STarting Artificial Intelligence Researchers Symposium
Vol. 77. F. van Harmelen (Ed.), ECAI 2002 – 15th European Conference on Artificial Intelligence
Vol. 76. P. Sinčák et al. (Eds.), Intelligent Technologies – Theory and Applications
Vol. 75. I.F. Cruz et al. (Eds.), The Emerging Semantic Web – Selected Papers from the first Semantic Web Working Symposium
Vol. 74. M. Blay-Fornarino et al. (Eds.), Cooperative Systems Design – A Challenge of the Mobility Age
Vol. 73. H. Kangassalo et al. (Eds.), Information Modelling and Knowledge Bases XIII
Vol. 72. A. Namatame et al. (Eds.), Agent-Based Approaches in Economic and Social Complex Systems
Vol. 71. J.M. Abe and J.I. da Silva Filho (Eds.), Logic, Artificial Intelligence and Robotics – LAPTEC 2001
Vol. 70. B. Verheij et al. (Eds.), Legal Knowledge and Information Systems – JURIX 2001: The Fourteenth Annual Conference
Vol. 69. N. Baba et al. (Eds.), Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies – KES'2001
Vol. 68. J.D. Moore et al. (Eds.), Artificial Intelligence in Education – AI-ED in the Wired and Wireless Future
Vol. 67. H. Jaakkola et al. (Eds.), Information Modelling and Knowledge Bases XII
Vol. 66. H.H. Lund et al. (Eds.), Seventh Scandinavian Conference on Artificial Intelligence – SCAI'01

ISSN 0922-6389

Knowledge Transformation for the Semantic Web

Edited by

Borys Omelayenko

Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

and

Michel Klein

Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

IOS Press

Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

ISBN 1 58603 325 5 (IOS Press)
ISBN 4 274 90580 2 C3055 (Ohmsha)
Library of Congress Control Number: 2003101036

Publisher
IOS Press
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
fax: +31 20 620 3419
e-mail: order@iospress.nl

Distributor in the UK and Ireland
IOS Press/Lavis Marketing
73 Lime Walk
Headington
Oxford OX3 7AD
England
fax: +44 1865 75 0079

Distributor in the USA and Canada
IOS Press, Inc.
5795-G Burke Centre Parkway
Burke, VA 22015
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

Distributor in Germany, Austria and Switzerland
IOS Press/LSL.de
Gerichtsweg 28
D-04103 Leipzig
Germany
fax: +49 341 995 4255

Distributor in Japan
Ohmsha, Ltd.
3-1 Kanda Nishiki-cho
Chiyoda-ku, Tokyo 101-8460
Japan
fax: +81 3 3233 2426

PRINTED IN THE NETHERLANDS

Preface

The Web is changing. HTML and plain text documents designed for a human user are currently being replaced by the documents enriched with conceptual models and logical domain theories that can be processed automatically, following the emerging vision of the Semantic Web. In addition to the documents, the models for underlying databases, software, and business processes running behind the web pages are in the process of making their models open and available automatic processing. These models are focused on different design aspects and different modelling tasks and they need to be integrated and translated.

Historically different modelling means have been used to model data, knowledge, software architectures, business processes, etc., and the integration problems have always been the bottleneck in computer science development. The space of modelling languages is pretty large and includes XML,[1] XML Schemas,[2] relational and Entity-Relation (ER) models, Resource Description Framework[3] (RDF), RDF Schema,[4] DAML+OIL[5] and OWL[6] ontology languages, Universal Modelling Language[7] (UML), Prolog, Web Service Modelling Language[8] (WSDL), Topic Maps,[9] etc. Various practical integration tasks require many of these frameworks to be connected and the documents described in one language to be translated into another one. In this book we present various approaches that describe some of these connections.

The first part of the book is devoted to several techniques describing the transformation of schemas between different formalisms. In the first chapter on schema conversion methods between XML and relational models, Lee et al. describe three different methods for transforming XML Schemas into relational schemas and vice versa. In the following chapter on transforming data models with UML Gogolla and Lindow present an approach for describing and transforming different data models with meta-modelling techniques based on the Meta Object Facility (a part of the UML language). In the third chapter devoted to modelling conformance for flexible transformation over data models Bowers and Delcambre present a schema transformation technique based on meta-modelling. Their approach allows performing the transformations via mappings between general modelling constructs of data schemas, taking XML, RDF, relational model, and topic maps as a running example. Euzenat and Stuckenschmidt look at the problem of schema transformation from a completely different perspective. In their chapter on the 'family of languages' approach to semantic interoperability they suggest a framework in which different schemas can be transformed between a set of languages that differ in expressivity, preserving a particular formal property. The first part is concluded by the chapter written by Fan and Poulovassilis on tracing data lineage using schema transformation pathways, which discusses a way of tracing data lineage by using the schema transformation pathways taking relational schemas as a running example.

---

[1] http://www.w3.org/XML/
[2] http://www.w3.org/XML/Schema
[3] http://www.w3.org/RDF/
[4] http://www.w3.org/TR/rdf-schema/
[5] http://www.daml.org/
[6] http://www.w3.org/2001/sw/
[7] http://www.uml.org/
[8] http://www.w3.org/TR/wsdl
[9] http://www.topicmaps.org/

The second part of the book presents different approaches aimed at reusing existing resources as models that can be used on the Semantic Web. Sleeman et al. in their chapter on ontology extraction for distributed environments discuss a technique and a tool capable of identifying ontological knowledge that is implicitly available in logic-based Prolog and rule-based CLIPS knowledge bases. In the following chapter about transformation-based approaches for using UML on the Semantic Web Falkovych et al. focus on the conversion between UML and web-based ontology languages DAML+OIL and OWL, which allows using UML tools and models in the Semantic Web. Castano and Ferrara in their work on knowledge representation and transformation in ontology-based data integration use the knowledge that is available in XML structures and show how it can be expressed in DAML+OIL ontologies and used to integrate heterogeneous XML data sources. In the next chapter, devoted to a logic programming approach to RDF document and query transformation, Peer discusses the transformation of existing RDF documents and RDF queries by converting them to Prolog and constructing Prolog programs within the Logic Programming paradigm.

The last two chapters of the book focus on the knowledge transformation problems that arise in the area of dynamic web services, where conceptual models represented by XML documents need to be interpreted together with the process descriptions representing the dynamic aspect of a service. In the first chapter Omelayenko presents a mapping meta-ontology for web service integration capable of mapping different document and process models in a unified way that is easily interpretable by inference engines. Finally, Felfernig et al. present an approach for transforming UML domain descriptions into configuration knowledge bases by converting the UML descriptions into DAML+OIL and using them to derive capability descriptions for Web services.

Undoubtedly, further research in the Semantic Web area will bring new knowledge transformation tasks into the focus of various research communities, and new bridges will be made between different modelling means. The community still has a long way to go before the integration tasks are solved, and we hope that this book makes a step along this way.

We would like to thank the authors who allocated their time and effort in contributing their high-quality chapters. Finally, we appreciate the support provided by Dieter Fensel and the OntoWeb[10] network of excellence.

December 2002

Borys Omelayenko
Michel Klein
Editors

[10] http://www.ontoweb.org

Contents

This page intentionally left blank

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Schema Conversion Methods between XML and Relational Models

Dongwon Lee[1] Murali Mani[2] Wesley W. Chu[2]

[1] *School of Information Sciences and Technology*
Penn State University
dongwon@psu.edu

[2] *Dept. of Computer Science*
University of California, Los Angeles
{mani,wwc}@cs.ucla.edu

Abstract. In this chapter, three semantics-based schema conversion methods are presented: 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 formats 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 formats between XML and relational models will grow substantially. To this end, several schema conversion 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 conversion algorithms only capture the *structure* of the original schema and largely ignore the hidden *semantic constraints*. To clarify, consider the following 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 conversion algorithms, so far, have largely ignored the hidden *semantic constraints* of the original schema.

1.1 Related Work

Schema Conversion vs. Schema Matching: It is important to differentiate the problem that we deal with in this chapter, named as *schema conversion* problem, from another similar one known as *schema matching* problem. Given a *source* schema s_1 and a *target* schema t_1, the schema matching problem finds a "mapping" that relates elements in s_1 to ones in t_1. On the other hand, in the schema conversion problem, only a *source* schema s_2 is given and the goal is to find a *target* schema t_2 that is equivalent to s_2. Often, the source and target schemas in the schema matching problem belong to the same data model[1] (e.g., relational model), while they belong to different models in the schema conversion problem (e.g., relational and XML models). Schema matching problem itself is a difficult problem with many important applications and deserves special attention. For further discussion on the schema matching problem, refer to [6] (survey), [7] (latest development), etc.

Between XML and Non-relational Models: Schema conversion between different models has been extensively investigated. Historically, the trend for schema conversion has always been between consecutive models or models with overlapping time frames, as they have evolved (e.g., between Network and Relational models [8, 9], between ER and OO models [10, 11], or between UML and XML models [12, 13, 14, 15]). For instance, [16] deals with conversion 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 conversion techniques to other data models have been extensively studied in ER research. For instance, [17] presents an overview of such techniques. However, due to the differences between ER and XML models, those conversion techniques need to be modified substantially. In general, since works developed in this category are often ad hoc and were aimed at particular applications, it is not trivial to apply them to schema conversion between XML and relational models.

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

[1] There are cases where schema matching problem deals with a mapping between different data models (e.g., [6]), but we believe most of such cases can be replaced by: 1) a schema conversion between different models, followed by 2) a schema matching within the same model.

documents to relational format. Because [18] discusses template language-based conversion from DTD to relational schema, it requires human experts to write an XML-based conversion 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 [19]. Since the method is orthogonal to the structure-oriented conversion method, it can be used along with algorithms in [2, 18, 4, 3].

From Relational to XML: There have been different approaches for the conversion from the relational model to XML model, such as XML Extender from IBM, XML-DBMS, SilkRoute [20], and XPERANTO [5]. All of 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.

1.2 Overview of Three Schema Translation Algorithms

In this chapter, we present three schema conversion algorithms that not only capture the structure, but also the semantics of the original schema.

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 inter-connected tables into a coherent and hierarchical parent-child structure in the final XML schema.

2 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 conversion

```
<!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`.

algorithms (e.g., [18, 2, 3, 4]) have so far focused mainly on the issue of structural conversion, largely ignoring the semantics that 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 DTD and XML document in Tables 1 and 2 as examples.

2.1 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 [17], let us call each cardinality relationship as type (0,1), (1,1), (0,N), (1,N), respectively. From these cardinality relationships, three major constraints can be

```
<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.

inferred. The 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. The second is whether or not more than one sub-element can occur. This is also known as *singleton constraint* in [21] and is one kind of equality-generating dependencies. The 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 appear as values in the columns of other fragments and is a generalization of the notion of *referential integrity*.

A 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[2]. 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 best be 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.

Equality-Generating Dependencies (EGDs): The *Singleton Constraint* [21] 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 is 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 an 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* [21].

[2]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 [22].

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.

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.

2.2 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 2.1. 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 1 illustrates an example of such 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, 19] 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 2.1. The details of the algorithm are beyond the scope of this chapter and interested readers can refer to [19]. For instance, Figure 2 and Table 4 are such output relational schema and data in SQL notation, automatically generated by the CPI algorithm.

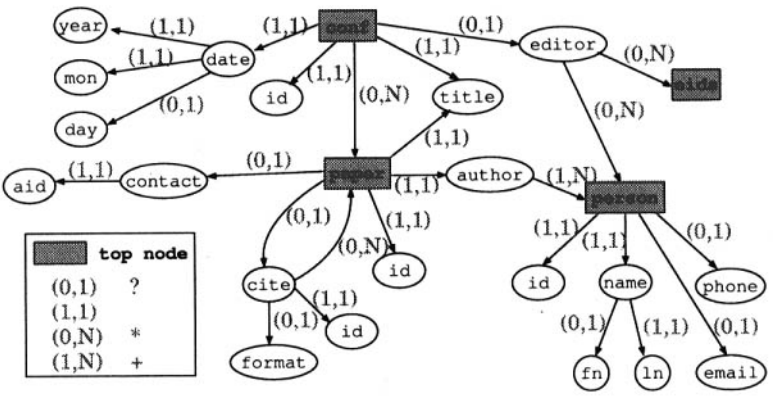


Figure 1: An annotated *DTD graph* for the Conference DTD in Table 1.

```
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)
);
```

Figure 2: Final relational "schema" for the paper element in the Conference DTD in Table 1, generated by CPI algorithm.

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.

3 The NeT Algorithm

The simplest Relational-to-XML translation method, termed as FT (Flat Translation) in [23], 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 sub-elements* 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 [24] 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[3]. Formally,

Definition 1 (Nest) *[24]. 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 only has 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 1 *Consider a table R in Table 5. Here we assume that the columns A, B, and 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 1, there are various ways to

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

	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.

nest the given table. Therefore, it is important to find an efficient way (that uses the *nest* operator a 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 [23, 25].

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 may remain 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.

4 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 an *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:
   (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 graphical representation, termed 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*.*

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

Table 6: An example schema with associated INDs.

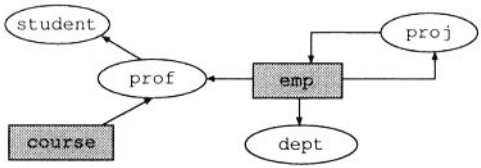


Figure 3: The IND-Graph representation of the schema in Table 6 (*top nodes* denoted by rectangular nodes).

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 a 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 2 *Consider a schema and its associated INDs in Table 6. The IND-Graph with two top nodes is shown in Figure 3: 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 a BFS scan. Pull up a reachable node* prof *into* course *and label it as a 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>*.*

DTD Semantics		DTD Schema		Relational Schema			
Name	Domain	Elm/Attr	ID/IDREF(S)	Table/Attr	$\rightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow \emptyset$
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.

- *Next, visit a node* `proj` *and pull it up to* `emp` *by* `<!ELEMENT emp (Eid, Name, ProjName, dept*, proj*)>` *and* `<!ELEMENT proj (Pname)>`. *In the next step, visit a node* `emp` *from* `prof`. *Since it was already visited, an attribute* `Ref_emp` *of type* `IDREFS` *is added to* `proj`, *and the 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.

5 Experimental Results

5.1 CPI Results

CPI was tested against DTDs gathered from OASIS[4]. 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 in 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 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.

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

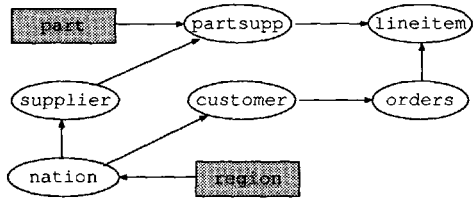
Test Set	Attr. / tuple	NestRatio	ValueRatio	Size before / after	Nested attr.	Time (sec.)
Balloons1	5 / 16	42 / 64	80 / 22	0.455 / 0.152	3	1.08
Balloons2	5 / 16	42 / 64	80 / 22	0.455 / 0.150	3	1.07
Balloons3	5 / 16	40 / 64	80 / 42	0.455 / 0.260	3	1.14
Balloons4	5 / 16	42 / 64	80 / 22	0.455 / 0.149	3	1.07
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.

$$\text{NestRatio} = \frac{\#\text{ of successful nesting}}{\#\text{ of total nesting}}$$

$$\text{ValueRatio} = \frac{\#\text{ of original data values}}{\#\text{ of data values in nested table}}$$

(a) Metrics



(b) The IND-Graph representation of TPC-H schema

Figure 4: Metrics and IND-Graph.

5.2 *NeT Results*

Our preliminary results comparing the goodness of the *XSchema* obtained from NeT and FT with that obtained from DB2XML v 1.3 [26] appeared in [23]. We further applied our NeT algorithm on several test sets drawn from UCI KDD[5] / ML[6] repositories, which contain a multitude of single-table relational schemas and data. Sample results are shown in Table 8. Two metrics are shown in Figure 4(a). A high value for *NestRatio* shows that we did not perform unnecessary nesting and the high value for *ValueRatio* shows that the nesting removed a great deal of redundancy.

In our experimentation[7], we observed that most of the attempted nestings were successful, and hence our optimization rules are quite efficient. In Table 8, we see that nesting was useful for all 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.

[5] http://kdd.ics.uci.edu/

[6] http://www.ics.uci.edu/~mlearn/MLRepository.html

[7] Available at http://www.cs.ucla.edu/~mani.xml

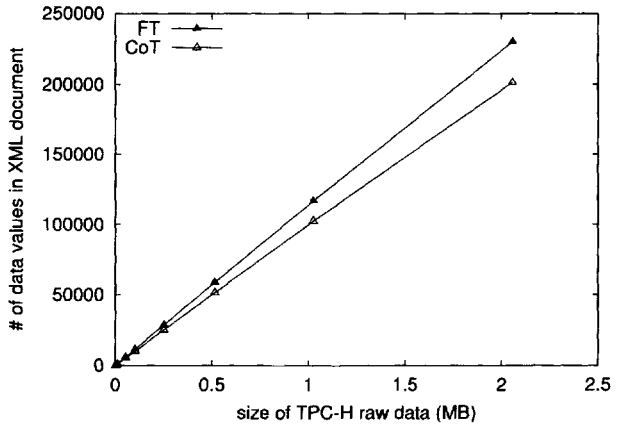


Figure 5: Size comparison of two algorithms.

5.3 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[8], 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 4(b). 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 a sub-element, and the remaining two are mapped using `IDREF` attributes.

Figure 5 shows a comparison of the number of data values originally present in the database, and the number of data 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%.

6 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 have all preserved the original information of relational schema. For instance, using the notion of information capacity [27], 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 conversions*.

Despite the difficulties in conversions between XML and relational models, there are many practical benefits. We strongly believe that devising more accurate and efficient con-

[8] `http://www.tpc.org/tpch/spec/h130.pdf`

version 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 International Conference on Management of Data, Philadephia, PA (1998)

[3] Florescu, D., Kossmann, D.: "Storing and Querying XML Data Using an RDBMS". IEEE Data Engineering 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: International Conference on Very Large Data Bases, 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: International Workshop on the Web and Databases, Dallas, TX (2000)

[6] Madhavan, J., Berstein, P.A., Rahm, E.: "Generic Schema Matching with Cupid". In: International Conference on Very Large Data Bases, Roma, Italy (2001)

[7] Miller, R.J., Haas, L., Hernandez, M.A.: "Schema Mapping as Query Discovery". In: International Conference on Very Large Data Bases, Cairo, Egypt (2000)

[8] Navathe, S.B.: "An Intuitive Approach to Normalize Network Structured Data". In: International Conference on Very Large Data Bases, Montreal, Quebec, Canada (1980)

[9] Lien, Y.E.: "On the Equivalence of Database Models". Journal of the ACM **29** (1982) 333–362

[10] Boccalatte, A., Giglio, D., Paolucci, M.: "An Object-Oriented Modeling Approach Based on Entity-Relationship Diagrams and Petri Nets". In: IEEE Internal conference on Systems, Man and Cybernetics, San Diego, CA (1998)

[11] Gogolla, M., Huge, A.K., Randt, B.: "Stepwise Re-Engineering and Development of Object-Oriented Database Schemata". In: International Workshop on Database and Expert Systems Applications, Vienna, Austria (1998)

[12] Bisova, V., Richta, K.: "Transformation of UML Models into XML". In: ADBIS-DASFAA Symposium on Advances in Databases and Information Systems, Prague, Czech Republic (2000)

[13] Conrad, R., Scheffner, D., Freytag, J.C.: "XML Conceptual Modeling using UML". In: International Conference on Conceptual Modeling, Salt Lake City, UT (2000)

[14] Hou, J., Zhang, Y., Kambayashi, Y.: "Object-Oriented Representation for XML Data". In: International Symposium on Cooperative Database Systems for Advanced Applications, Beijing, China (2001)

[15] Al-Jadir, L., El-Moukaddem, F.: "F2/XML: Storing XML Documents in Object Databases". In: International Conference on Object Oriented Infomation Systems, Montpellier, France (2002)

[16] Christophides, V., Abiteboul, S., Cluet, S., Scholl, M.: "From Structured Document to Novel Query Facilities". In: ACM SIGMOD International Conference on Management of Data, Minneapolis, MN (1994)

[17] Batini, C., Ceri, S., Navathe, S.B.: "Conceptual Database Design: An Entity-Relationship Approach". The Benjamin/Cummings Pub. (1992)

[18] Bourret, R.: "XML and Databases". Web page (1999) `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.

[19] Lee, D., Chu, W.W.: "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". Journal of Data & Knowledge Engineering **39** (2001) 3–25

[20] Fernandez, M.F., Tan, W.C., Suciu, D.: "SilkRoute: Trading between Relations and XML". In: International World Wide Web Conference, Amsterdam, Netherlands (2000)

[21] Wood, P.T.: "Optimizing Web Queries Using Document Type Definitions". In: International Workshop on Web Information and Data Management, Kansas City, MO (1999) 28–32

[22] Lee, D., Chu, W.W.: "Comparative Analysis of Six XML Schema Languages". ACM SIGMOD Record **29** (2000) 76–87

[23] Lee, D., Mani, M., Chiu, F., Chu, W.W.: "Nesting-based Relational-to-XML Schema Translation". In: International Workshop on the Web and Databases, Santa Barbara, CA (2001)

[24] Jaeschke, G., Schek, H.J.: "Remarks on the Algebra of Non First Normal Form Relations". In: ACM Symposium on Principles of Database Systems, Los Angeles, CA (1982)

[25] Lee, D., Mani, M., Chiu, F., Chu, W.W.: "NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints". In: ACM International Conference on Information and Knowledge Management, McLean, VA (2002)

[26] Turau, V.: "Making Legacy Data Accessible for XML Applications". Web page (1999) `http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html`.

[27] Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: "Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice (Extended Abstract)". In: Extending Database Technology, Cambridge, UK (1994)

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Transforming Data Models with UML

Martin Gogolla Arne Lindow

University of Bremen, Computer Science Department
Database Systems Group, D-28334 Bremen, Germany
{gogolla,lindow}@informatik.uni-bremen.de

Abstract. This chapter studies an approach to establish a formal connection between data models, in particular between conceptual data models and implementation data models. We use metamodeling techniques based on the Meta Object Facility MOF. MOF may be regarded as a subset of the Unified Modeling Language UML. As prominent example data models, we formally describe and thereby analyze the Entity-Relationship and the Relational data model. In addition, we represent the transformation between these data models by MOF language features. Thus we describe the data models and their transformation within a single framework. All results are formally represented and validated by a MOF compliant tool. The approach presented is general enough so that it can used for other data models being important for the Semantic Web, e.g., the object-oriented data model or semi-structured data models like XML-based models.

1 Introduction

Without doubt conceptual modeling of information systems with the Entity-Relationship model is a well-accepted standard technique in software design. The area of conceptual modeling also has connections to ontology based approaches which claim to provide a semantically richer grounding of information. Thus conceptual modeling is relevant for questions concerning the Semantic Web. Nowadays, central aspects of Entity-Relationship schemata can be found in the Unified Modeling Language UML [1] under the name class diagram. Transformations of Entity-Relationship schemata into implementation-oriented data models like the Relational data model also belong to standard software development techniques.

Recently, much attention has been paid to metamodeling approaches within the software engineering community. For example, UML is described in a semi-formal way by a metamodel with language features of the so-called Meta Object Facility MOF [2]. Roughly speaking, MOF is a subset of UML using class diagrams, i.e. a description formalism originating from the Entity-Relationship model. For UML, the abstract syntax is described with class diagrams and context-sensitive conditions are formulated with the Object Constraint Language OCL [1, 3], a textual sublanguage of UML having a precise set-theoretic semantics [4].

The present chapter now links classical database models like the Entity-Relationship and the Relational model with the MOF metamodeling approach. We present a MOF model for the Entity-Relationship and the Relational model. This is similar to modeling of UML with MOF where class diagrams are specified and corresponding invariants are formulated with OCL. In addition, we present the transformation of Entity-Relationship schemata into Relational schemata with a MOF class diagram and corresponding OCL rules. All three layers,

i.e. Entity-Relationship, Relational, and transformation layer, are formally specified and validated with the UML Specification Environment USE [5], an animation system for UML descriptions. USE supports also central MOF language features and we employ it as a MOF compliant tool.

The purpose of the chapter is to make assumptions of data models explicit and provide possibility for verification support. We want to formally ground and to represent well-known information system techniques by current formal methods. We do this in order to better understand underlying principles and to gain new formal insight into the fundamental nature of these data models and their transformation. Such formal MOF representations of data models are also needed in the context of newer proposals for handling data warehouses like the Common Warehouse Metamodel CWM [6]. The transformation of Entity-Relationship schemata into Relational schemata is long known. However, we are not aware of an approach explaining both data models and their transformation in a single, complete formal and logical setting. Work on transformation is usually done in an algorithmic way. This algorithmic way of describing translations has its analogy in the web area in terms of different serializations of ontology metamodels. Therefore, the purpose of this chapter is to precisely present that transformation and to show and study formal and informal transformation properties. Our approach has the advantage that it allows a very condensed representation of schemata and transformations. The central ingredients of the Entity-Relationship and the Relational data model and their transformation can be pictured on a single diagram.

Our work has connections to other approaches studying metamodels or metaobjects. Formal transformations between data models become important in the context of newer approaches [6, 7] which link metamodeling and data warehouses. Beside current work on formal foundation of the UML metamodel [8] there have been studies on the metamodel of predecessors of UML like OMT [9] or Booch [10]. The metamodel of OML has also been worked out [11]. Apart from these efforts concentrating on software engineering aspects, metamodels have also been considered in the context of information systems [12, 13]. Metamodeling has been successfully employed for repositories [14, 15], database design [16], and data warehouse design [17]. Other use of metamodels and metaobjects was made in the area of schema integration [18], engineering data warehouses [19], and Semantic Web applications [20]. Well recognized in the Semantic Web community are approaches like [21] studying also metamodels for transformations.

The structure of the rest of this chapter is as follows. Section 2 shortly introduces our metamodel for the Entity-Relationship and the Relational data model and their transformation. Section 3 explains our running example in conventional notation. Section 4 shows how the example schemata can be considered as instantiations of the defined Entity-Relationship and Relational metamodels. In Section 5 the details of restricting OCL invariants are explained. Section 6 compares properties of the Entity-Relationship and the Relational model on the basis of the metamodel representation. Section 7 closes the chapter with concluding remarks.

2 MOF Class Diagram

The central ingredients of the Entity-Relationship and the Relational data model and their transformation can be pictured on a single diagram. In Figure 1 we show a metamodel, a MOF class diagram, for the Entity-Relationship (ER) data model, for the Relational data

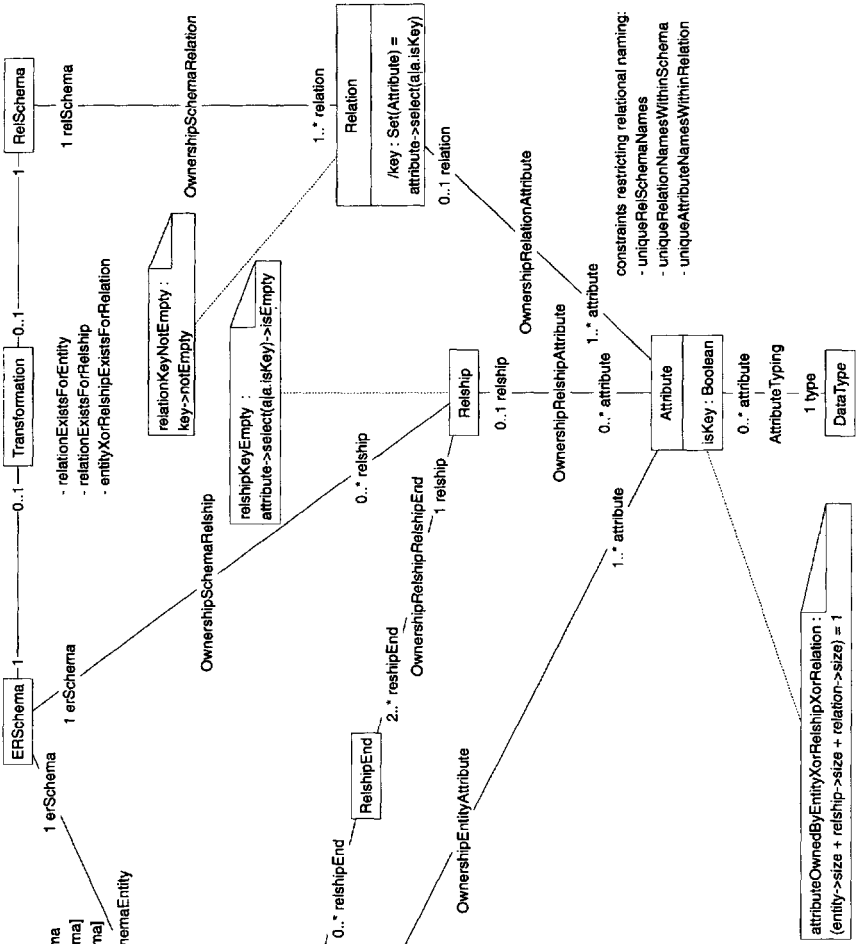
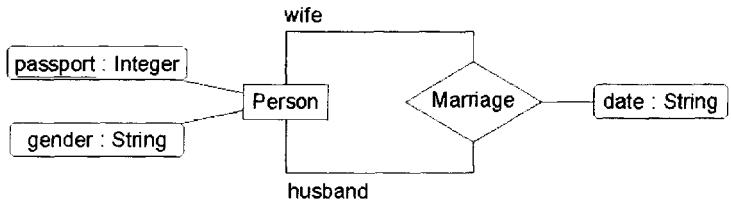


Figure 1: Metamodel for Transforming the ER into the Relational Model




```
Person(passport:Integer, gender:String)
Marriage(wife_passport:Integer, husband_passport:Integer, date:String)
```

Figure 2: Example for an Entity-Relationship and a Relational Schema

model, and the transformation from the Entity-Relationship model into the Relational model. The left part with classes ERSchema, Entity, Relship, and RelshipEnd belongs to the Entity-Relationship model, the right part with classes RelSchema and Relation represents the Relational model, the bottom part with classes Attribute and DataType is shared by both data models, and the top part with class Transformation reflects the data model transformation aspect. One main purpose of this chapter is explaining this metamodel. We do this by studying a detailed example.

But first we will have a short walk through that metamodel. The Entity-Relationship part of the metamodel expresses that an ERSchema consists of at least one Entity object and possibly zero or more Relship objects. A Relship object possesses two or more RelshipEnd objects which in turn are typed through an association to class Entity. Entity and Relship objects may possess Attribute objects which are typed through an association to class DataType.

The Relational part of the metamodel expresses that a RelSchema consists of at least one Relation which in turn may possess one or more Attribute objects. As in the Entity-Relationship part, Attribute objects are typed through an association to class DataType.

The transformation part of the metamodel consists of class Transformation together with two associations expressing that a Transformation object is linked to exactly one ERSchema object and one RelSchema object. This expresses that translation of the Entity-Relationship schema will result in the Relational schema.

Concerning attributes and operations of the above classes, all classes inherit from a class Named (not shown in Figure 1) so that they all possess an attribute name giving a String value for the object's name. Class Attribute additionally has a boolean attribute isKey indicating whether the respective Attribute object contributes to the key of the schema component, i.e. the entity or the relation, to which the Attribute object belongs. Classes Entity and Relation possess a derived, set-valued operation key() giving the set of key attributes of the entity and the relation, respectively. The diagram also mentions the names of constraints to be explained in detail in Section 5.

3 Running Example in Conventional Notations

Figure 2 shows an Entity-Relationship and a Relational schema which we will use as a running example. The Entity-Relationship schema pictures one entity Person with two exemplary attributes passport and gender. The underlining indicates that attribute passport is the key attribute for entity Person. The Entity-Relationship schema also includes one relationship Marriage. It is a reflexive relationship where a single entity participates twice. Two different kinds of participation are distinguished by role names wife and husband. Relationship Marriage possesses one attribute date. The Relational schema represents a direct translation of the Entity-Relationship schema into the Relational data model. Both main Entity-Relationship schema components are translated into a relation, respectively. Relation Person possesses two attributes passport and gender. Relation Marriage owns three attributes wife_passport, husband_passport, and date. Again, key attributes are underlined. Thus, the key for relation Person consists of attribute passport, and the key for relation Marriage is made of two attributes wife_passport and husband_passport. The data types of attributes are notated in both schemata after colons.

4 Running Example as MOF Object Diagram

The object diagram in Figure 3 shows three things: (1) The ER schema as well as (2) the Relational schema from Figure 2 and (3) the formal transformation between them. That object diagram belongs to metaclass diagram given in Figure 1. It is a screenshot of the USE system. The object diagram is divided into four parts: (1) The upper part with white object rectangles for Transformation, ERSchema, and RelSchema objects, (2) the middle left part with light grey rectangles for components of the Entity-Relationship schema, (3) the middle right part with dark grey rectangles for components of the Relational schema, and (4) the lower part with white rectangles for data types. We now discuss parts of that object diagram in detail. By doing this we explain how the respective data models and their transformation are represented in our metamodeling approach.

Data types: Both database schemata use two data types String and Integer. Correspondingly two DataType objects are introduced in the lower part of Figure 3. One DataType object possesses object identifier String within the USE system. This is visualized by writing down the identifier String in front of the colon in the upper part of the object rectangle. The value of the object's attribute name is also equal to String. Alternatively, we could have used a more abstract object identifier like for example dataTypeOne for that DataType object and dataTypeTwo for the other DataType object which in Figure 3 now has object identifier Integer. Then these identifiers dataTypeOne and dataTypeTwo would appear in front of colons in the upper part of object rectangles, but still these objects would represent the data types String and Integer, because the name attribute value would indicate this. We have preferred to have this strong correspondence between the name attributes values of the objects and its object identifiers but we are not forced to do so, because from a formal viewpoint there is no compelling relationship between the name attribute value and the object identifier. However, we will proceed analogously in having a naming correspondence as far as possible between object identifiers and name attribute values of objects.

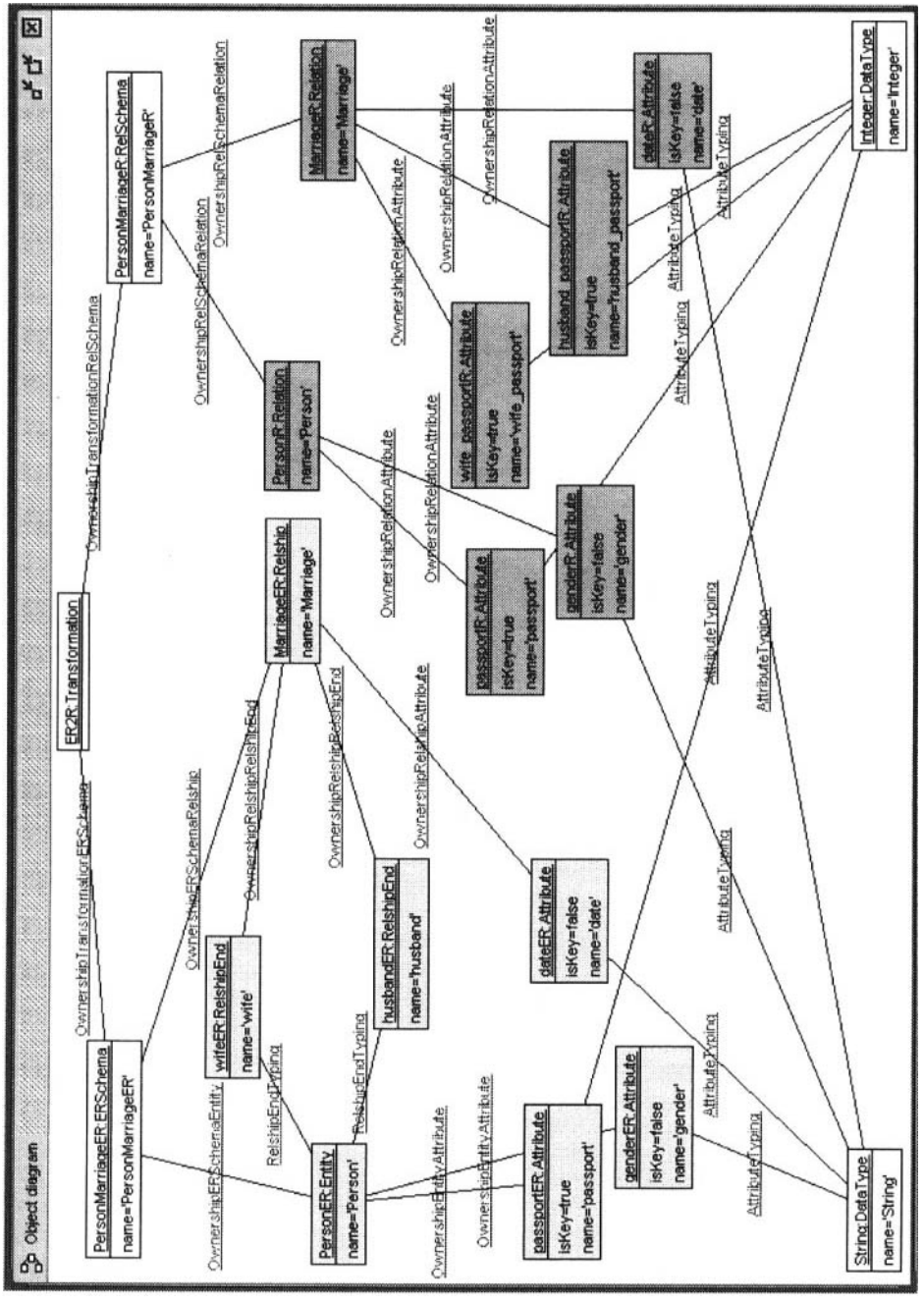


Figure 3: Example for the Formal Transformation

Entity-Relationship schema components: The light grey objects in the left part of Figure 3 represent components of the Entity-Relationship schema. We see one Entity object PersonER with name attribute value Person and one Relship object MarriageER with name attribute value Marriage. Here, we have chosen object identifiers close to, but not identical to name attribute values, because in the Relational schema to be explained below we will have components with the same name attribute values but this time being objects of class Relation. They will have object identifiers PersonR and MarriageR. Object identifiers in the Entity-Relationship part will always end with ER and those in the Relational part with R. Entity and Relship objects are connected by two RelshipEnd objects wifeER and husbandER representing role names of the relationship. In addition, three Attribute objects passportER, genderER, and dateER represent the two attributes of the Entity object and the single attribute of the Relship object, respectively. The isKey attribute value for passportER indicates that this attribute is a key attribute. Links to the above explained DataType objects constitute the attribute types.

Relational schema components: The dark grey objects in the right part of Figure 3 constitute components of the Relational schema. This Relational schema has two Relation objects with object identifiers PersonR and MarriageR and name attribute values Person and Marriage, respectively. The relation with name Person possesses two attributes with names passport and gender, and the relation with name Marriage has three attributes with names wife_passport, husband_passport, and date. As indicated by the isKey attribute values, the key for the relation named Person consists of the passport attribute, and the key of the relation named Marriage is made of attributes named wife_passport and husband_passport. As in the Entity-Relationship schema, the attribute typing is provided by links to DataType objects.

Transformation, ER and Relational schema: In the upper part of Figure 3, two white schema object rectangles represent the ER and the Relational schema. Both are connected with links to its direct components: The ERSchema object is linked to Entity and Relship objects and the RelSchema object to two Relation objects. The Transformation object is connected to both of these schema objects.

5 Constraints

As a central description ingredient, the metamodel is restricted by a number of OCL constraints in order to establish correct system states, for example, to guarantee that Entity-Relationship and Relational schemata are represented correctly. Without such constraints undesirable system states would be allowed. To mention only one example, without constraints it would be possible that an Entity object is connected to a RelshipEnd and to an Attribute object where all three objects have the same value for their name attribute.

5.1 Name Constraints

One category of constraints concerns naming of objects, i.e. these constraints restrict values of the name attribute. There is a distinction between globally unique names and locally unique names. For example, names of Entity-Relationship schemata must be globally unique which

means that two distinct Entity-Relationship schemata also generally must have distinct name attribute values. Formally, this is notated as:

```
context ERSchema inv uniqueERSchemaNames:
  ERSchema.allInstances->forAll(s1,s2|
    s1.name=s2.name implies s1=s2)
```

The implication could have alternatively been written the other way round emphasizing distinctness and using inequations instead of equations.

```
context ERSchema inv uniqueERSchemaNames:
  ERSchema.allInstances->forAll(s1,s2|
    s1<>s2 implies s1.name<>s2.name)
```

Locally unique names mean that object names are unique within a certain local namespace or context. This is required, for example, for entities within an Entity-Relationship schema. It means that two distinct entities belonging to the same Entity-Relationship schema must have distinct names, but in contrast to globally unique names it is allowed that two distinct entities with the same name exist if they come from two distinct Entity-Relationship schemata. Formally this is expressed as:

```
context ERSchema inv uniqueEntityNamesWithinSchema:
  entity->forAll(e1,e2|e1.name=e2.name implies e1=e2)
```

Constraints concerning globally unique names use the OCL construct `allInstances` retrieving for a class all current objects of that class. The other constraint for globally unique names is:

```
context RelSchema inv uniqueRelSchemaNames:
  RelSchema.allInstances->forAll(s1,s2|
    s1.name=s2.name implies s1=s2)
```

As a convention, we let names of constraints concerning locally unique names always include the word `Within` and indicate the namespace after `Within`. Other constraints for locally unique names are:

```
context ERSchema inv uniqueRelshipNamesWithinSchema:
  relship->forAll(r1,r2|r1.name=r2.name implies r1=r2)

context Relship inv uniqueRelshipEndNamesWithinRelship:
  relshipEnd->forAll(re1,re2|
    re1.name=re2.name implies re1=re2)

context Entity inv uniqueAttributeNamesWithinEntity:
  attribute->forAll(a1,a2|a1.name=a2.name implies a1=a2)

context Relship inv uniqueAttributeNamesWithinRelship:
  attribute->forAll(a1,a2|a1.name=a2.name implies a1=a2)
```

```
context RelSchema inv uniqueRelationNamesWithinSchema:
  relation->forAll(r1,r2|r1.name=r2.name implies r1=r2)

context Relation inv uniqueAttributesNamesWithinRelation:
  attribute->forAll(a1,a2|a1.name=a2.name implies a1=a2)
```

A third group of constraints concerning naming requires that components within one schema but different in nature must have different names. For example, entities and relationships within one Entity-Relationship schema must possess different names.

```
context ERSchema
  inv differentEntityAndRelshipNamesWithinSchema:
  entity->forAll(e|relship->forAll(r|e.name<>r.name))

context Entity
  inv differentAttributeAndRelshipEndNamesWithinEntity:
  relshipEnd->forAll(re|attribute->forAll(a|re.name<>a.name))

context Relship
  inv differentAttributeAndRelshipEndNamesWithinRelship:
  relshipEnd->forAll(re|attribute->forAll(a|re.name<>a.name))
```

5.2 Constraints Concerning Keys

This group of constraints concerns requirements for **isKey** attribute values and **key()** operation values. **Entity** and **Relation** objects must have a non-empty key. **Relship** objects are not allowed to have key attributes.

```
context Entity inv entityKeyNotEmpty:
  key()->notEmpty

context Relship inv relshipKeyEmpty:
  attribute->select(a|a.isKey)->isEmpty

context Relation inv relationKeyNotEmpty:
  key()->notEmpty
```

5.3 Attribute Ownership and Commutativity Constraints

Class **Attribute** participates in three associations **OwnershipEntityAttribute**, **OwnershipRelshipAttribute**, and **OwnershipRelationAttribute**. But an **Attribute** object must belong to either an **Entity** object or a **Relship** object or a **Relation** object, i.e. an **Attribute** object must belong to exactly one of these three schema components.

```
context Attribute
  inv attributeOwnedByEntityXorRelshipXorRelation:
  (entity->size + relship->size + relation->size) = 1
```

The next constraint concerns a commutativity property in the Entity-Relationship part of the schema. One must require that, if navigating from a **RelshipEnd** object via association **RelshipEndTyping** to the **Entity** class and from there via association **OwnershipSchemaEntity** to the the **ERSchema** class, this results in the same **ERSchema** object as when going the other way via association **OwnershipRelshipRelshipEnd** and association **OwnershipSchemaRelship** through class **Relship**.

```
context RelshipEnd inv commutativityRelshipEnd:
  entity.erSchema=relship.erSchema
```

This constraint **commutativityRelshipEnd** implies that other commutativity requirements also hold. Thus we have not to require explicitly that the following invariants will be true.

```
context Entity inv commutativityEntity:
  -- erSchema=relshipEnd.relship.erSchema
  relshipEnd->forAll(re|re.relship.erSchema=erSchema)
context Relship inv commutativityRelship:
  -- erSchema=relshipEnd.entity.erSchema
  relshipEnd->forAll(re|re.entity.erSchema=erSchema)
```

5.4 Transformation Constraints

The transformation invariants are the most challenging expressions in the specification. These invariants can be used in the first place to verify a transformation, but it is in principle also possible to try to systematically derive concrete transformation classes or transformation operations from these constraints. This is subject to future work.

Recall that transformation of Entity-Relationship schemata into Relational schemata is represented by class **Transformation**. This class has no attributes, but participates in two associations: One **Transformation** object is linked to exactly one **ERSchema** object and one **RelSchema** object. The purpose of **Transformation** invariants is to guarantee that the **RelSchema** object is the result of transforming the **ERSchema** object into the Relational data model. The transformation is restricted by these three invariants: `relationExistsForEntity`, `relationExistsForRelship`, and `entityXorRelshipExistsForRelation`. As the names indicate, the first two are responsible for the task that for each entity and each relationship in the Entity-Relationship schema there exists a corresponding relation in the Relational schema, and the last invariant assures that for each relation there is an entity or a relationship. The context for all invariants is a transformation, i.e. one Entity-Relationship schema and one Relational schema.

```
context Transformation inv relationExistsForEntity:
   erSchema.entity->forAll(e| relSchema.relation->exists(rl|
(A)    e.name=rl.name and
(B)    e.attribute->forAll(ea| rl.attribute->exists(ra|
          ea.name=ra.name and
          ea.type=ra.type and
          ea.isKey=ra.isKey))))
```

Invariant relationExistsForEntity requires that for each entity e in the considered Entity-Relationship schema there is a relation rl in the Relational schema such that (A) entity e and relation rl have the same name and (B) for each attribute ea of the entity there is an attribute ra in the relation having the same name, the same type, and the same key property. As a forward reference, we remark that the requirement that relation rl has no more attributes will be guaranteed by the invariant entityXorRelshipExistsForRelation together with naming restriction that an attribute name is unique within a relation.

```
context Transformation inv relationExistsForRelship:
  erSchema.relship->forAll(rs|
    relSchema.relation->exists(rl|
(A)   rs.name=rl.name and
(B)   rs.relshipEnd->forAll(rse|
        rse.entity.key()->forAll(rsek|
         rl.attribute->exists(ra|
           rse.name.concat('_').concat(rsek.name)=ra.name and
           rsek.type=ra.type and
           ra.isKey=true))) and
(C)   rs.attribute->forAll(rsa| rl.attribute->exists(ra|
        rsa.name=ra.name and
        rsa.type=ra.type and
        ra.isKey=false))))
```

Invariant relationExistsForRelship asserts that for each relationship rs in the Entity-Relationship schema there is a relation rl in the Relational schema such that (A) relationship rs and relation rl have identical names, (B) for each relationship end there are appropriate attributes in the relation, and (C) for each relationship attribute there is an attribute in the relation. Subrequirement (B) means, that for each relationship end rse, any key attribute rsek of the referenced entity appears in the relation, and the name of the attribute in the relation consists of the relationship end name followed by underscore followed by the Entity-Relationship name of the key attribute. Subrequirement (C) says, that each attribute rsa of the relationship appears under the same name and the same type as a non-key attribute in the relation.

```
context Transformation inv entityXorRelshipExistsForRelation:
  relSchema.relation->forAll(rl|
    erSchema.entity->exists(e|
(A)   rl.name=e.name and
(B)   rl.attribute->forAll(ra| e.attribute->exists(ea|
        ra.name=ea.name and
        ea.type=ra.type and
        ra.isKey=ea.isKey)))
    xor
    erSchema.relship->exists(rs|
(C)   rl.name=rs.name and
      rl.attribute->forAll(ra|
```

```
(D)     rs.relshipEnd->exists(rse|
         rse.entity.key()->exists(rsek|
(E)        ra.name=rse.name.concat('_').concat(rsek.name) and
(F)        ra.type=rsek.type and
(G)        ra.isKey=true))
        xor
(H)     rs.attribute->exists(rsa|
          ra.name=rsa.name and
          ra.type=rsa.type and
          ra.isKey=false))))
```

Invariant `entityXorRelshipExistsForRelation` assures that each relation `rl` in the Relational schema either has a corresponding entity or relationship in the Entity-Relationship schema but not both. If a relation `rl` corresponds to an entity `e`, then (A) the name of the relation and entity coincide and (B) for each relation attribute `ra` there is a corresponding entity attribute `ea` with the same name, type, and key property. If a relation `rl` corresponds to a relationship `rs`, then (C) both have the same name and *EITHER* (D) a relation attribute `ra` comes from a relationship end `rse` such that (E) the name of the attribute `ra` is equal to the name of the relationship end `rse` followed by underscore followed by the name of a key attribute `rsek` of the referenced entity of the relationship end, (F) the attribute `ra` has the same type as key attribute `rsek`, and (G) attribute `ra` is part of the key of relation `rl`, *OR* (H) a relation attribute `ra` comes from a relationship attribute `rsa` with the same name, type, and attribute `ra` is a non-key attribute in the relation.

The above invariants can be used in the first place to verify a transformation. Similar work on verification of integrating ER diagrams has been done in [22]. But in principle, it should also be possible to systematically derive concrete transformation classes or transformation operations from these constraints.

5.5 USE Screenshot on Invariants

Figure 4 shows a screenshot from a USE session. It displays the explained 20 invariants. That screenshot was taken after the complete schemata and the transformation object have been established. We see that all invariants are satisfied, i.e. the result of evaluating each invariant is true. This means that all invariants are valid in the current system state. Or in other words, the Relational schema is the correct transformation of the Entity-Relationship schema. As a side remark concerning functionality of USE, we note that, if an invariant is false, USE supports failure analysis by an evaluation browser window which allows to trace which part of the current system state and which part of the respective invariant contribute to the undesired result.

6 Comparison of ER and Relational Data Model

The exemplary object diagram in Figure 3 also gives rise to a short, but substantial comparison of the Entity-Relationship and Relational data model. Both the Entity-Relationship and the Relational schema consist of seven objects:

1 Entity + 1 Relship + 2 RelshipEnds + 3 Attributes = 7 Objects (ER)

2 Relations + 5 Attributes = 7 Objects (Rel)

Figure 4: USE Screenshot with Invariant Evaluation

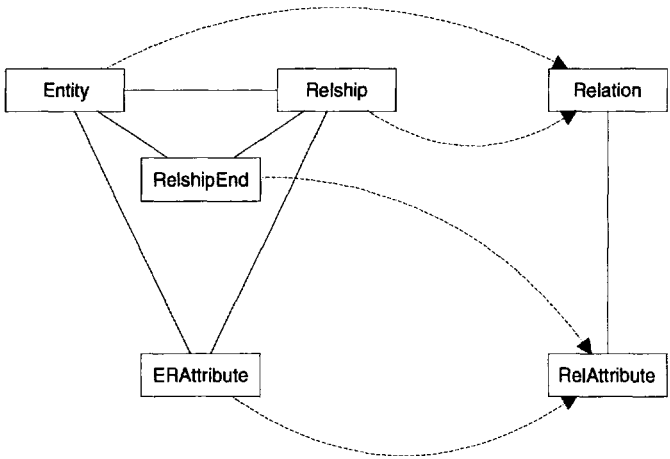


Figure 5: Transformation of Entity-Relationship Concepts into Relational Concepts

The single **Entity** object and the single **Relship** object correspond to two **Relation** objects. The two **RelshipEnd** and the three Entity-Relationship **Attribute** objects correspond to the five Relational **Attribute** objects. Figure 5 gives an overview by showing the data model concepts, their associations within the respective data model, and their transformation. The simple enumeration shows in exemplary form richness of concepts in the ER model with four concepts, namely **Entity**, **Relship**, **RelshipEnd**, and **Attribute**, in comparison to the Relational model with only two concepts, namely **Relation** and **Attribute**. The example also points out that **Entity** and **Relship** objects are transformed into **Relation** objects, as well as that **RelshipEnd** and Entity-Relationship **Attribute** objects are transformed into Relational **Attribute** objects.

The comparison in Figure 5 also indicates how an equivalent alternative to the present transformation could look like. One could divide the general **Transformation** class into specialized **Transformation** classes. Then one would introduce **Transformation** classes establishing (Entity,Relation)-, (Relship,Relation)-, (RelshipEnd,Attribute)-, and (Attribute,Attribute)-pairs. The general **Transformation** class could contain these specialized **Transformations** as components. The difference to the present solution is that then the concept mapping (which in the present solution is expressed in the `exists` and `forAll` quantifiers in the OCL expressions) becomes explicit.

Yet another alternative being also equivalent would be to have an operation with name **transform2rel():RelSchema** in class **ERSchema** instead of class **Transformation**. The requirements which are now stated as invariants for class **Transformation** would become postconditions for that operation. We have chosen an approach with invariants in order to emphasize that the transformation can be given in a purely descriptive way.

7 Conclusion

In this chapter we have introduced a formal MOF modeling for the Entity-Relationship and the Relational model as well as the transformation between these data models. All steps have been formally specified and validated with the USE system which has been employed here as a MOF compliant tool. Our approach allows to concentrate the central ideas of the Entity-Relationship and the Relational data model and their transformation into a single diagram having a precise formal semantics.

Other language features of the Entity-Relationship and the Relational data model like, for example, foreign keys have to be studied further. We feel however that such extensions can be handled within the MOF and OCL language elements due to the flexible and powerful possibilities offered. We will also consider other data models like object-oriented or semi-structured models and data models important in the context of the Semantic Web. Due to the modeling power of UML including OCL, these data models and their translation can be handled in the same way as we did here. Thus the treatment of the Entity-Relationship and Relational model can be considered as a feasibility study for the general approach which involves the description of data models and their transformation in an object-oriented and descriptive way. Further attention will also be paid to include in addition information system states already on the metamodel layer. First steps in this direction have already been studied in [23].

Acknowledgements

Thanks to Heiner Stuckenschmidt for commenting a draft version of this chapter. The contribution of Mark Richters and Paul Ziemann in other papers on similar topics is gratefully acknowledged.

References

[1] OMG, ed.: OMG Unified Modeling Language Specification, Version 1.4. OMG (2001) OMG Document `formal/01-09-68` through `formal/01-09-80`, `www.omg.org`.

[2] OMG, ed.: Meta Object Facility (1.3). OMG (1999) `www.omg.org`.

[3] Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (1998)

[4] Richters, M., Gogolla, M.: On Formalizing the UML Object Constraint Language OCL. In Ling, T.W., Ram, S., Lee, M.L., eds.: Proc. 17th Int. Conf. Conceptual Modeling (ER'98), Springer, Berlin, LNCS 1507 (1998) 449–464

[5] Richters, M., Gogolla, M.: Validating UML Models and OCL Constraints. In Evans, A., Kent, S., eds.: Proc. 3rd Int. Conf. Unified Modeling Language (UML'2000), Springer, Berlin, LNCS 1939 (2000) 265–277

[6] OMG, ed.: The Common Warehouse Metamodel Specification. OMG (2000) `www.omg.org`.

[7] Vetterli, T., Vaduva, A., Staudt, M.: Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. SIGMOD Record **29** (2000) 68–75

[8] Clark, T., Evans, A., Kent, S.: The Metamodelling Language Calculus: Foundation Semantics for UML. In Hussmann, H., ed.: Proc. Int. Conf. FASE, Springer, LNCS (2001) 17–31

[9] Ebert, J., Süttenbach, R.: An OMT Metamodel. Technical report, University Koblenz-Landau, Fachbericht Informatik 13-97 (1997)

[10] Ebert, J., Süttenbach, R.: A Booch Metamodel. Technical report, University Koblenz-Landau, Fachbericht Informatik 5-97 (1997)

[11] Henderson-Sellers, B., Firesmith, D., Graham, I.M.: OML Metamodel: Relationships and State Modeling. Journal Of Object-Oriented Programming **10** (1997) 47–51

[12] Atzeni, P., Torlone, R.: A metamodel approach for the management of multiple models and translation of schemes. Information Systems **18** (1993) 349–362

[13] Misic, V.B., Moser, S.: A Formal Approach to Metamodeling: A Generic Object-Oriented Perspective. In: Int. Conf. Entity-Relationship Approach (ER'97). (1997) 243–256

[14] Iyengar, S.: Distributed object repositories: Concepts and standards. In Embley, D.W., Goldstein, R.C., eds.: Proc. 16th Int. Conf. Conceptual Modeling (ER'97), Springer, Berlin, LNCS 1331 (1997) 85–101

[15] Blaha, M., LaPlant, D., Marvak, E.: Requirements for repository software. In: Working Conf. on Reverse Engineering, IEEE Computer Society (1998) 164–173

[16] Terrasse, M.N., Savonnet, M., Becker, G.: A UML-based Metamodeling Architecture for Database Design. In: Proc. IDEAS'2001. (2001) 231–236

[17] Miura, T., Matsumoto, W., Shioya, I.: Managing Meta Objects for Design of Warehouse Data. In: Proc. DaWaK'1999. (1999) 33–40

[18] Tan, J., Zaslavsky, A.B., Bond, A.: Meta Object Approach to Database Schema Integration. In: Proc. DOA'2000. (2000) 145–154

[19] McClatchey, R., Kovacs, Z., Estrella, F., Goff, J.M.L., Varga, L., Zsenei, M.: The Role of Meta-Objects and Self-Description in an Engineering Data Warehouse. In: Proc. IDEAS'1999. (1999) 342–350

[20] Cranefield, S.: UML and the Semantic Web. In: Proc. Int. Semantic Web Working Symposium (SWWS). (2001)

[21] Bowers, S., Delcambre, L.: Representing and transforming model-based information. In: Proc. Int. Workshop on the Semantic Web at the 4th European Conference on Research and Advanced Technology for Digital Libraries (SemWeb). (2000)

[22] Franconi, E., Ng, G.: The i.com Tool for Intelligent Conceptual Modelling. In: Proc. 7th Intl. Workshop on Knowledge Representation meets Databases (KRDB'00). (2000)

[23] Gogolla, M., Lindow, A., Richters, M., Ziemann, P.: Metamodel Transformation of Data Models. In Bezivin, J., France, R., eds.: Proc. UML'2002 Workshop in Software Model Engineering (WiSME 2002). Technical Report, University of Nantes, `http://www.metamodel.com/wisme-2002` (2002)

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

On Modeling Conformance for Flexible Transformation over Data Models

Shawn Bowers Lois Delcambre

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

Abstract. Data models use a limited number of *basic structures* to represent data such as collections, attribute-value pairs, and scalars. They differ, however, in how structures are composed and whether they *permit* the specification of schema, permit *more than one schema*, or *require* schema. Support for transforming between heterogeneous representation schemes remains a significant challenge. To this aim, we extend our 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 through mapping 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 transformation.

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.

In this chapter, 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.

The rest of this chapter 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]. 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 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 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

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: 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

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.

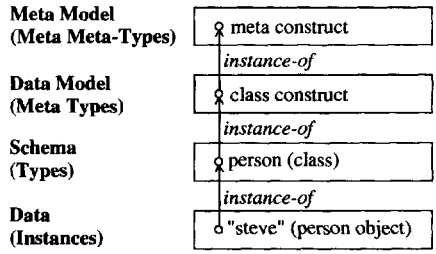


Figure 1: A typical four-level metamodel architecture.

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. 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 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).

A *configuration* 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

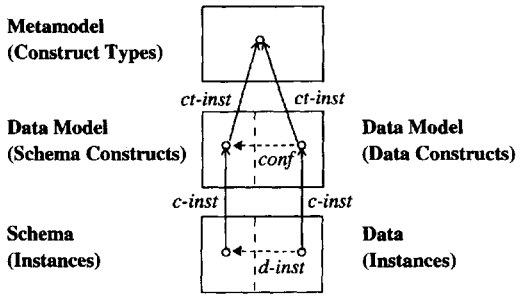


Figure 2: The three-level metamodel architecture.

configuration.

A configuration 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.

- 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 a value 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.

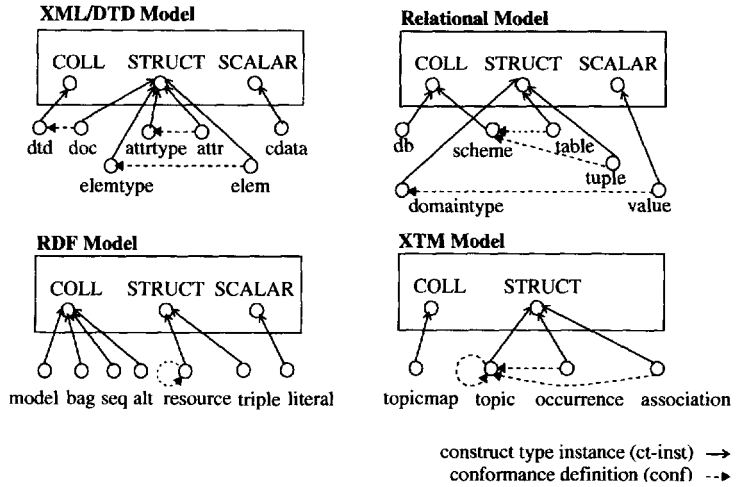


Figure 3: Relational, RDF, and XTM data models described within the generic framework.

3.2 Example Description Language and Basic Structures

Here we introduce 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 the metamodel are collection, (specifically, set_{ct}, $list_{ct}$, bag_{ct}), $struct_{ct}$, and $scalar_{ct}$, 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 chapter. 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. The $ct\text{-}inst(c, ct)$ predicate relates a construct c to a construct type ct, which must be either set_{ct}, $list_{ct}$, bag_{ct}, $struct_{ct}$, or $scalar_{ct}$. The $conf(c_1, c_2)$ predicate specifies that a construct c_1 can conform to another construct c_2. The $c\text{-}inst(d, c)$ predicate relates an instance d with a construct c. Finally, the $d\text{-}inst(d_1, d_2)$ predicate defines a conformance relationship between instances d_1 and d_2.

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

$comp(c_1, setof(c_1, \ldots, c_n))$
$comp(c_2, listof(c_1, \ldots, c_n))$
$comp(c_3, bagof(c_1, \ldots, c_n))$
$comp(c_4, structof(a_1 \rightarrow c_1, \ldots, a_n \rightarrow c_n))$
$comp(c_5, union(c_1, \ldots, c_n))$

$ct\text{-}inst(dtd, set_{ct})$	$comp(dtd, setof(elem\text{-}t))$
$ct\text{-}inst(elem\text{-}t, struct_{ct})$	$comp(elem\text{-}t, structof(tag \rightarrow string, attrtypes \rightarrow attr\text{-}tset))$
$ct\text{-}inst(attr\text{-}t, struct_{ct})$	$comp(attr\text{-}t, structof(name \rightarrow string))$
$ct\text{-}inst(subelem\text{-}t, struct_{ct})$	$comp(subelem\text{-}t, structof(parent \rightarrow elem\text{-}t, child \rightarrow elem\text{-}t))$
$ct\text{-}inst(attr\text{-}tset, set_{ct})$	$comp(attr\text{-}tset, setof(attr\text{-}t))$
$ct\text{-}inst(doc, struct_{ct})$	$comp(doc, structof(root \rightarrow elem))$
$ct\text{-}inst(elem, struct_{ct})$	$comp(elem, structof(tag \rightarrow string, attrs \rightarrow attrset))$
$ct\text{-}inst(attr, struct_{ct})$	$comp(attr, structof(name \rightarrow string, val \rightarrow cdata))$
$ct\text{-}inst(attrset, set_{ct})$	$comp(attrset, setof(attr))$
$ct\text{-}inst(nestedelem, struct_{ct})$	$comp(nestedelem, structof(parent \rightarrow elem, child \rightarrow node))$
$ct\text{-}inst(pcdata, scalar_{ct})$	$ct\text{-}inst(cdata, scalar_{ct})$
$comp(node, union(elem, pcdata))$	$conf(doc, dtd)$
$conf(elem, elem\text{-}t)$	$conf(attr, attr\text{-}t)$

Figure 4: Simplified description of the XML with DTD data model.

$ct\text{-}inst(db, set_{ct})$	$comp(db, setof(scheme, table))$
$ct\text{-}inst(scheme, set_{ct})$	$comp(scheme, setof(fieldtype))$
$ct\text{-}inst(table, struct_{ct})$	$comp(table, structof(name \rightarrow string, rows \rightarrow tuples))$
$ct\text{-}inst(tuples, bag_{ct})$	$comp(tuples, bagof(tuple))$
$ct\text{-}inst(tuple, set_{ct})$	$comp(tuple, setof(field))$
$ct\text{-}inst(fieldtype, struct_{ct})$	$comp(fieldtype, structof(name \rightarrow string, dom \rightarrow domaintype))$
$ct\text{-}inst(field, struct_{ct})$	$comp(field, structof(name \rightarrow string, val \rightarrow value))$
$ct\text{-}inst(domaintype, struct_{ct})$	$comp(domaintype, structof(name \rightarrow string))$
$ct\text{-}inst(value, scalar_{ct})$	$conf(taple, scheme)$
$conf(tuple, scheme)$	$conf(value, domaintype)$

Figure 5: Simplified description of the Relational data model.

As shown, each kind of *comp* formula uses an additional formula to specify composition constraints, namely, *setof*, *listof*, *bagof*, or *structof*. For example, c_1 is assumed to be defined as $ct\text{-}inst(c_1, set_{ct})$ whose instances are sets that can contain instances of constructs $c_1, c_2, \ldots, c_n$ (*i.e.*, the *setof* predicate defines the homogeneity of the construct's instances). We also permit the definition of convenience-constructs (as shown by c_5 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.

$val(d_1, set(v_1, \ldots, v_n))$
$val(d_2, list(v_1, \ldots, v_n))$
$val(d_3, bag(v_1, \ldots, v_n))$
$val(d_4, struct(a_1 = v_1, \ldots, a_n = v_n))$

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.

$ct\text{-}inst(tm, set_{ct})$
$ct\text{-}inst(assoc, struct_{ct})$
$ct\text{-}inst(topic, struct_{ct})$
$ct\text{-}inst(assoc\text{-}mem, struct_{ct})$
$ct\text{-}inst(topic\text{-}occ, struct_{ct})$
$ct\text{-}inst(occurrence, struct_{ct})$
$ct\text{-}inst(member, struct_{ct})$
$ct\text{-}inst(mem\text{-}topic, struct_{ct})$
$ct\text{-}inst(resource, struct_{ct})$
$comp(occ\text{-}val, union(resource, string))$
$conf(assoc, topic)$

$comp(tm, setof(topic, assoc))$
$comp(assoc, structof())$
$comp(topic, structof(name \rightarrow string))$
$comp(assoc\text{-}mem, structof(ac \rightarrow assoc, mem \rightarrow member))$
$comp(topic\text{-}occ, structof(top \rightarrow topic, occ \rightarrow occurrence))$
$comp(occurrence, structof(val \rightarrow occ - val))$
$comp(member, structof(role \rightarrow string))$
$comp(mem\text{-}topic, structof(mem \rightarrow member, top \rightarrow topic))$
$comp(resource, structof(uri \rightarrow string))$
$conf(topic, topic)$
$conf(occurrence, topic)$

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

$ct\text{-}inst(rdfmodel, set_{ct})$
$ct\text{-}inst(uri, struct_{ct})$
$ct\text{-}inst(blank, struct_{ct})$
$ct\text{-}inst(alt, set_{ct})$
$ct\text{-}inst(seq, list_{ct})$
$ct\text{-}inst(bag, bag_{ct})$
$ct\text{-}inst(triple, struct_{ct})$
$ct\text{-}inst(literal, scalar_{ct})$
$comp(coll, union(set, bag, alt))$
$conf(resource, resource)$

$comp(rdfmodel, setof(literal, blank, triple, coll))$
$comp(uri, structof(ref \rightarrow string))$
$comp(blank, structof())$
$comp(alt, setof(node))$
$comp(seq, listof(node))$
$comp(bag, bagof(node))$
$comp(triple, structof(pred \rightarrow uri, subj \rightarrow resource, obj \rightarrow node))$
$comp(node, union(resource, literal))$
$comp(resource, union(uri, blank, coll, triple))$

Figure 7: Simplified description of the RDF data model.

(a).

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

(b).

```
<activity>
<profession>
<professor>Maier</professor>
</profession>
</activity>
```

(c).

$c\text{-}inst(t, dtd)$
$c\text{-}inst(t1, elem\text{-}t)$
$c\text{-}inst(t2, elem\text{-}t)$
$c\text{-}inst(t3, elem\text{-}t)$
$c\text{-}inst(s1, subelem\text{-}t)$
$c\text{-}inst(s2, subelem\text{-}t)$
$c\text{-}inst(d, doc)$
$c\text{-}inst(e1, elem)$
$c\text{-}inst(e2, elem)$
$c\text{-}inst(e3, elem)$
$c\text{-}inst(n1, nestedelem)$
$c\text{-}inst(n2, nestedelem)$
$c\text{-}inst(n3, nestedelem)$
$c\text{-}inst(d, t)$
$d\text{-}inst(e1, t1)$
$d\text{-}inst(e3, t3)$

$val(t, set(et1, et2, et3))$
$val(t1, struct(name =$"activity"$))$
$val(t2, struct(name =$ "profession" $))$
$val(t3, struct(name =$ "professor" $))$
$val(s1, struct(parent = t1, child = t2))$
$val(s2, struct(parent = t2, child = t3))$
$val(d, struct(root = e1))$
$val(e1, struct(tag =$ "activity" $))$
$val(e2, struct(tag =$ "profession" $))$
$val(e3, struct(tag =$ "professor" $))$
$val(n1, struct(parent = e1, child = e2))$
$val(n2, struct(parent = e2, child = e3))$
$val(n3, struct(parent = e3, child =$ "Maier" $))$
$c\text{-}inst($"Maier"$, pcdata)$
$d\text{-}inst(e2, t2)$

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

3.3 Specifying Constraints

In addition to describing composition of basic structures, we can 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 *constrain*, *e.g.*, the following rule defines a conformance constraint over the Relational model.

$constrain : d\text{-}inst(x,t) \leftarrow$
$\quad c\text{-}inst(x,\text{table}), c\text{-}inst(y,\text{scheme}),$
$\quad \neg\, d\text{-}inst(x,x_s), \neg\, d\text{-}inst(y_t,y),$
$\quad \neg\, non\text{-}conform(x,y).$
$non\text{-}conform(x,y) \leftarrow$
$\quad val(x,b), member(r,b), \neg d\text{-}inst(r,y).$

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 [2].

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 $\sigma : M \times C \rightarrow C'$ where M is a set of *mapping rules*, C is the source configuration, and 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:

$$dp_1, \ldots, dp_n \leftarrow p_1, \ldots, p_m,$$

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.[1] 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):

$$head(dp_1, \ldots, dp_n) \leftarrow p_1, p_2, \ldots, p_m.$$

[1] We use the abbreviation $id(v_1, v_2, \ldots, id)$ to denote the list $[v_1, v_2, \ldots, v_n]$.

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

Figure 9: Rules for a model-to-model transformation.

Mapping rules are read in the normal way, *i.e.*, if each p_i for $1 \leq i \leq n$ is true, then each 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.

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 called "modeling the model" [3]), and various mixtures of each [4, 5].

```
% 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(tag = _, 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)),
val(FT2, struct(name = CN, dom = stringtype)) ←
  c-inst(ST, subelem-t), val(ST, struct(parrent = P, child = C),
  val(P, struct(tag = PN, attrtypes = _), val(C, struct(tag =CN, attrtypes = _),
  id(P, FT1), id(C, FT2).
% create a table element type
add : c-inst(R, tuples), val(R, bag()), val(T, struct(name = N, rows = R)) ←
  c-inst(ET, elem-t), val(ET, struct(tag = 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).
...
```

Figure 10: Structural rules for a model-to-model transformation.

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.

For example, there are often many ways to transform between models allowing multiple

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

<professor dept="CSE">
<person name="Maier"/>
</professor>

(c).

Professor Table

id	*dept*
e1	"CSE"

Person Table

id	*name*
e2	"Maier"

Professor-Person Table

professor	*person*
e1	e2

Figure 11: Results from the XML to Relational transformation.

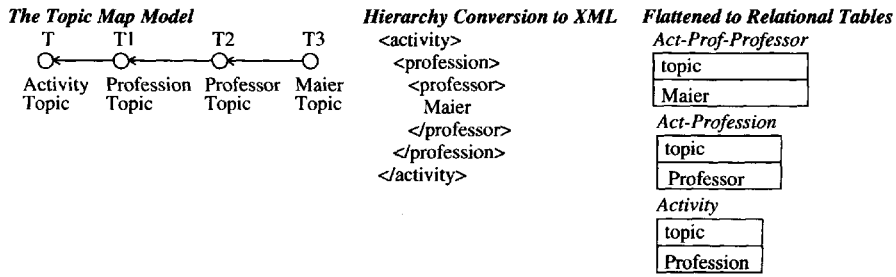


Figure 12: An example of flattening and hierarchy conversion of the Topic Map model.

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 "Maier" is an instance of topic "Professor"; topic "Professor" is an instance of topic "Profession"; and topic "Profession" is an instance of topic "Activity."[2] 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.

The following rule 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 exploring appropriate languages):

$pattern$: $hierarchy\text{-}convert(T, M_1, M_2, A_1, A_2)$ =
  % convert inter-nodes T and T1
  if $(d\text{-}inst(T_3, T_2), d\text{-}inst(T_1, T), c\text{-}inst(T, C), c\text{-}inst(T_1, C), c\text{-}inst(T_2, C))$ **then**
    map (using rule M_1) $T \rightarrow S$ **and** $T_1 \rightarrow S_1$ and $attach$ (using rule A_1) $S_1 \rightarrow S$
  % convert last node
  if $(d\text{-}inst(T_3, T_2), \neg d\text{-}inst(T_4, T_3), c\text{-}inst(T_2, C), c\text{-}inst(T_3, C), c\text{-}inst(T_4, C)$ **then**
    map (using M_1) $T_2 \rightarrow S_2$ **and** map (using M_2) $T_3 \rightarrow S_3$ **and** $attach$ (using A_2) $S_3 \rightarrow S_2$.

The pattern takes five arguments: the construct instance T serving as the top item of the

[2]Notice that "Maier" is an instance of a "Professor," but not an instance of a "Profession."

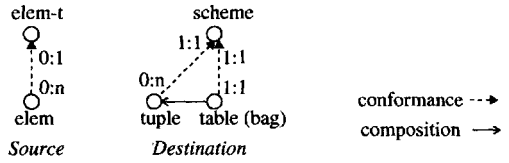


Figure 13: Source and destination constructs for rule derivation.

multi-level conformance relationship (*e.g.*, "Activity" in Figure 12); a mapping rule M_1 to map source items to target items (*e.g.*, topics to XML elements); a special mapping rule 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 A_1 and 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 M_1). The second rule is similar, but for leaf nodes. In our example, the second rule would convert the topic "Maier" into XML content (using M_2), and attach it to the "Professor" element (using A_2).

We see a number of potential patterns as useful, *e.g.*, a simple class of patterns are those that convert between construct 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 13. The following mapping rule specifies a construct transformation between XML element types and Relational schemes:

$$c\text{-}inst(ET, scheme) \leftarrow c\text{-}inst(ET, elem\text{-}t).$$

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 *table*'s conforming *scheme*, and (4) more than one *elem* can conform to an *elem-t*.

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* 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 such higher-level transformation rules further.

5 Related Work

This chapter extends our earlier work [4, 5], which introduced the use of a uniform, RDF-based representation scheme allowing model, schema, and instance data to be represented using triples. The metamodel introduced here is significantly more complex and has a relatively complete set of structural primitives, including typical database modeling constructs: *struct* (similar to tuple) and explicit collection primitives *set*, *bag*, and *list*. More importantly, these structures can be freely composed as needed for the data model being described. We also define a clear separation of basic structures (*i.e.*, construct types) from instance-of and conformance relationships—this separation differs from RDF, which has only a single version.

The metamodel architecture we propose is the only approach we know of that explicitly models conformance. Existing approaches [6, 7, 8, 9, 10] focus on solving database interoperability issues and assume (and exploit) schema-first models. Typically, transformation is defined at the data model or schema level (but not both). The approach we propose enables mappings at various 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 [9, 2], *e.g.*, WOL [2] is a schema transformation language that uses first-order logic rules for both schema transformations and specifying (schema) constraints.

Finally, a number of recent, informal model-to-model and model-to-schema mappings have been defined between XML and Relational models [11] and between Topic Maps and RDF [3, 12]. We believe having a formal tranformation language can benefit both defining and implementing such mappings.

6 Summary

The framework we propose has the following advantages. It allows a data model designer to explicitly specify when and how one construct conforms to another. 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 exploited for transformation and 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 data structures that exist in the information representation. We believe patterns and constraint-based transformation also offer powerful mechanisms for transformation and intend to further investigate their use.

Acknowledgement

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

References

[1] Marshall, C., Shipman, F., Coombs, J.: VIKI: Spatial hypertext supporting emergent structure. In: Proceedings of the European Conference on Hypertext Technology (ECHT'94), Edinburgh, Scotland, ACM (1994) 13–23

[2] Davidson, S., Kosky, A.: WOL: A language for database transformations and constraints. In: Proceedings of 13th International Conference on Data Engineering (ICDE'97), Birmingham, U.K. (1997) 55–65

[3] Moore, G.: RDF and Topic Maps: An exercise in convergence. In: Proceedings of the XML Europe Conference, Berlin, Germany (2001)

[4] Bowers, S., Delcambre, L.: Representing and transforming model-based information. In: Proceedings of the First International Workshop on the Semantic Web (SemWeb'00), held in conjunction with the 4th European Conference on Digital Libraries, Lisbon, Portugal (2000) 1–14

[5] Bowers, S., Delcambre, L.: A generic representation for exploiting model-based information. In: The Electronic Transactions on Artificial Intelligence (ETAI Journal). Volume 5. (2001, Section D) 1–34

[6] Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96). Volume 1057 of Lecture Notes in Computer Science., Avignon, France (1996) 79–95

[7] Barsalou, T., Gangopadhyay, D.: M(DM): An open framework for interoperation of multimodel multidatabase systems. In: Proceedings of the 8th International Conference on Data Engineering (ICDE'92). IEEE Computer Society Press, Tempe, Arizona, USA (1992) 218–227

[8] Christophides, V., Cluet, S., Siméon, J.: On wrapping query languages and efficient XML integration. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA (2000) 141–152

[9] McBrien, P., Poulovassilis, A.: A uniform approach to inter-model transformations. In: Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99). Volume 1626 of Lecture Notes in Computer Science., Heidelberg, Germany (1999) 333–348

[10] Papazoglou, M., Russell, N.: A semantic meta-modeling approach to schema transformation. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), Baltimore, Maryland, USA, ACM (1995) 113–121

[11] Bohannon, P., Freire, J., Roy, P., Siméon, J.: From XML schema to relations: A cost-based approach to XML storage. In: Proceedings of the 18th International Conference on Data Engineering (ICDE'02). IEEE Computer Society Press, San Jose, California, USA (2002) 64–75

[12] Lacker, M., Decker, S.: On the integration of Topic Maps and RDF. In: Proceedings of the 1st International Semantic Web Working Symposium (SWWS '01), California, USA (2001) 331–344

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

The 'Family of Languages' Approach to Semantic Interoperability

Jérôme Euzenat[1] Heiner Stuckenschmidt[2]

[1] *INRIA Rhône-Alpes*
655 avenue de l'Europe, Montbonnot Saint Martin
38334 Saint-Ismier, France
Jerome.Euzenat@inrialpes.fr

[2] *Vrije Universiteit Amsterdam*
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
heiner@cs.vu.nl

Abstract. Different Semantic Web applications can use different knowledge representation languages. Exchanging knowledge thus requires techniques for ensuring semantic interoperability across languages. We present the 'family of languages' approach based on a set of knowledge representation languages whose partial ordering depends on the transformability from one language to another by preserving a particular formal property such as logical consequence. For the same set of languages, there can be several such structures based on the property selected for structuring the family. Properties of different strength allow performing practicable but well founded transformations. The approach offers the choice of the language in which a representation will be imported and the composition of available transformations between the members of the family.

1 Motivation

The World Wide Web is the largest information system that ever existed. Its size and heterogeneity makes ontology-based search and integration even more important than in other information systems. The "Semantic Web" [1] 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, timelessness, heterogeneity of devices and adaptability), it is unlikely that this formal knowledge will be encoded in the same language. Thus, the interoperability of formal knowledge languages must be studied 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*. We will further characterize it below. There are several reasons for non interoperability and several approaches to semantic interoperability using different techniques [2, 3, 4]. In this chapter, we emphasize 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 [5] and the second one in the OIL language [6].

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 chapter first contrasts the 'family of languages' approach with other known approaches (Section 2). It then puts forth several structures for a 'family of languages' based on different properties (Section 3). We show that all these properties concur to semantic interoperability. Then, a concrete implementation of this approach is presented (Section 4) that we used to integrate languages in the example described above.

2 Approaches to language interoperability

We first give a few definitions of the kind of languages considered in this chapter. Then, we present several approaches for translating from one language to another.

2.1 Languages

For the purpose of the present chapter, 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 chapter 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 [7] or conceptual graphs [8]. In the following we give an abstract definition of such a language:

Example 1 (Abstract description language [9]) *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 an L-expression*
- *if δ is an L-expression, then $\neg\delta$ 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 an 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 [10]. This trade-off leads to identify different formalisms that are suited for different application scenarios. This also holds for ontology languages: there is no language that fits all situations. Several approaches have been proposed for ensuring semantic interoperability. We present them from 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 to corresponding expressions in the target language. The formal nature of these mappings vary from purely syntactic matches to "theory interpretations" [7] 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.

$$\exists \tau, (\forall r \subseteq L, \tau(r) \subseteq L') \tag{1}$$

The existence of a transformation τ from L to L' satisfying a property p is denoted by by $L \preceq_p L'$. In general, the property depends on the purpose of the transformation: change of language will require equivalence of meaning, importation will need preservation of consequences, simplification or abstraction will require preservation of models. Semantic properties will be considered more in depth in Section 3.

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. A current example of the mapping approach is described in [11].

2.3 The Pivot Approach

In order to reduce the number of transformations necessary to integrate 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. To preserve semantics, this pivot language has to be able to express all other languages; an adequate property p must then be used. More formally, the pivot approach is characterized by the following assumption:

$$\exists! P, \forall L, (L \preceq_p P) \tag{2}$$

Probably the most prominent example of a pivot architecture is Ontolingua [12]. In this approach the Ontolingua language serves as a pivot language.

However, the approach has often been criticized for information loss during translations from the pivot language to less expressive languages.

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 (denoted by a corresponding p property). 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, without semantic mismatch, into languages higher in the hierarchy. Formally speaking, the languages form a total order induced by the coverage relation.

$$\forall i, j, (i \leq j \Leftrightarrow L_i \preceq_p L_j) \tag{3}$$

A recent example of a layered architecture is the ontology language OIL [6] 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 should be possible to translate RDF schema definitions into languages of the higher levels in order to enrich it. Recently, the idea of layering different languages has been taken up again by the standardization committee for the web ontology language standard OWL.

2.5 The 'Family of Languages' Approach

The 'family of languages' approach, presented in this chapter, considers a set of languages structured by a partial order ($\preceq_p$). 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.

$$\forall L, L', \exists L'', (L \preceq_p L'' \wedge L' \preceq_p L'') \tag{4}$$

This equation is different from equation (2) because L'' is dependent on L and L'. In fact, the 'family of languages' 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 L" language for equation (4). However, the 'family of languages' approach is more flexible, because it does not require a fixed pivot language nor a fixed layering of languages. 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.

Consequence 1 *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 is 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.

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 a 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 family structure. It will then be easier to use this structure in transformations. The structure of a family of language is given by ordering this set with regard to available transformations satisfying some constraints (with the covering order $\preceq_p$).

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 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.

Again, this can be illustrated in the description logics framework.

Example 2 (Abstract description model [9]) *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 [9]) *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 (denoted as $\delta_1 \leq \delta_2$) if $\delta_1^{\Im,\mathcal{A}} \subseteq \delta_2^{\Im,\mathcal{A}}$.

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 in 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.

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 ($\preceq_p$ above) that can be established between two languages when a transformation from one to the other exists. In this section, the coverage relation will be characterized with regard to a property that it satisfies. The ultimate goal of these properties is 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 one 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 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)

$$L \preceq_{synt} L' \Leftrightarrow_{def} (L \subseteq L')$$

Language inclusion can be characterized 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 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 former proposal is restricted in the sense that it only allows, target languages that contain the source language, though there could be equivalent non-syntactically comparable languages. This applies to the description logic languages $\mathcal{ALC}$ and $\mathcal{ALCUE}$ which are known to be equivalent while none has all the constructors of the other[1]. 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*

$$\forall \delta \in L, \forall I, I(\tau(\delta)) = I(\delta)$$

In fact, there can be no other interpretations because of the requirement that the languages must be interpreted homogeneously.

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 [13]. 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[2]:*

$$\tau(R \leq (\texttt{domain}\, C)) = \top \leq (\texttt{all}\, (\texttt{inv}\, R)\, C)$$

For that purpose, one can define $L \preceq_{int} L'$ if and only if there exists a transformation from L to L' that preserves the interpretations of the expressions.

Definition 4 (Interpretation-based coverage)

$L \preceq_{int} L' \Leftrightarrow_{def}$ *there is an interpretation preserving transformation* $\tau : L \rightarrow L'$

[1] 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.

[2] 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.

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* $L' \preceq_{synt} L$ *then* $L' \preceq_{int} L$.

The τ transformation is, in general, not easy to produce (and it can generally be computationally expensive) but we have shown, in [14], how this can 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 [15] has provided a definition of expressiveness of a first-order knowledge representation language in 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 with the same models up to predicate renaming.

His definitions is based on the idea of "abstract models" in which a language is a pair 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 $\mathcal{M}_L(L)$.

Definition 5 (Expressibility modulo renaming [15]) *A language L is expressible in a language L' if and only if $\forall r \subseteq L$, $\exists$ a transformation $\tau : L \rightarrow L'$, $\exists \nu : Pred(r) \rightarrow Pred(\tau(r))$ such that $\forall m \in \mathcal{M}_L(r), \exists m' \in \mathcal{M}_{L'}(\tau(r)); \forall \delta \in L, m(\delta) = m'(\nu(\delta))$ and $\forall m' \in \mathcal{M}_{L'}(\tau(r)), \exists m \in \mathcal{M}_L(r); \forall \delta \in L, m(\delta) = m'(\nu(\delta))$. $Pred(r)$ is the set of atomic terms T found in the expression r.*

The notion of expressibility modulo renaming can best be explained by an example:

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 (only specified by implication $\leq$), is introduced by a definition (an equivalence axiom using $\equiv$) as the conjunction* (and) *of its known subsumers and an undefined part (expressed with an overline here):*

$$\tau(Man \leq Human) \;=\; [Man \equiv (\texttt{and}\ Human\ \overline{Man})]$$

This transformation preserves expressiveness [15].

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 redefined by simply using the transformation τ and ignoring ν. We also strengthened this definition by using a global transformation (independent from the theory to be transformed).

Definition 6 (Expressibility modulo transformation) *A language L is expressible in a language L' if and only if $\exists$ a transformation $\tau : L \rightarrow L'$, such that $\forall r \subseteq L$, $\forall m \in \mathcal{M}_L(r), \exists m' \in \mathcal{M}_{L'}(\tau(r)); \forall \delta \in L, m(\delta) = m'(\tau(\delta))$ and $\forall m' \in \mathcal{M}_{L'}(\tau(r)), \exists m \in \mathcal{M}_L(r); \forall \delta \in L, m(\delta) = m'(\tau(\delta))$*

Naturally, expressibility modulo transformation entails expressibility modulo renaming.

Definition 7 (Expressibility-based coverage)

$$L \preceq_{exprt} L' \Leftrightarrow_{def} L \textit{ is expressible (modulo transformation) in } L'$$

The following proposition is easily obtained by noting that an interpretation-preserving transformation entails expressibility modulo 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* $L \preceq_{int} L'$, *then* $L \preceq_{exprt} L'$.

The only asymmetry of these definitions is in the orientation of the transformation. Basically, the two theories are required to have a symmetric correspondence between models. There is certainly room for relaxing this constraint. Our definition of epimorphic transformations goes this way.

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 nature (e.g., when objects are taken as relations and vice versa as in dual representations of graphs).

Definition 8 (Model epimorphism) *A model epimorphism* $\pi : M \rightarrow M'$ *is a surjective map from a set of models* 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 constrain each of these models to preserve all the consequences of one of the models of the initial representation.

Definition 10 (Correspondance-based coverage)

$$L \preceq_{epi} L' \Leftrightarrow_{def} \textit{there is an epimorphic transformation } \tau : L \rightarrow L'$$

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 epimorphism from the corresponding models in the second equation of definition 6.

Proposition 3 (Expressibility-based coverage entails correspondance-based coverage) *If* $L \preceq_{exprt} L'$, *then* $L \preceq_{epi} L'$.

3.5 Consequence Preservation

Consequence preservation can be considered the ultimate goal of semantic interoperability: it denotes the fact that the consequences (i.e., the formulas satisfied by all models) of the source and the target representations are preserved (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 containment in $\mathcal{DLR}$, the authors of [16] define a mapping from the $\mathcal{DLR}$ logic (which introduces n-ary relations) to $\mathcal{CPDL}$ (Propositional Dynamic Logic with Converse)*[3]*. 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 the definition of a consequence-based coverage as usual:

Definition 12 (Consequence-based coverage)

$$L \preceq_{csq} L' \Leftrightarrow_{def} \text{there is a consequence preserving transformation } \tau : L \rightarrow L'$$

Correspondance-based coverage is stronger than consequence-based coverage because it already includes 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, thus they cannot inhibit any consequence.

Proposition 4 (Correspondance-based coverage entails consequence-based coverage) *If $L \preceq_{epi} L'$, then $L \preceq_{csq} L'$.*

It is known that expressivity modulo renaming alone does not necessarily entail consequence preservation [15].

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 inconsistent too.

Definition 13 (Consistency preservation) *A transformation τ is said to be consistency-preserving iff* $\forall r \subseteq L, \mathcal{M}_L(r) \neq \emptyset \Rightarrow \mathcal{M}_{L'}(\tau(r)) \neq \emptyset$

[3] The mapping is defined for the restriction introduced in [17] where $\mathcal{DLR}$ does not contain regular path expressions.

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 [13].*

$$\tau((\texttt{one}-\texttt{of}\ i_1\ i_2)) = (\texttt{or}\ I_1\ I_2)$$

This transformation replaces the enumeration of instances by a disjunction of concepts with the same name (here capitalized).

Consistency-based coverage is defined as usual.

Definition 14 (Consistency-based coverage)

$$L \preceq_{sat} L' \Leftrightarrow_{def} \text{there is a consistency-preserving transformation } \tau : L \rightarrow L'$$

Proposition 5 (Expressivity-based coverage entails consistency-based coverage) *If $L \preceq_{exprt} L'$, then $L \preceq_{sat} L'$.*

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 languages'. Establishing one of these structures can be more or less difficult, so it is important to be able to find the more adapted to a particular application (the p property of the beginning) and not a more powerful one. 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 strongest property satisfied by the composition of two transformations given their properties.

	$\preceq_{synt}$	$\preceq_{int}$	$\preceq_{exprt}$	$\preceq_{sat}$	$\preceq_{epi}$	$\preceq_{csq}$
$\preceq_{synt}$	$\preceq_{synt}$	$\preceq_{int}$	$\preceq_{exprt}$	$\preceq_{sat}$	$\preceq_{epi}$	$\preceq_{csq}$
$\preceq_{int}$	$\preceq_{int}$	$\preceq_{int}$	$\preceq_{exprt}$	$\preceq_{sat}$	$\preceq_{epi}$	$\preceq_{csq}$
$\preceq_{exprt}$	$\preceq_{exprt}$	$\preceq_{exprt}$	$\preceq_{exprt}$	$\preceq_{sat}$	$\preceq_{epi}$	$\preceq_{csq}$
$\preceq_{sat}$	$\preceq_{sat}$	$\preceq_{sat}$	$\preceq_{sat}$	$\preceq_{sat}$	$\emptyset$	$\emptyset$
$\preceq_{epi}$	$\preceq_{epi}$	$\preceq_{epi}$	$\preceq_{epi}$	$\emptyset$	$\preceq_{epi}$	$\preceq_{csq}$
$\preceq_{csq}$	$\preceq_{csq}$	$\preceq_{csq}$	$\preceq_{csq}$	$\emptyset$	$\preceq_{csq}$	$\preceq_{csq}$

In summary, the semantic structure of a 'family of languages' provides us with different coverage criteria 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 mechanism 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 instantiation of this approach.

4 Implementing the Approach

The 'family of languages' 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 (Section 4.2) using the DLML encoding of description logics supplied with transformations (Section 4.1).

4.1 A Concrete 'Family of Languages'

DLML [18] 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 description logic system. The FaCT system [19] 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

Recall the example of the company which needs a printer support ontology and has to merge different knowledge sources mentioned in the introduction: the technical support application ontology in DAML-ONT and the printer ontology written in the OIL language [6]. It also wants to translate the merged ontology into the $\mathcal{SHIQ}$ language in order to check the consistency of the result. The transformation must be consistency preserving. The translation methodology, from one language to another, consists of 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 languages 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 using 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 [13]. 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 (see example 5);

oneof2ornot which replaces enumerated domains (`oneof`) by disjunctive concepts whose disjuncts represent the elements of this domain: this transformation is only consistency preserving (see example 8);

cexcl2not which replaces concept exclusion (introduced by the previous transformation) by conjunction with the negated concept. This transformation is also interpretation preserving. (as (`disjoint C D`) $\Leftrightarrow C \equiv \neg D$)

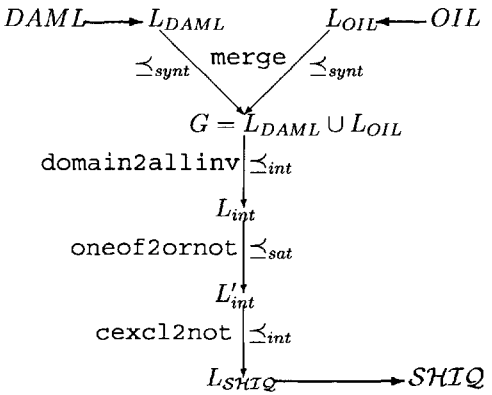


Figure 1: The transformation flow involved in importing the two ontologies to $\mathcal{SHIQ}$.

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 generalizes previous proposals for translation architectures and provides a greater flexibility in terms of languages that can be used for the integrated models. This approach is not exclusive to other approaches like direct translation or pivot approaches. But 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 properties 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.

The 'family of languages' approach is thus a tool for better 'ontology engineering'.

We have presented 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. Many work is still required for characterizing other useful properties, including properties on the syntax of the languages.

As shown, 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.

Acknowledgements

This work has benefited from the support of the European Union OntoWeb thematic network (IST-2000-29243) and the PAI Procope program of the Ministère des Affaires Étrangères (MAE) and Deutscher Akademischer Austauschdienst (DAAD).

References

[1] Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific american **279** (2001) 35–43 `http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html`.

[2] Masolo, C.: 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)

[3] Ciocoiu, M., Nau, D.: Ontology-based semantics. In: Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US). (2000) 539–546 `http://www.cs.umd.edu/~nau/papers/KR-2000.pdf`.

[4] Stuckenschmidt, H., Visser, U.: 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`.

[5] McGuinness, D., Fikes, R., Stein, L.A., James, H.: DAML-ONT: An ontology language for the semantic web. In Fensel, D., Hendler, J., Lieberman, H., Walhster, W., eds.: The semantic web: why, what, and how? The MIT press (2002) to appear.

[6] Fensel, D., Horrocks, I., Harmelen, F.V., Decker, S., Erdmann, M., Klein, M.: Oil in a nutshell. In: 12th International Conference on Knowledge Engineering and Knowledge Management EKAW 2000, Juan-les-Pins, France (2000)

[7] Enderton, H.: A mathematical introduction to logic. Academic press (1972) revised 2001.

[8] Baget, J.F., Mugnier, M.L.: The $\mathcal{SG}$ family: extensions of simple conceptual graphs. In: Proc. 17th IJCAI, Seattle (WA US). (2001) 205–210

[9] Baader, F., Lutz, C., Sturm, H., Wolter, F.: Fusions of description logics and abstract description systems. Journal of Artificial Intelligence Research **16** (2002) 1–58

[10] Levesque, H., Brachmann, R.: A Fundamental Tradeoff in Knowledge Representation and Reasoning (Revised Version). In: Readings in Knowledge Representation. Morgan Kaufmann Publishers, San Mateo (1985) 31–40

[11] Chalupsky, H.: OntoMorph: a translation system for symbolic knowledge. In: Proceedings of 7th international conference on knowledge representation and reasoning (KR), Breckenridge, (CO US). (2000) 471–482

[12] Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition **5** (1993) 199–220

[13] Horrocks, I.: A denotational semantics for Standard OIL and Instance OIL (2000) `http://www.ontoknowledge.org/oil/downl/semantics.pdf`.

[14] Euzenat, J.: Towards a principled approach to semantic interoperability. In Gomez Perez, A., Gruninger, M., Stuckenschmidt, H., Uschold, M., eds.: Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US). (2001) 19–25

[15] Baader, F.: A formal definition of the expresive power of terminological knowledge representation languages. Journal of logic and computation **6** (1996) 33–54

[16] Calvanese, D., De Giacomo, G., Lenzerini, M.: 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). (1998) 149–158

[17] Horrocks, I., Sattler, U., Tessaries, S., Tobies, S.: Query containment using a $\mathcal{DLR}$ abox - preliminary version. LCTS Report 99-15, RWTH Aachen (1999)

[18] Euzenat, J.: Preserving modularity in XML encoding of description logics. In McGuinness, D., Patel-Schneider, P., Goble, C., Möller, R., eds.: Proc. 14th workshop on description logics (DL), Stanford (CA US). (2001) 20–29

[19] Bechhofer, S., Horrocks, I., Patel-Schneider, P., Tessaris, S.: 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`.

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Tracing Data Lineage Using Schema Transformation Pathways

Hao Fan 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 chapter, we propose a new approach for tracing data lineage which is 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 common data model and a functional query language, our approach is not limited to these and would be useful in any data transformation/integration framework based on sequences of primitive schema transformations.

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 [1]. 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, 3, 4, 1, 5, 6].

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 — see `http://www.ic.ac.uk/automed`.

Common to many methods for integrating heterogeneous data sources is the requirement for logical integration [7] 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*. Using a high-level modelling language (e.g. ER, OO or relational) as the common data 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 previous work within the AutoMed project [8, 9, 10], a general framework has been developed to support schema transformation and integration. This framework provides a lower-level *hypergraph based data model (HDM)* as the common data model and a set of primitive schema transformations for schemas defined in this data model. One advantage of using a low-level data model such as the HDM is that semantic mismatches between modelling constructs are avoided. Another advantage is that it provides a unifying semantics for higher-level modelling constructs.

In particular, [10] shows how ER, relational and UML data models, and the set of primitive schema transformations on each of them, can be defined in terms of the lower-level HDM and its set of primitive schema transformations. That paper also discusses how inter-model transformations are possible within the AutoMed framework, thus allowing a schema expressed in one high-level modelling language to be incrementally transformed into a schema expressed in a different high-level modelling language. The approach was extended to also encompass XML data sources in [11], and ongoing work in AutoMed is also extending its scope to encompass formatted data files, plain text files, and RDF [12].

In the AutoMed approach, the integration of schemas is specified as a sequence of primitive schema transformation steps, which incrementally add, delete or rename schema constructs, thereby transforming each source schema into the target schema Each transformation step affects one schema construct, expressed in some modelling language. Thus, the intermediate (and indeed the target) schemas may contain constructs of more than one modelling language. We term the sequence of primitive transformations steps defined for transforming a schema S_1 into 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.

[9] discusses how AutoMed transformation pathways are *automatically reversible*, thus allowing automatic translation of data and queries between schemas. In this chapter 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. The data sources and the global schema may be expressed in the same or in different data models. AutoMed uses a functional *intermediate query language* (IQL) for defining the semantic relationships between schema constructs in each transformation step. In this chapter we show how these queries can be used to trace the lineage of data in the data warehouse.

The fundamental definitions regarding data lineage were developed in [1], including the concept of a *derivation pool* for tracing the data lineage of a tuple in a materialised view, and methods for derivation tracing with both *set* and *bag* semantics. Another fundamental concept was addressed in [13, 14], 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 [1, 3, 4, 15]. Here, we introduce the notions of *affect* and *origin* provenance in the context of AutoMed, and discuss lineage tracing algorithms for both these the two kinds of provenance. There are also other previous works related to data lineage tracing [2, 6, 5]. 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.

The remainder of this chapter is as follows. Section 2 reviews the aspects of the AutoMed framework necessary for this chapter, including the HDM data model, IQL syntax, and transformation pathways. Section 3 gives our definitions of data lineage and describes the methods of tracing data lineage that we have developed. Section 4 gives our conclusions and directions of future work.

2 The AutoMed Framework

This section gives a short review of the AutoMed framework, including the HDM data model, IQL language, and transformation pathways. More details of this material can be found in [8, 9, 10, 16].

As mentioned as above, the AutoMed framework consists of a low-level hypergraph-based data model (HDM) and a set of primitive schema transformations for schemas defined in this data model. Higher-level data models and primitive schema transformations for them are defined in terms of the lower-level HDM and its primitive schema transformations.

A *schema* in the HDM data model 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.

These definitions extend to schemas expressed in higher-level modelling languages (see for example [10] for encodings in the HDM of ER, relational and UML data models, [11] for XML and [12] for RDF). Schemas in general contain two kinds of constructs: *structural constructs* that have data extents associated with them and *constraint constructs* that map onto some set of HDM constraints. A query on a schema is defined over its structural constructs. Within AutoMed transformation pathways and queries, schema constructs are identified by a unique *scheme*, which is a tuple of labels delimited by double chevrons $\langle\langle$ and $\rangle\rangle$.

In the AutoMed integration approach, schemas are incrementally transformed by applying to them a sequence of primitive transformation steps. Each primitive transformation makes a 'delta' change to the schema, by adding, deleting or renaming just one schema construct. Each *add* or *delete* step is accompanied by a query specifying the extent of the new or deleted construct in terms of the rest of constructs in the schema. Two additional kinds of primitive transformation, *extend* and *contract*, behave in the same way as *add* and *delete* except that they indicate their accompanying query may only partially construct the extent of the schema construct being added to, or deleted from, the schema. Their accompanying query may be just the distinguished constant *void*, which indicates that there is no information about how to derive the extent of the new/deleted construct from the rest of the schema constructs, even partially. We refer the reader to [17] for an extensive discussion of the AutoMed integration approach.

To illustrate our work in this chapter, we define below a simple relational data model. However, we stress that the development in this chapter is generally applicable to all modelling languages supported by the AutoMed framework.

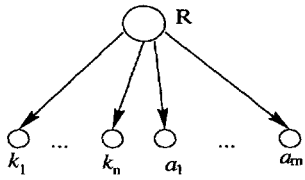


Figure 1: a simple relational data model

2.1 A Simple Relational Data Model

Schemas in our simple relational model are constructed from primary key attributes, non-primary key attributes, and the relationships between them. Figure 1 illustrates the representation of a relation R with primary key attributes $k_1, ..., k_n$ and non-primary key attributes $a_1, ..., a_m$. There is a one-one correspondence between this representation and the underlying HDM graph. In our simple relational model, there are two kinds of structural schema construct: **Rel** and **Att** (for simplicity, we ignore here the constraints present in a relational schema, e.g. key and foreign key constraints, since these are not used by our data lineage tracing methods, but see [10] for an encoding of a richer relational data model). The extent of a **Rel** construct $\langle\langle R\rangle\rangle$ is the projection of the relation R onto its primary key attributes $k_1, ..., k_n$. The extent of each **Att** construct $\langle\langle R, a\rangle\rangle$ where a is an attribute (key or non-key) is the projection of relation R onto $k_1, ..., k_n, a$.

For example, a relation $StoreSales(\underline{store\_id}, daily\_total, date)$ would be modelled by a **Rel** scheme $\langle\langle StoreSales\rangle\rangle$, and three **Att** schemes $\langle\langle StoreSales, store\_id\rangle\rangle$, $\langle\langle StoreSales, daily\_total\rangle\rangle$, $\langle\langle StoreSales, date\rangle\rangle$.

The set of primitive transformations for schemas expressed in this simple relation data model is as follows:

- **addRel**($\langle\langle R\rangle\rangle, q$) adds to the schema a new relation R.

  The query q specifies the set of primary key values in the extent of R in terms of the already existing schema constructs.

- **addAtt**($\langle\langle R, a\rangle\rangle, q$) adds to the schema an attribute a (key or non-key) for relation R.

  The query q specifies the extent of the binary relationship between the primary key attribute(s) of R and this new attribute a in terms of the already existing schema constructs.

- **delRel**($\langle\langle R\rangle\rangle, q$) deletes from the schema the relation R (provided all its attributes have first been deleted).

  The query q specifies how the set of primary key values in the extent of R can be restored from the remaining schema constructs.

- **delAtt**($\langle\langle R, a\rangle\rangle, q$) deletes from the schema attribute a of relation R.

  The query q specifies how the extent of the binary relationship between the primary key attribute(s) of R and a can be restored from the remaining schema constructs.

- **renameRel**($\langle\langle R\rangle\rangle, R'$) renames the relation R to R' in the schema.

- **renameAtt**($\langle\langle R, a\rangle\rangle, a'$) renames the attribute a of R to a'.

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, appearing in a primitive transformation is expressed in a functional *intermediate query language*, IQL [16]. For the purposes of this chapter we assume that the extents of all schema constructs are *bags*.

2.2 Simple IQL Queries

In order for our data lineage tracing methods to be unambiguous, we limit the syntax of the query q that may appear within a transformation step to the set of *simple IQL* (*SIQL*) queries — we refer the reader to [14] for a discussion of ambiguity of data lineage tracing for different classes of query language. More complex IQL queries can be encoded as a series of transformations with SIQL queries on intermediate schema constructs.

The syntax of SIQL queries, q, is listed below (lines 1 to 12). D, D_1 ..., D_n denote a bag of the appropriate type (base collections). The construct in line 1 is an enumerated bag of constants. $++$ is the bag union operator and $--$ is the bag *monus* operator [18]. *group* groups a bag of pairs on their first component. *distinct* removes duplicates from a bag. $aggFun$ is an aggregation function (*max*, *min*, *count*, *sum*, *avg*). *gc* groups a bag of pairs on their first component and applies an aggregation function to the second component.

The constructs in lines 9, 10, 11 are instances of *comprehension* syntax [19, 20]. Each $\overline{x_1}$, ..., $\overline{x_n}$ is either a single variable or a tuple of variables. $\overline{x}$ is either a single variable or value, or a tuple of variables or values, and must include all of variables appearing in $\overline{x_1}$, ..., $\overline{x_n}$. Each C_1, ..., C_k is a condition not referring to any base collection. In 9, by the constraints of comprehension syntax, each variable appearing in $\overline{x}$, C_1, ..., C_k must appear in some $\overline{x_i}$. In 10 and 11, the variables in $\overline{y}$ must appear in $\overline{x}$[1]. In line 12, *map* applies a function f to a collection D, where f is constrained not to refer to any base collections.

1. $q = [c_1, c_2, ..., c_n]$ (where $n \geq 0$ and each $c_1, ..., c_n$ is a constant)
2. $q = D_1 + +D_2 + + \ldots + +D_n$ (where $n \geq 1$)
3. $q = D_1 - -D_2$
4. $q = group\ D$
5. $q = sort\ D$
6. $q = distinct\ D$
7. $q = aggFun\ D$
8. $q = gc\ aggFun\ D$
9. $q = [\overline{x}|\overline{x_1} \leftarrow D_1; \ldots; \overline{x_n} \leftarrow D_n; C_1; ...; C_k]$ (where $n \geq 1, k \geq 0$)
10. $q = [\overline{x}|\overline{x} \leftarrow D_1;\ member\ D_2\ \overline{y}]$
11. $q = [\overline{x}|\overline{x} \leftarrow D_1;\ not(member\ D_2\ \overline{y})]$
12. $q = map\ f\ D$

IQL can represent common database queries, such as select-project-join (SPJ) queries and SPJ queries with aggregation (ASPJ). For example, to obtain the maximum daily sales total for each store in a relation $StoreSales(\underline{store\_id}, daily\_total, date)$, in SQL we would use:

[1] General comprehensions have the syntax $[e|Q_1; \ldots; Q_n]$. 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, as are $member\ D_2\ \overline{y}$ and $not(member\ D_2\ \overline{y})$. A generator has syntax $p \leftarrow c$ where p is a pattern and c is a collection-valued expression. In SIQL, the patterns p are constrained to be single variables or tuples thereof while the collection-valued expressions c are constrained to be base collections.

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

In IQL, assuming the simple relational model defined in Section 2.1, this query would be expressed by

$$gc\ max\ \langle\langle StoreSales, daily\_total\rangle\rangle$$

2.3 An Example: Transforming Relational Schemas

Consider two relational schemas S_1 and S_2. S_1 is a source schema containing two relations $mathematician(\underline{emp\_id}, salary)$ and $compScientist(\underline{emp\_id}, salary)$. S_2 is the target chema containing two relations $person(\underline{emp\_id}, salary, dept)$ and $department\ (\underline{deptName}, avgDeptSalary)$.

By our definition of the simple relational model in Section 2.1, S_1 has a set of **Rel** constructs Rel_1 and a set of **Att** constructs Att_1, while S_2 has a set of **Rel** constructs Rel_2 and a set of **Att** constructs Att_2, where:

$$\begin{array}{lcl} Rel_1 & = & \{\langle\langle mathematician\rangle\rangle, \langle\langle compScientist\rangle\rangle\} \\ Att_1 & = & \{\langle\langle mathematician, emp\_id\rangle\rangle, \langle\langle mathematician, salary\rangle\rangle \\ & & \langle\langle compScientist, emp\_id\rangle\rangle, \langle\langle compScientist, salary\rangle\rangle\} \\ Rel_2 & = & \{\langle\langle person\rangle\rangle, \langle\langle department\rangle\rangle\} \\ Att_2 & = & \{\langle\langle person, emp\_id\rangle\rangle, \langle\langle person, salary\rangle\rangle, \langle\langle person, dept\rangle\rangle \\ & & \langle\langle department, deptName\rangle\rangle, \langle\langle department, avgDeptSalary\rangle\rangle\} \end{array}$$

Schema S_1 can be transformed to S_2 by the sequence of primitive schema transformations given below. The first seven 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 extents of the existing schema constructs. The last six 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.

(1) **addRel** $(\langle\langle person\rangle\rangle, \langle\langle mathematician\rangle\rangle + + \langle\langle compScientist\rangle\rangle)$

(2) **addAtt** $(\langle\langle person, emp\_id\rangle\rangle,$
$\langle\langle mathematician, emp\_id\rangle\rangle + + \langle\langle compScientist, emp\_id\rangle\rangle)$

(3) **addAtt** $(\langle\langle person, salary\rangle\rangle,$
$\langle\langle mathematician, salary\rangle\rangle + + \langle\langle compScientist, salary\rangle\rangle)$

(4) **addAtt** $(\langle\langle person, dept\rangle\rangle, [(x, "Maths") | x \leftarrow \langle\langle mathematician\rangle\rangle] + +$
$[(x, "CompSci") | x \leftarrow \langle\langle compScientist\rangle\rangle])$

(5) **addRel** $(\langle\langle department\rangle\rangle, ["Maths", "CompSci"])$

(6) **addAtt** $(\langle\langle department, deptName\rangle\rangle, [("Maths", "Maths"), ("CompSci", "CompSci")])$

(7) **addAtt** $(\langle\langle department, avgDeptSalary\rangle\rangle,$
$gc\ avg\ [("Maths", s) | (x, s) \leftarrow \langle\langle mathematician, salary\rangle\rangle]$
$+ +\ gc\ avg\ [("Maths", s) | (x, s) \leftarrow \langle\langle mathematician, salary\rangle\rangle])$

(8) **delAtt** $(\langle\langle mathematician, salary\rangle\rangle, [(x, s) | (x, s) \leftarrow \langle\langle person, salary\rangle\rangle;$
$(x', d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "Maths"; x = x'])$

(9) **delAtt** $(\langle\langle mathematician, emp\_id\rangle\rangle, [(x, id) | (x, id) \leftarrow \langle\langle person, emp\_id\rangle\rangle;$
$(x', d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "Maths"; x = x'])$

(10) **delRel** $(\langle\langle mathematician\rangle\rangle, [x | (x, d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "Maths"])$

(11) **delAtt** $(\langle\langle compScientist, salary\rangle\rangle, [(x, s) | (x, s) \leftarrow \langle\langle person, salary\rangle\rangle;$
$(x', d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "CompSci"; x = x'])$

(12) **delAtt** $(\langle\langle compScientist, emp\_id\rangle\rangle, [(x, id)|(x, id) \leftarrow \langle\langle person, emp\_id\rangle\rangle; (x', d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "CompSci"; x = x'])$

(13) **delRel** $(\langle\langle compScientist\rangle\rangle, [x|(x, d) \leftarrow \langle\langle person, dept\rangle\rangle; d = "CompSci"])$

Note that we actually permit the specification of compositions of SIQL queries within primitive transformations e.g. the queries in steps (4) and (7) above are not SIQL queries but are composed of nested SIQL sub-queries. Such queries are automatically broken down by our software into a sequence of *add* or *delete* transformation steps with SIQL queries within them. The decomposition procedure undertakes a depth-first search of the query tree and generates the sequence of transformations from the bottom up. For example, the following decompositions would be equivalent to steps (4) and (7) above, with (4.1) $\sim$ (4.5) replacing step (4) and (7.1) $\sim$ (7.9) replacing step (7)[2]:

(4.1) **addAtt** $(\langle\langle person, mathsDept\rangle\rangle, [(x, "Maths")|x \leftarrow \langle\langle mathematician\rangle\rangle])$

(4.2) **addAtt** $(\langle\langle person, CSDept\rangle\rangle, [(x, "CompSci")|x \leftarrow \langle\langle compScientist\rangle\rangle])$

(4.3) **addAtt** $(\langle\langle person, dept\rangle\rangle, \langle\langle person, mathsDept\rangle\rangle + + \langle\langle person, CSDept\rangle\rangle)$

(4.4) **delAtt** $(\langle\langle person, CSDept\rangle\rangle, [(x, "CompSci")|x \leftarrow \langle\langle compScientist\rangle\rangle])$

(4.5) **delAtt** $(\langle\langle person, mathsDept\rangle\rangle, [(x, "Maths")|x \leftarrow \langle\langle mathematician\rangle\rangle])$

(7.1) **addRel** $(\langle\langle mathsSalary\rangle\rangle, map\ (\lambda(x, s).("Maths", s))\ \langle\langle mathematician, salary\rangle\rangle)$

(7.2) **addRel** $(\langle\langle avgMathsSalary\rangle\rangle, gc\ avg\ \langle\langle mathsSalary\rangle\rangle)$

(7.3) **addRel** $(\langle\langle compSciSalary\rangle\rangle, map\ (\lambda(x, s).("CompSci", s))\ \langle\langle compScientist, salary\rangle\rangle)$

(7.4) **addRel** $(\langle\langle avgCompSciSalary\rangle\rangle, gc\ avg\ \langle\langle compSciSalary\rangle\rangle)$

(7.5) **addAtt** $(\langle\langle department, avgDeptSalary\rangle\rangle, \langle\langle avgMathsSalary\rangle\rangle + + \langle\langle avgCompSciSalary\rangle\rangle)$

(7.6) **delRel** $(\langle\langle avgCompSciSalary\rangle\rangle, gc\ avg\ \langle\langle compSciSalary\rangle\rangle)$

(7.7) **delRel** $(\langle\langle compSciSalary\rangle\rangle, map\ (\lambda(x, s).("CompSci", s))\ \langle\langle compScientist, salary\rangle\rangle)$

(7.8) **delRel** $(\langle\langle avgMathsSalary\rangle\rangle, gc\ avg\ \langle\langle mathsSalary\rangle\rangle)$

(7.9) **delRel** $(\langle\langle mathsSalary\rangle\rangle, map\ (\lambda(x, s).("Maths", s))\ \langle\langle mathematician, salary\rangle\rangle)$

3 Tracing Data Lineage in AutoMed

3.1 Data Lineage with Bag Semantics in SIQL

The fundamental definitions regarding data lineage were developed in [1], as were methods for derivation tracing with both *set* and *bag* semantics. However, these definitions and methods are limited to *why-provenance*[14], and also to the 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 based on *bag* semantics, allowing duplicate elements within a source construct or integrated construct, 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. In particular, we use the notions of *maximal witness* and *minimal witness* from [14] in order to define the

[2]We have left the IQL queries in steps (8) $\sim$ (13) as is, since all *delete* transformations are ignored in our data lineage tracing procedures (see Section 3.2 below).

notions of *affect-pool* and *origin-pool* in Definitions 1 and 2 below.

In both these definitions, condition (a) states that the result of applying query q to the lineage must be the bag of all tuples t in V.

Condition (b) is used to enforce the maximizing and minimizing properties, respectively. Thus, the affect-pool includes all elements in data sources which could generate t by applying q to them, while if any element and all of its copies in the origin-pool was deleted, then t or all of t's copies in V could not be generated by applying the query q to the lineage.

Condition (c) in both definitions removes the redundant elements in the computed derivation of tuple t (see [1]). Condition (d) in Definition 2 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^*.

Definition 1 (Affect-pool for a SIQL query) Let q be any SIQL 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) $q(T_1^*, \ldots, T_m^*) = \{x | x \leftarrow V; x = t\}$

(b) $\forall T_i$': $q(T_1^*, \ldots, T_i', \ldots, T_m^*) = \{x | x \leftarrow V; x = t\} \Rightarrow T_i' \subseteq T_i^*$

(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 2 (Origin-pool for a SIQL query) 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) $q(T_1^*, \ldots, T_m^*) = \{x | x \leftarrow V; x = t\}$

(b) $\forall T_i^*$: $\forall t^* \in T_i^*$: $q(T_1^*, \ldots, \{x | x \leftarrow T_i^*; x \neq t^*\}, \ldots, T_m^*) \neq \{\, x | x \leftarrow V; x = t\}$

(c) $\forall T_i^*$: $\forall t^* \in T_i^*$: $q(T_1^*, \ldots, \{t^*\}, \ldots, T_m^*) \neq \emptyset$

(d) $\forall T_i^*$: $\neg\exists t^*$: $t^* \in T_i^*, t^* \in (T_i - T_i^*)$

Proposition 1. The origin-pool of a tuple t is a sub-bag of the affect-pool of t.

Following on from the above definitions and the definition of SIQL queries in Section 2.2, we now specify the affect-pool and origin-pool for SIQL queries. As in [1], 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 SIQL queries). Let $V = q(D)$ be the bag that results from applying a SIQL 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:

1. $q = [c_1, ..., c_n]$
   $TQ_D^{AP}(t) = TQ_D^{OP}(t) = [x|x \leftarrow [c_1, ..., c_n]; x = t]$
   /*An enumerated bag can be regarded as a system-defined base collection*/
   /*for the purposes of data lineage tracing. */
2. $q = D_1 + + \ldots + + D_n \quad (D = \langle D_1, \ldots, D_n \rangle)$
   $TQ_D^{AP}(t) = TQ_D^{OP}(t) = \langle [x|x \leftarrow D_1; x = t], \ldots, [x|x \leftarrow D_n; x = t] \rangle$
3. $q = D_1 - - D_2 \quad (D = \langle D_1, D_2 \rangle)$
   $TQ_D^{AP}(t) = \langle [x|x \leftarrow D_1; x = t], D_2 \rangle$
   $TQ_D^{OP}(t) = \langle [x|x \leftarrow D_1; x = t], [x|x \leftarrow D_2; x = t] \rangle$
4. $q = group\ D$
   $TQ_D^{AP}(t) = TQ_D^{OP}(t) = [x|x \leftarrow D; first\ x = first\ t]$
5. $q = sort\ D / distinct\ D$
   $TQ_D^{AP}(t) = TQ_D^{OP}(t) = [x|x \leftarrow D; x = t]$
6. $q = max\ D\ /\ min\ D$
   $TQ_D^{AP}(t) = D$
   $TQ_D^{OP}(t) = [x|x \leftarrow D; x = t]$
7. $q = sum\ D$
   $TQ_D^{AP}(t) = D$
   $TQ_D^{OP}(t) = [x|x \leftarrow D; x \neq 0]$
8. $q = count\ D\ /\ avg\ D$
   $TQ_D^{AP}(t) = TQ_D^{OP}(t) = D$
9. $q = gc\ max\ D\ /\ gc\ min\ D$
   $TQ_D^{AP}(t) = [x|x \leftarrow D; first\ x = first\ t]$
   $TQ_D^{OP}(t) = [x|x \leftarrow D; x = t]$
10. $q = gc\ sum\ D$
    $TQ_D^{AP}(t) = [x|x \leftarrow D; first\ x = first\ t]$
    $TQ_D^{OP}(t) = [x|x \leftarrow D; first\ x = first\ t; second\ x \neq 0]$
11. $q = gc\ count\ D\ /\ gc\ avg\ D$
    $TQ_D^{AP}(t) = TQ_D^{PP}(t) = [x|x \leftarrow D; first\ x = first\ t]$
12. $q = [\overline{x}|\overline{x_1} \leftarrow D_1; \ldots; \overline{x_n} \leftarrow D_n; C_1; ...; C_k] \qquad (D = \langle D_1, \ldots, D_n \rangle)$
    $TQ_D^{AP}(t) = TQ_D^{OP}(t) =$
    $\langle [x_1|x_1 \leftarrow D_1; x_1 = (\lambda\overline{x}.\overline{x_1})\ t], \ldots, [x_n|x_n \leftarrow D_n; x_n = (\lambda\overline{x}.\overline{x_n})\ t] \rangle$
13. $q = [\overline{x}|\overline{x} \leftarrow D_1; member\ D_2\ \overline{y}] \qquad (D = \langle D_1, D_2 \rangle)$
    $TQ_D^{AP}(t) = TQ_D^{OP}(t) = \langle [\overline{x}|\overline{x} \leftarrow D_1; \overline{x} = t], [y|y \leftarrow D_2; y = (\lambda\overline{x}.\overline{y})\ t] \rangle$
14. $q = [\overline{x}|\overline{x} \leftarrow D_1; not(member\ D_2\ \overline{y})] \qquad (D = \langle D_1, D_2 \rangle)$
    $TQ_D^{AP}(t) = \langle [\overline{x}|\overline{x} \leftarrow D_1; \overline{x} = t], [y|y \leftarrow D_2; y = (\lambda\overline{x}.\overline{y})\ t] \rangle$
    $TQ_D^{OP}(t) = \langle [\overline{x}|\overline{x} \leftarrow D_1; \overline{x} = t], \emptyset \rangle$
15. $q = map\ f\ D$
    $TQ_D^{AP}(t) = TQ_D^{OP}(t) = [\overline{x}|\overline{x} \leftarrow D; (f\ \overline{x}) = t]$
    If f is invertible, we can obtain t's data lineage directly using f's inverse function f^{-1} i.e. $TQ_D^{AP}(t) = TQ_D^{OP}(t) = f^{-1}(t)$.

It is straightforward to show that the results of queries $TQ_D^{AP}(t)$ and $TQ_D^{OP}(t)$ satisfy Definition 1 and 2 respectively.

3.2 Tracing Data Lineage Through Transformation Pathways

For simplicity of exposition, 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 3 (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 $D_i^* = q_i^{AP}(D_{i+1}^*)$ $(1 \leq i \leq r)$, $D_{i+1}^* = \{t\}$ and $D^* = D_1^*$.

Definition 4 (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 $D_i^* = q_i^{OP}(D_{i+1}^*)$ $(1 \leq i \leq r)$, $D_{i+1}^* = \{t\}$ and $D^* = D_1^*$.

Definitions 3 and 4 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 consists of a sequence of primitive transformations which generate the integrated schema from the given source schemas. The schema constructs are generally different for different modelling languages. For example, HDM schemas have **Node**, **Edge**, and **Constraint** constructs, ER schemas have **Entity**, **Attribute**, **Relationship** and **Generalisation** constructs [10], while the simple relational schemas of Section 2.1 have **Rel** and **Att** constructs.

Thus, each modelling language also generally has a different set of primitive transformations. For example, for the HDM these are **addNode**, **delNode**, **addEdge**, **delEdge** etc. while for our simple relational data model of Section 2.1 they are **addRel**, **delRel**, **addAtt**, **delAtt** etc.

When considering data lineage tracing, we are only concerned with structural constructs associated with a data extent e.g. **Node** and **Edge** constructs in the HDM, and **Rel** and **Att** constructs in the simple relational data model. Thus, for data lineage tracing, we ignore primitive schema transformation steps which are adding, deleting or renaming only constraints. Moreover, we treat any primitive transformation which is adding a construct to a schema as a generic *addConstruct* transformation, any primitive transformation which is deleting a construct from a schema as a generic *delConstruct* transformation, and any primitive transformation which is renaming a schema construct as a generic *renameConstruct* transformation. We can summarize the problem of data lineage for each of these transformations 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 the query 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.

3.3 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 $\emptyset$.

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. a 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 schema 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') | O' ← O.relateTP.source]
          D* = TQ_D^AP(t);
          DL = [(B, B.construct) | B ← 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. The procedure *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 an 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 sequenceDL^{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 $tp_1, \ldots, tp_r$ 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, ..., tp_r
output : B's affect-pool, DL
begin
         DL = ⟨(B, O)⟩;
         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 = distinct D_i;
               DL = merge(DL, affectPoolOfSet(D_i, O_i)); }
           }
         return(DL);
end
```

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

We illustrate the use of the *traceAffectPool* algorithm above by means of a simple example. Referring back to the example schema transformation in Section 2.3, suppose we have a tracing tuple $t = (\,"Maths", 2500)$ in the extent of $\langle\langle department, avgDeptSalary\rangle\rangle$ in S_2. The affect-pool, DL, of this tuple is traced as follows.

Initially, $DL = \langle\langle(\,"Maths", 2500), \langle\langle department, avgDeptSalary\rangle\rangle\rangle\rangle$. *traceAffectPool* ignores all the *del* steps, and finds the *add* transformation step whose *result* is $"\langle\langle department, avgDeptSalary\rangle\rangle"$. This is step (7.5), $tp_{(7.5)}$, and:

$tp_{(7.5)}.query = \langle\langle avgMathsSalary\rangle\rangle + + \langle\langle avgCompSciSalary\rangle\rangle$ and

$tp_{(7.5)}.source = [\langle\langle avgMathsSalary\rangle\rangle,\ \langle\langle avgCompSciSalary\rangle\rangle]$

Using algorithm *affectPoolOfSet*, t's lineage at $tp_{(7.5)}$ is as follows:

$$\begin{aligned} DL_{(7.5)} &= \langle([x|x \leftarrow \langle\langle avgMathsSalary\rangle\rangle; x = ("Maths", 2500)], \langle\langle avgMathsSalary\rangle\rangle), \\ &\quad ([x|x \leftarrow \langle\langle avgCompSciSalary\rangle\rangle; x = ("Maths", 2500)], \langle\langle avgCompSciSalary\rangle\rangle)\rangle \\ &= \langle(("Maths", 2500), \langle\langle avgMathsSalary\rangle\rangle), (\emptyset, \langle\langle avgCompSciSalary\rangle\rangle)\rangle \\ &= \langle(("Maths", 2500), \langle\langle avgMathsSalary\rangle\rangle)\rangle \end{aligned}$$

After removing the original tuple, $(\,"Maths", 2500), \langle\langle department, avgDeptSalary\rangle\rangle)$, and merging its lineage $DL_{(7.5)}$, the updated DL is $\langle(("Maths", 2500), \langle\langle avgMathsSalary\rangle\rangle)\rangle$.

Similarly, we obtain the data lineage relating to above DL. Thus, $DL_{(7.2)}$ is all of the tuples in $\langle\langle mathsSalary\rangle\rangle$ and $DL_{(7.1)}$ is all of the tuples in $\langle\langle mathematician, salary\rangle\rangle$, where $\langle\langle mathematician, salary\rangle\rangle$ is a base collection in S_1.

We conclude that the affect-pool of tuple ($"Maths", 2500$) in the extent of $\langle\langle department,\ avgDeptSalary\rangle\rangle$ in S_2 consists of all of the tuples in the extent of $\langle\langle mathematician, salary\rangle\rangle$ in S_1.

4 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, OO or relational model, the AutoMed integration approach is based on a lower-level common data model – the HDM data model. High-level modelling languages, and the set of primitive schema transformations on each of them, are defined in terms of the HDM and its set of primitive schema transformations. The integration of schemas is specified as a sequence of primitive schema transformation steps, which incrementally add, delete or rename schema constructs, thereby transforming each source schema into the target schema. Each transformation step affects one schema construct, expressed in some modelling language, and the intermediate and target schemas may contain constructs of more than one modelling language.

The contribution of the work described in this chapter 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* schema transformations carry a query which defines the new schema construct in terms of the other schema constructs. Although developed for a graph-based common data model and a functional query language, our lineage tracing approach is not limited to these and could be applied in any data transformation/integration framework which based on sequences of primitive schema transformations.

We are currently implementing the algorithms presented here, and also algorithms for incremental view maintenance. The data lineage problem and the solutions presented in this chapter have led to a number of areas of further work:

- Making use of the information imparted by queries in *delete* transformation steps, and also the partial information imparted by queries in *extend* and *contract* transformation steps.
- Combining data lineage tracing 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.
- Extending the lineage tracing and view maintenance algorithms to a more expressive transformation language: [21] 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] Cui, Y., Widom, J., Wiener, J.: Tracing the lineage of view data in a warehousing environment. ACM Transactions on Database Systems (TODS) **25** (2000) 179–227

[2] Bernstein, P.A., Bergstraesser, T.: Meta-data support for data transformations using microsoft repository. IEEE Data Engineering Bulletin **22** (1999) 9–14

[3] Woodruff, A., Stonebraker, M.: Supporting fine-grained data lineage in a database visualization environment. In: Proceedings of the Thirteenth International Conference on Data Engineering (ACDE'97), Birmingham, U.K, IEEE Computer Society (1997) 91–102

[4] Cui, Y.: Lineage tracing in data warehouses. PhD thesis, Computer Science Department, Stanford University (2001)

[5] Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.: Improving data cleaning quality using a data lineage facility. In: Proc. Design and Management of Data Warehouses (DMDW), Interlaken, Switzerland. (2001) 3

[6] Faloutsos, C., Jagadish, H., Sidiropoulos, N.: Recovering information from summary data. In: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, Morgan Kaufmann (1997) 36–45

[7] Hull, R.: Managing semantic heterogeneity in databases: A theoretical perspective. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona, ACM Press (1997) 51–61

[8] Poulovassilis, A., McBrien, P.: A general formal framework for schema transformation. Data and Knowledge Engineering **28** (1998) 47–71

[9] McBrien, P., Poulovassilis, A.: Automatic migration and wrapping of database applications - a schema transformation approach. In: Conceptual Modeling - ER '99, 18th International Conference on Conceptual Modeling, Paris, France. Volume 1728 of Lecture Notes in Computer Science., Springer (1999) 96–113

[10] McBrien, P., Poulovassilis, A.: A uniform approach to inter-model transformations. In: Advanced Information Systems Engineering, 11th International Conference CAiSE'99, Heidelberg, Germany. Volume 1626 of Lecture Notes in Computer Science., Springer (1999) 333–348

[11] McBrien, P., Poulovassilis, A.: A semantic approch to integrating XML and structured data sources. In: Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland. Volume 2068 of Lecture Notes in Computer Science., Springer (2001) 330–345

[12] Williams, D.: Representing RDF and RDF Schema in the HDM. Technical report, Automed Project (2002) BBKCS-02-11.

[13] Buneman, P., Khanna, S., Tan, W.: Data provenance: some basic issues. In: Foundations of Software Technology and Theoretical Computer Science, 20th Conference, (FST TCS) New Delhi, India. Volume 1974 of Lecture Notes in Computer Science., Springer (2000) 87–93

[14] Buneman, P., Khanna, S., Tan, W.: Why and where: A characterization of data provenance. In: Database Theory - ICDT 2001, 8th International Conference, London, UK. Volume 1973 of Lecture Notes in Computer Science., Springer (2001) 316–330

[15] Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy, Morgan Kaufmann (2001) 471–480

[16] Poulovassilis, A.: The Automed Intermediate Query Language. Technical report, Automed Project (2001)

[17] McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: 19th International Conference on Data Engineering, ICDE'03(to appear), March 5 - March 8, 2003 - Bangalore, India. (2003)

[18] Albert, J.: Algebraic properties of bag data types. In: 17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings, Morgan Kaufmann (1991) 211–219

[19] Trinder, P.W.: Comprehensions, a query notation for dbpls. In: Database Programming Languages: Bulk Types and Persistent Data. 3rd International Workshop, Nafplion, Greece, Proceedings, Morgan Kaufmann (1991) 55–68

[20] Buneman, P., Libkin, L., Suciu, D., Tanen, V., Wong, L.: Comprehension syntax. SIGMOD Record **23** (1994) 87–96

[21] Poulovassilis, A.: An enhanced transformation language for the HDM. Technical report, Automed Project (2001)

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Ontology Extraction for Distributed Environments

Derek Sleeman[1] Stephen Potter[2] Dave Robertson[2] Marco Schorlemmer[2]

[1] *Department of Computing Science,*
University of Aberdeen, United Kingdom
sleeman@csd.abdn.ac.uk

[2] *Division of Informatics,*
University of Edinburgh, United Kingdom
stephenp@aiai.ed.ac.uk, {dr,marco}@inf.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 characterisation that would enable them to be exploited fully. In this chapter we discuss a technique which can be applied to identify ontological knowledge implicit in a knowledge base. Based on this technique, a tool has been implemented which allows this knowledge to be extracted, thereby promoting the re-use of the resource. A discussion of some complementary research into brokering services within distributed knowledge architectures serves to illustrate the sort of environment in which such re-use might be enacted.

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]. In this chapter, we describe one outcome of our research into techniques to enable brokering systems to become more effective and more intelligent: a technique for the semi-automatic 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*[1] 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.

The rest of the chapter 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 gives a brief overview of our conception of a brokering system, in order to illustrate the role that this technique can play in facilitating knowledge services within distributed environments. Section 4 discusses related work, and, to conclude, Section 5 summarises the chapter.

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:

$$\begin{array}{l} member(X, [X|T]) \\ member(X, [H|T]) \quad \leftarrow \quad member(X, T) \end{array}$$

The Clark completion of this predicate is:

$$member(X, L) \quad \leftrightarrow \quad L = [X|T] \ \vee \ (L = [H|T] \wedge member(X, T)) \tag{1}$$

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.

$$member(c, [a, b]) \quad \rightarrow \quad [a, b] = [c|T] \ \vee \ ([a, b] = [H|T] \wedge member(c, T)) \tag{2}$$

[1] Throughout this chapter, 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.

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 $animal(X)$:

$$\begin{aligned} animal(X) &\leftarrow mammal(X) \\ animal(X) &\leftarrow fish(X) \end{aligned}$$

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

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:

$$\begin{aligned} animal(X) &\leftarrow mammal(X) \\ animal(X) &\leftarrow fish(X) \\ mammal(X) &\leftarrow vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X) \\ fish(X) &\leftarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X) \end{aligned} \quad (3)$$

The constraints extracted for this program (seen in the lower window of Figure 1) are:

$$\begin{aligned} animal(X) &\rightarrow mammal(X) \vee fish(X) \\ fish(X) &\rightarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X) \\ mammal(X) &\rightarrow vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X) \end{aligned} \quad (4)$$

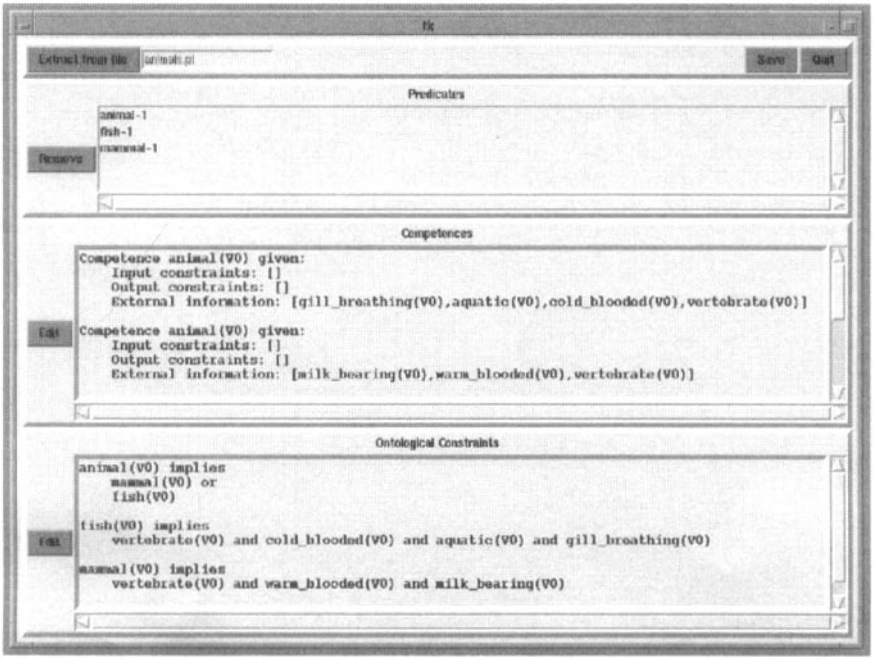


Figure 1: Ontology extraction tool

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 in expression (5) provide a basic example (extended versions of this appear in [5]).

$$
\begin{aligned}
&solve(true, \{\}) \\
&solve((A \wedge B), E_a \cup E_b) \leftarrow solve(A, E_a) \wedge solve(B, E_b) \\
&solve((A \vee B), E) \leftarrow solve(A, E) \vee solve(B, E) \\
&solve(X, E \cup \{C | (X \rightarrow C \wedge not(C))\}) \leftarrow clause(X, B) \wedge solve(B, E)
\end{aligned}
\tag{5}
$$

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 by, for example, giving the standard interpreter 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 must now be asserted in the form (for example) `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` $\quad$ `C` $= \{\}$

`X = a2` $\quad$ `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 extraction 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:

$$compatible(Person1, Person2) \leftarrow object(person, O1) \wedge name(O1, Person1) \wedge pet(O1, Pet1) \wedge object(person, O2) \wedge name(O2, Person2) \wedge pet(O2, Pet2) \wedge object(pet, O3) \wedge name(O3, Pet1) \wedge likes(O3, Pet2)$$

$object(person, p1)$	$name(p1, fred)$	$gender(p1, male)$	$pet(p1, tiddles)$
$object(person, p2)$	$name(p2, sue)$	$gender(p2, female)$	$pet(p2, claud)$
$object(person, p3)$	$name(p3, tom)$	$gender(p3, male)$	$pet(p3, rover)$
$object(person, p4)$	$name(p4, jane)$	$gender(p4, female)$	$pet(p4, squeak)$
$object(pet, x1)$	$name(x1, tiddles)$	$likes(x1, claud)$	
$object(pet, x2)$	$name(x2, claud)$	$likes(x2, tiddles)$	
$object(pet, x3)$	$name(x3, rover)$	$likes(x3, rover)$	
$object(pet, x4)$	$name(x4, squeak)$	$likes(x4, claud)$	

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 the extraction tool. 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 Knowledge Services and Brokering

Alongside research into knowledge services such as ontology extraction, we have been pursuing parallel research into brokering mechanisms for knowledge resources (for further details see [6]). The purpose of this section is to give a brief overview of this work and to indicate how it relates to the ontology extraction tool described above (which is the principal focus of this chapter).

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 [7, 8, 9]), 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,[2] 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.[3] 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.

3.1 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, the extraction tool described above offers a means by which resources lacking ontological definitions can be made accessible to brokers.

This tool might be applied locally by the 'owner' of the resource in order to augment it with its ontology before introducing it into the brokering environment. Alternatively (and more interestingly), the extraction tool could itself be an agent in this environment, offering its services via the broker.

4 Related Work

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 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 [11, 12, 13]). 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 [14, 15]. Thus far, the successes of these text-centred approaches have been limited, with unresolved questions surrounding the extent of

[2]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.

[3]The use of bridges here is analogous to the use of bridges in UPML[10].

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* [16], 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 [17, 18].

However, while our extraction tool undoubtedly has similar intentions and shares certain concerns with the work outlined above, it is distinguished from it 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 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 chapter has described a technique by which latent information — namely implicit ontological constraints — in knowledge bases can be extracted, and done so in a relatively simple, low-cost manner. 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] Crubezy, M., Lu, W., Motta, E., Musen, M.: The internet reasoning service: delivering configurable problem-solving components to web users. In: Proceedings of the Workshop on Interactive Tools for Knowledge Capture at the First International Conference on Knowledge Capture (K-CAP 2001), Victoria, Canada. (2001) 15–22

[2] Lopez, M., Gomez-Perez, A., Rojas-Amaya, M.: Ontology's crossed life cycle. In: Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000), Springer (2000) 65–79

[3] Lei, G., Sleeman, D., Preece, A.: 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] Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Databases. Plenum Press (1978) 293–322

[5] Kalfoglou, Y., Robertson, D.: Use of formal ontologies to support error checking in specifications. In: Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW-99), Springer (1999) 207–221

[6] Schorlemmer, M., Potter, S., Robertson, D., Sleeman, D.: Knowledge Life-Cycle Management over a Distributed Architecture. Expert Update **5** (2002)

[7] Arisha, K., Eiter, T., Kraus, S., Ozcan, F., R., R., Subrahmanian, V.: IMPACT: interactive Maryland platform for agents collaborating together. IEEE Intelligent Systems Magazine **14** (2000) 64–72

[8] Nodine, M., Unruh, A.: Facilitating open communication in agent systems. In Singh, M., Rao, A., Wooldridge, M., eds.: Intelligent Agents IV: Agent Theories, Architectures, and Languages. Springer (1998) 281–296

[9] Sycara, K., Klusch, M., Widoff, S., Lu, J.: Dynamic service matchmaking among agents in open information environments. ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems) **28** (1999) 47–53

[10] Fensel, D., Benjamins, V., Motta, E., Wielinga, B.: UPML: a framework for knowledge system reuse. In: Proceedings of the International Joint Conference on AI (IJCAI-99), Stockholm, Sweden, July 31–August 5, 1999, Morgan Kaufmann (1999) 16–23

[11] Bowden, P., Halstead, P., Rose, T.: Extracting conceptual knowledge from text using explicit relation markers. In: Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996, Springer (1996) 147–162

[12] Faure, D., Nédellec, C.: Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system ASIUM. In: Proceedings of the 11th European Workshop on Knowledge Acquisition Modeling and Management (EKAW '99), Springer (1999) 329–334

[13] Hahn, U., Klenner, M., Schnattinger, K.: Automated knowledge acquisition meets metareasoning: incremental quality assessment of concept hypotheses during text understanding. In: Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996, Springer (1996) 131–146

[14] Maedche, A., Staab, S.: Discovering conceptual relations from text. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000), August 20-25 2000, Berlin, Germany, Amsterdam, IOS Press (2000) 321–325

[15] Reimer, U.: Automatic acquisition of terminological knowledge from texts. In: Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90), Stockholm, August 6-10, 1990, London, Pitman (1990) 547–549

[16] Biggerstaff, T., Mitbander, B., Webster, D.: Program understanding and the concept assignment problem. Communications of the ACM **37** (1994) 72–83

[17] Li, Y., Yang, H., Chu, W.: Clarity guided belief revision for domain knowledge recovery in legacy systems. In: Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, USA, Springer (2000)

[18] Yang, H., Cui, Z., O'Brien, P.: Extracting ontologies from legacy systems for understanding and re-engineering. In: Proceedings of the 23rd IEEE International Conference on Computer Software and Applications (COMPSAC '99), IEEE Press (1999)

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

UML for the Semantic Web: Transformation-Based Approaches

Kateryna Falkovych[1] Marta Sabou[2] Heiner Stuckenschmidt[2]

[1] *Department of Multimedia and Human Computer Interaction*
CWI, Kruislaan 413 P.O. Box 94079 1090 GB
Amsterdam, The Netherlands
Kateryna.Falkovych@cwi.nl

[2] *Artificial Intelligence Department*
Vrije Universiteit, De Boelelaan 1081, 1081HV,
Amsterdam, The Netherlands
{marta,heiner}@cs.vu.nl

Abstract. The perspective role of UML as a conceptual modelling language for the Semantic Web has become an important research topic. We argue that UML could be a key technology for overcoming the ontology development bottleneck thanks to its wide acceptance and sophisticated tool support. Transformational approaches are a promising way of establishing a connection between UML and web-based ontology languages. We compare some proposals for defining transformations between UML and web ontology languages and discuss the different ways they handle the conceptual differences between these languages. We identify commonalities and differences of the approaches and point out open questions that have not or not satisfyingly been addressed by existing approaches.

1 Introduction

The so-called Semantic Web [1] aims at enriching the World Wide Web with semantic information to enable systems to access and use information more efficiently. For this purpose a number of annotation languages have been developed in order to annotate information resources with content-related meta-data. In order to create, interpret and compare meta-data annotations, ontologies, explicit definitions of the vocabulary used in an information source, are needed. While meta-data can often be generated from the content of an information source, the development of ontologies is likely to become the major bottleneck in scaling up semantic annotations. Overcoming the modelling bottleneck requires a large number of professional ontology builders equipped with powerful modelling tools. The current situation is far from the required one. Today, ontologies are built by a small number of people, in most cases researchers, with prototype tools that provide some basic functionality for editing and storing ontological models. These tools have little in common with professional development environments we know from the area of software engineering. Admittedly, the tool support for ontological modelling has improved over the last years and editors are available now that

support consistency checking, import of existing ontologies and visualization of ontologies, but at the moment it is unrealistic to claim that we can give these tools to non experts in ontological modelling and expect good modelling results.

Recently, the Unified Modelling Languages (UML) has been identified as a way of providing a partial solution to the modelling bottleneck. Being the standard modelling language in software engineering, UML has received wide attention not only in academia, but also in professional software development. As a consequence, UML is much better supported in terms of tools and available expertise than the emerging semantic web languages. The wide acceptance of UML makes it an ideal language to be used by a critical mass of people to build high quality models of information semantics for the semantic web. A number of researchers have recognized the importance of UML with ontological modelling, especially for the semantic web (e.g. [2]). In this paper, we discuss approaches, that are based on the idea of transforming domain models between UML and semantic web languages, focusing on the actual transformation between the two languages.

In the following, we contrast the different ideas behind UML and the semantic web languages concluding the resulting differences between these languages. Based on this high level discussion, we turn our interest to transformations between these languages. We first identify two slightly different use cases of employing transformations between UML and semantic web languages. Afterwards, we outline some existing approaches and their specific solutions to the transformation problem. After a discussion of the different choices made by different approaches we identify a number of open issues with respect to transformation that are essential for using UML on the semantic web. We conclude with an outline of a research agenda for an effective use of UML on the semantic web.

2 UML and Semantic Web Languages

The common feature of all transformation-based approaches to using UML on the semantic web is the fact that they focus on the languages as such. Other important aspects such as methodologies, modelling tools or reasoning support are assumed to be addressed by the corresponding languages communities. In fact, the standardization of UML on the one and OWL on the other hand has stimulated many developments around these languages that we can take advantage of once we have transformed a model into the respective language. Consequently, the central question in transformation-based approaches is concerned with the relation between UML and the semantic web languages. In order to understand the technical differences, it is important to recall a little bit of the history and the motivation that underlies the two languages.

2.1 Unified Modelling Language UML

Aim: The UML language is designed in order to integrate competing proposals for modelling languages in the area of software engineering. This integration effort was undertaken in order to push object-oriented design methods into industrial practice by providing a well-supported standard method that makes development processes more transparent.

Principles: UML is primarily designed as a language to be used by humans to document and communicate software designs. As a consequence, the main notation of UML is de-

fined in terms of a graphical model rather than a formal language. In order to capture formal constraints, an additional constraint language (OCL) has to be used. Being an integration of different pre-existing proposals, UML is rather the union of these approaches than an intersection and aims at maximal expressivity. In order to cover all aspects of a system, the language uses different interacting diagram types each covering a specific aspect of the overall system.

Semantics: The fact that UML consists of different diagram types that interact with each other makes it difficult to define a uniform semantics. The only approach to define the semantics of UML that covers all diagrams is the meta-model approach. In this approach the modelling elements of the language are defined in terms of UML class diagrams and OCL constraints. Further, there are attempts to define formal semantics of parts of the language, such as class diagrams or state charts. Validation and transformation methods for UML models have been defined on the basis of these partial semantics.

In order to capture the various aspects of complex software systems, UML consists of not less than twelve different diagram types (according to OMG) each being designed to describe the system from a specific point of view. Four diagrams describe the static application structure, five different aspects of the dynamic behavior, and three represent ways to organize and manage the application modules. The class diagrams belong to the first category and they have been in the center of attention because there exists a direct relation between their elements and the parts of an ontology (classes, hierarchies, class attributes and axioms). Some language features, especially from the system structure part will be mentioned in the description of the transformation approaches in the following section. For a complete description of UML we refer to the official specifications [3].

2.2 Web Ontology Language OWL

Aim: The goal of the Web Ontology Language OWL is to provide a standard language for the representation of ontologies on the World Wide Web. Such a standardized language will support the development of a web-based ontology infrastructure by providing editors, storages, inference engines and meta-data annotation tools.

Principles: In order to fit into the general web architecture, OWL adopts a number of principles, including a XML-based encoding and backward compatibility with RDF Schema, the current W3C standard for conceptual modelling. As such, the language is designed to be handled by systems rather than by human users. The provision of a well founded logical semantics is a design principle of OWL. Beyond this, the developers of OWL follow the principle of minimality: rather than including as many modelling features as possible, the OWL language is restricted to a set of features that make logical reasoning feasible.

Semantics: OWL has a well founded semantics in the spirit of description logics [4], which is established by an interpretation mapping into an abstract domain. The language is very close to a specific description logic called $\mathcal{SHOIQ}$. Existing sound and complete reasoning procedures for this logic can be used to provide reasoning support for OWL models.

As a result of the ongoing process of defining a standard ontology web language, a number of intermediate versions of the language have been defined. These languages differ in some features, but their underlying design principles are similar. As different transformation

appróaches refer to different language versions, we briefly outline the development at this point. For details about differences between the languages, we refer to the original language specifications.

OIL The Ontology Inference Layer OIL [5] has been developed in the On-to-Knowledge project [6] as a language to support logical reasoning with and about ontologies. The principle behind the language is to have different levels of expressiveness where the lowest level is equivalent to RDF schema.

DAML The DARPA Agent Markup Language DAML [7] was developed in parallel with OIL. The goal was to provide a common basis for the DAML project. DAML primarily originated from the area of frame-based knowledge representation systems and hence it is less influenced by formal logic than OIL.

DAML+OIL The name DAML+OIL resulted from the decision to create a joint language on the basis of existing DAML and OIL. The language replaced DAML as the standard language of the DAML project and was defined in an official committee. Two versions of DAML+OIL exist, the March 2001 version [8] being more widely supported than the December 2000 version [9].

OWL The Web Ontology Language OWL [10] is the label under which DAML+OIL undergoes the process of becoming a W3C recommendation. OWL is based on DAML+OIL with different levels of expressivity included in the language. There exist two versions of the language: OWL Lite that includes restricted vocabulary sufficient to satisfy primarily user needs and OWL Full with complete OWL vocabulary.

2.3 Conclusions

By comparing the natures of the two languages we can draw two conclusions. On the one hand they complement each other. While UML is designed for model building by human experts, OWL is designed to be used at run time to provide guidance for intelligent processing methods. This complementary character justifies the idea of combining OWL and UML in order to overcome the acquisition bottleneck. On the other hand, we can see that the translation is less than trivial, because of the differences between the two languages. The first challenge is to identify corresponding elements in the two languages, which will sometimes be difficult, because UML is biased by the later implementation. Second, we have to make sure that translations are backed by the semantics of the languages.

3 Existing Transformation Approaches

The benefits of using UML and DAML+OIL together have led to research where (1) *UML is used as a modelling syntax for knowledge representation languages such as DAML [11]*, or (2) *standard UML models are transformed into OIL or DAML+OIL ontologies [12], [13].*

The first direction arose due to the fact that current ontology representation languages lack representation possibilities such as a graphical front-end. As a recently emerged language, DAML does not have sufficient tool support, while UML is a commonly accepted graphical notation that can be used to build DAML ontologies. It was discussed in the previous section that UML and ontology representation languages have certain incompatibilities.

These incompatibilities prevent the representation of all elements from the ontology representation languages with the help of UML. In order to avoid this a number of extensions to UML have been proposed.

The second direction addresses the problem of reusing previously specified knowledge. UML models cannot be easily exchanged over the Web, the reasoning possibilities with UML models being also quite restricted (see our arguments in section 4.2). To overcome these difficulties [13] and [12] propose transformations between UML and ontology representation languages.

These two directions induced the need to generate transformation rules between UML and ontology representation languages. Differently aimed matching between languages caused differences in the proposed mappings. While the first direction requires UML to be suited for representing DAML ontologies, the second one addresses the question of how UML concepts can be represented with an ontology representation language while preserving their semantics as much as possible. In the rest of this section we introduce these approaches and discuss transformation issues in more detail.

3.1 UML-Based Tools for DAML

The first direction is represented by the work of Baclawski [11], where the relationships between UML and DAML have been investigated in order to provide support for visualizing complex ontologies and managing the ontology development process. Baclawski proposes a mapping between the two languages and gives a solution to the one of the most problematic differences between the languages. The mapping is done from the UML point of view while assuming that UML is used to represent DAML ontologies. As a consequence of this assumption a number of the UML class diagram elements are not presented in the mapping table. Although the paper focused at DAML, it remains valid for DAML+OIL.

It was argued in all related investigations that the biggest obstacle on the way to map UML and ontology representation languages is the notion of `Property`. From the first glance the notion of `Property` in knowledge representation languages corresponds to the notion of `association` in UML. `Property` is a first-class modelling element in all web-based knowledge representation languages starting from RDF(S). This means that `Property` can exist as an independent element of a language without being attached to other constructs (such as classes). At the same time UML associations are connected to classes with association ends. An association cannot exist without explicit connections to classes. It also means that each association is unique. Even if two associations have the same name they are connected to different classes and thus they are considered to be different. Thus, UML associations express local restrictions on instances of classes, while DAML properties in combination with other elements can express local as well as global restrictions. An association has only one domain and range, DAML `Property` can have more than one domain class. This makes it difficult to propose a mapping that is correct at the conceptual level. Different approaches have been proposed to tackle this problem, as discussed below.

To handle the difference between the notions of UML `association` and DAML `Property` Baclawski proposes an extension to the UML meta-model [11]. He suggests to enrich the Meta-Object Facility (MOF) specification [14] with the notions of `Property` and `Restriction` as it is shown in Figure 1.

Both `Property` and `Restriction` are modelled on the diagram as UML

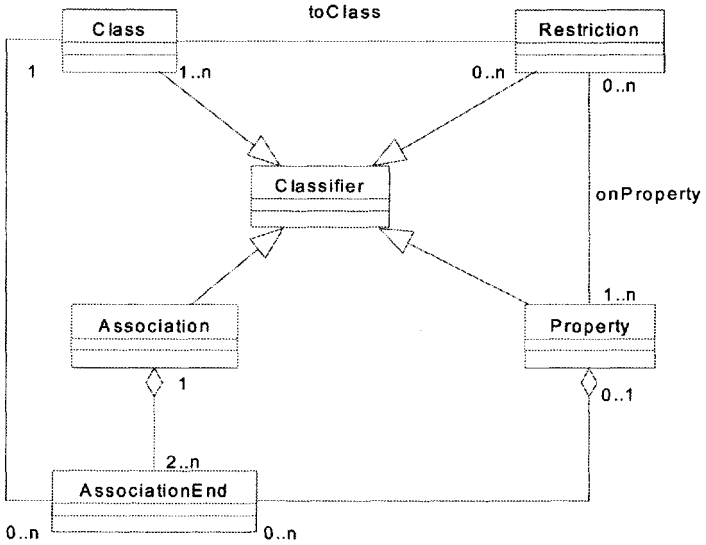


Figure 1: UML extension for property restrictions

`Classifiers`, enabling them to become first-class modelling primitives in UML. Also `Property` is modelled as an aggregation of zero or more association ends, allowing it to exist without being connected to a class. Each association end can be described by at most one `Property`.

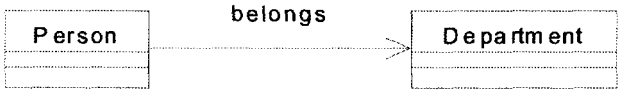


Figure 2: DAML property restriction as UML diagram

According to the diagram, the notions of `Property` and `Association` are overlapping to some extent, however `Property` relates to a DAML element. `Property` can be constrained by zero or more `Restrictions`. A `Restriction` can be related to one or more classes. This corresponds to a DAML construction of a property restriction to a class. Consider the diagram in Figure 2. It represents a restriction on the property `belongs` to the class `Department`.

By applying UML to DAML mapping the DAML translation of the UML diagram will result in a section of an ontology, depicted in the following. This example gives the short scenario of how the transformation works.

```
<daml:Class rdf:ID="Person">
    <rdfs:label>Person</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction >
            <daml:onProperty rdf:resource="#belongs"/>
```

```
            <daml:toClass rdf:resource="#Department"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="Department">
    <rdfs:label>Department</rdfs:label>
</daml:Class>

<daml:Property rdf:ID="belongs"/>
```

The inclusion of the proposed extension into a new UML 2.0/MOF 2.0 specification will have an impact on existing tools, but the authors believe that this impact will not be much bigger from the one anyway caused by the new specification. Moreover, it can lead to the general acceptance of UML as a modelling tool for ontologies.

3.2 Transformations of Standard UML Models into Ontology Representation Languages

We discuss two representative approaches in this direction: the first proposes a mapping that supports ontology extraction from UML diagrams; the second reports on using UML diagrams for modelling domain specific knowledge.

Extracting DAML+OIL Ontologies from UML Diagrams Falkovych [13] proposes a transformation of UML diagrams into DAML+OIL ontologies with preserving the semantics of UML concepts. This work emerged from an observation that there exist large sources of ontological knowledge already available in design documents of existing applications. Ontology extraction from UML diagrams and converting them into a powerful ontology representation language would enable reasoning about these ontologies and would allow knowledge interchange between heterogeneous environments. Hence, the main contribution of [13] lays in introducing a way of translating UML class diagrams into RDF(S) and DAML+OIL. Since RDF(S) can be considered as a subset of DAML+OIL, the UML to RDF(S) transformation is not discussed here.

All the issues about the differences between UML and DAML (DAML+OIL) discussed in Baclawski's work [11] are also relevant here. But the purposes of the work force us to look from the different perspective at the transformation task. The mapping is proposed for every UML element, which is possible to map into DAML+OIL constructs. For the complete mapping table the reader is referred to [13]. In order to have better understanding about the specificity of the mapping, the main issues are discussed below.

Since association is unique in UML, it is mapped into `daml:ObjectProperty` with a unique identifier for the property name. This makes it possible to distinguish different associations with the same name as different DAML+OIL properties.

UML attributes are mapped in a similar way, since each attribute has a local scope within its class. An attribute is mapped into `daml:DatatypeProperty` with a unique identifier attached. The rationale of mapping an attribute only to `daml:DatatypeProperty` and not to `daml:ObjectProperty` is that usually the type (range) of an attribute is a data type. Although in UML an attribute can have an object as its type, this situation occurs infrequently. Thus, in this mapping the assumption was made that all properties that have an object as a range are modelled as associations and not as attributes.

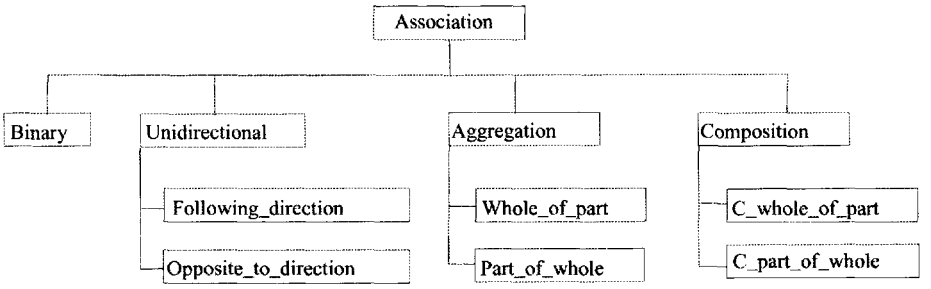


Figure 3: Taxonomy of association types

In order to preserve semantics of UML associations and to distinguish association ends, a taxonomy of association types is introduced (Figure 3). The notion of an association in the taxonomy is divided into four subtypes, namely binary association, unidirectional association, aggregation and composition, which are decomposed further into specific subtypes introduced for the mapping purposes. At the lowest level of decomposition, which is not present in the figure, all association types can have a name or be unnamed and they can have role names attached to association ends. All associations in a particular diagram are modelled as sub-properties of corresponding association types from the taxonomy.

Figure 4: Unnamed binary association with role names and multiplicity constraints

A binary association is navigable in both ways, thus it is mapped into two `daml:ObjectProperty` elements, which are inverse (`daml:inverseOf`) of each other. These two properties have a name of an association with the unique identifier attached to it (or just an identifier that serves as a property name in the case of an unnamed association) and they are distinguished by adding an underline symbol ('_') to one of them. The need to distinguish them comes from the need to map possibly present role names and multiplicity constraints attached to the ends of an association. Roles give some additional meaning about roles of classes in a relation, thus a role is specified as `rdfs:subClassOf` of an association it is attached to. Cardinality constraints are specified for associations as well as for roles. The example below clarifies these issues.

```
<daml:Class rdf:ID="Boundary point">
    <rdfs:label>Boundary point</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction daml:minCardinality="1" >
            <daml:onProperty rdf:resource="#has_G.4"/>
            <daml:toClass rdf:resource="#Boundary segment"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
```

```
        <daml:Restriction daml:minCardinality="1" >
            <daml:onProperty rdf:resource="#_G.2"/>
            <daml:toClass rdf:resource="#Boundary segment"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:ObjectProperty rdf:ID="has_G.4">
    <rdfs:subPropertyOf rdf:resource="#_G.2"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="_G.2">
    <rdfs:subPropertyOf rdf:resource="#binary_unnamed"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Boundary segment">
    <rdfs:label>Boundary segment</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1" >
            <daml:onProperty rdf:resource="#form_G.3"/>
            <daml:toClass rdf:resource="#Boundary point"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1" >
            <daml:onProperty rdf:resource="#G.2"/>
            <daml:toClass rdf:resource="#Boundary point"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:ObjectProperty rdf:ID="form_G.3">
    <rdfs:subPropertyOf rdf:resource="#G.2"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="G.2">
    <rdfs:subPropertyOf rdf:resource="#binary_unnamed"/>
</daml:ObjectProperty>
```

A unidirectional association is mapped in the same way with the difference that it is navigable in one way and thus two sub-properties of this association are distinguished (`Following_direction` and `Opposite_to_direction`). These sub-properties are not the inverse of each other. The same issues apply to aggregation and composition where different sub-property names are used to distinguish two ways of readability.

Domain-Specific Transformations An increasing interest in distributed problem solving for configuration tasks arose the need to represent configuration services as Web Services. To deal simultaneously with multiple service providers over a network and to interact with other configurators a commonly used representation formalism is required. Semantic Web languages such as OIL or DAML+OIL seem to be good candidates for such formalism. At the same time configuration system modelling requires a participation of domain experts and knowledge engineers. Knowledge representation languages are difficult to use during the modelling stage. The work of Felfernig [12] provides a set of rules to automatically transform configuration models represented in UML into an OIL representation. As such this

research corresponds to the second direction of existing transformation approaches. UML elements for which the mapping is proposed constitute a configuration ontology. The set of elements required for UML configuration profile consists of classes, generalizations, associations and aggregations (part-whole relationships). Compatibilities and requirements are introduced through the stereotype associations `incompatible` and `requires`. Resources are described by a stereotype `Resource` with `produces` and `consumes` dependencies to component types. The mapping of basic language constructs is similar with other approaches. The specific issue, which we would like to discuss here, is a way of treating UML part-whole relationships in OIL. The example below illustrates this (Figure 5):

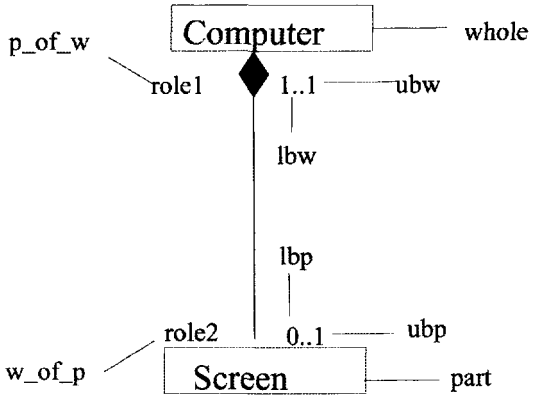


Figure 5: Graphical representation of part-whole relationships (taken from [12])

```
slot-def w-of-p
    subslot-of Partofcomposite
    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 lbw w,

p:slot-constraint w-of-p max-cardinality ubw w,

w:slot-constraint p-of-w min-cardinality lbp p,

w:slot-constraint p-of-w max-cardinality ubp p,
```

Two classes involved in a relation are distinguished as 'whole' and 'part'. Roles attached to the part-whole relation are denoted with `part-of-whole (p-of-w)` and `whole-of-part (w-of-p)` names. Multiplicities are represented through lower and upper bounds of part and whole. Aggregation and composition are considered being disjoint. An additional restriction in OIL is specified to indicate that a 'part' component that is connected

to a 'whole' component through a composition cannot be connected to any other component. The `source` and `target` classes of a relation are mapped into the `domain` and `range` OIL elements that occur due to the restrictive abilities of OIL with respect to DAML+OIL. The distinctive feature of DAML+OIL is the ability to define local constraints of domain and range, which correspond more to the UML notion of an association (see section 3.1).

4 Discussion

4.1 Summary of the Comparison

After introducing the main features of the approaches in the sections above we would like to underline differences and similarities between them.

Differences: The main difference between the approaches is the way they treat the semantics of UML.

Baclawski aims at providing a presentation syntax for DAML. He uses stereotypes in the mapping and proposes the extension to UML meta-model to make UML more expressive. The extension closes the gap between the two languages and makes UML more suited for representing DAML ontologies.

Falkovych approaches the problem in a different manner. The semantics of different kinds of associations are included in a taxonomy, which is then explicitly transformed into a DAML+OIL ontology. Every property that corresponds to a certain kind of an association in a UML diagram is defined as a sub-property of a corresponding association type from the taxonomy. In this way the semantics of associations is preserved after performing the transformation from one language to another. Being part of the resulting ontology, they can be used for example to ask for the relation between certain entities.

In the Felfernig work we saw an application of a transformation task to the configuration domain. His research shows a specific way of treating UML diagram elements representing a configuration model in OIL. The problem of the correspondence between UML association and DAML property is introduced here through the example of part-whole relationships. Since part-whole relationships are important for the configuration domain, Felfernig proposes the way to explicitly represent the intended interpretation of these relationships in OIL. He chooses a specific ontological interpretation of the part-whole relation that is most suited for the domain at hand.

Similarities: The similarities between the mappings are shown in Table 1. In the table we present an intersection of the mappings in order to specify the set of elements which are mapped similarly in all the approaches.

4.2 Conclusions and Future Work

Comparing different approaches for transforming UML models into ontology languages for the semantic web, we recognized that there is an agreement on how to translate a subset of UML class diagrams in general (see Table 1). Beyond this common core, different approaches take very different decisions of how to translate constructs. The actual choice for a specific translation is always a result of the purpose of the translation. We think that this situation

OIL [12]	**DAML+OIL** [13]	**Pure UML**	**DAML** [11]
class-def c	daml:Class	class	daml:Class
class-def c	rdfs:label	class name	rdfs:label
subclass-of	rdfs:subClassOf	generalization	rdfs:subClassOf
slot-def a	daml:DatatypeProperty	attribute	daml:ObjectProperty or daml:DatatypeProperty
slot-constraint a cardinality 1 d	daml:toClass	attribute type	daml:toClass
slot-constraint	daml:ObjectProperty	binary /unidirectional association	daml:ObjectProper
domain	daml:Class rdfs:subClassOf	source class of an association	daml:Class rdfs:subClassOf
range	daml:toClass	target class of an association	daml:toClass
cardinality & cardinality	**multiplicity & cardinality**		
min-cardinality 1 max-cardinality 1	daml:cardinality="1"	1	Unique/UnambiguousProperty daml:cardinality="1"
absence	absence	0..*	absence
min-cardinality 1	daml:minCardinality="1"	1..*	daml:minCardinality="1"
min-cardinality 0 max-cardinality 1	daml:minCardinality="0" daml:maxCardinality="1"	0..1	daml:minCardinality="0" daml:maxCardinality="1" Unique/UnambiguousProperty
min-cardinality n max-cardinality n	daml:cardinality="n"	n	daml:cardinality="n"
min-cardinality m max-cardinality n	daml:minCardinality="m" daml:maxCardinality="n"	m..n	daml:minCardinality="m" daml:maxCardinality="n"

Table 1: Overview of similar transformation choices.

that has naturally evolved in the parallel work of different authors actually points towards a successful strategy for the transformation task. Such a strategy should start from the common core mentioned above and make more specific transformation choices based on the intended use of the resulting ontology.

Apart from this general strategy for transforming UML diagrams, the main insight we got when looking at existing approaches is the fact that there are still many open questions concerning the transformation between UML and OWL. We discuss open issues we consider important in the following.

Implementation The question of how to implement these transformations comes naturally, and indeed we can report distinct approaches in this sense. A very simple way to perform these transformations is to take advantage of the XML encoding used in XMI (the serialization of UML) and DAML and to write an XSLT file that defines the transformation between the two tree structures. This method was used in [2] to translate XMI to RDF(S) and java files. We used this method in transforming XMI to DAML+OIL, as presented in [13]. In spite of the simplicity of this implementation, our experience shows that XSLT is quite cumbersome to use when one wants to express more complex mappings. In addition, it is very sensitive to the format of the input file.

There exist at least two initiatives which provide plug-ins to UML editors extending their functionality with the possibility to export UML diagrams into DAML. Within the CODIP project [1] a plug-in was developed which can be used both with Rational Rose and with another UML editor, ArgoUML. Similarly, the UBOT [2] tool suite includes a module for translating UML to DAML. These tools are currently in their early development stages and we expect more effort to be required in developing robust tools for such translations.

[1] `http://sylvester.va.grci.com/codipsite/codipsite/index.html`
[2] `http://ubot.lockheedmartin.com/ubot/`

Reasoning Existing UML editors, such as RationalRose,[3] offer basic services for checking UML models for syntactic errors. The range of support of checking the correctness of a model gets considerably wider if the model is translated to a formal language.

First, one can check the consistency of the models. This aspect is especially important when it comes to ontology development and testing. Indeed, existing ontology editors offer a consistency checking service. In the UBOT project the ConsVISor [15] tool is used to check the consistency of the modelled ontologies. Based on Prolog (and planned to be migrated to Jess), this tool returns both parsing and consistency errors. The SNARK theorem prover is used to check logical inconsistencies. Similarly to UBOT, the OilEd ontology editor uses FaCT (Fast Classification of Terminologies) [16] for consistency checking. FaCT is a Description Logic (DL) classifier that can also be used for modal logic satisfiability testing.

Second, a formal representation allows deriving new knowledge by using inference engines built for that particular representation. For example FaCT can derive new facts given an original model. A direct way of assessing the quality of the existent transformation schemes would be to (1) check the quality of their output and (2) measure the amount of information that can be derived from their outputs.

Beyond Class Diagrams The mappings that we have described address the content of class-diagrams. However, as we have stated earlier in this paper, UML is much richer. The class diagrams have been in the center of attention because there exists a direct relation between their elements and the parts of an ontology (classes, hierarchies, class attributes). However, other types of diagrams express different types of information. For example state-charts or activity diagrams are useful for service and process related ontologies. In this sense a proposal exists to model DAML-S using UML diagrams.[4] There is no doubt that other UML diagrams can also be used to model various knowledge aspects of the Semantic Web.

Rules UML provides a language for expressing rules, OCL. OCL is more complex than the diagrams, however we can suppose that UML users have more experience with OCL then with DAML. Also previously developed ontologies often describe axioms using OCL. These considerations imply that the translation from OCL to DAML would be beneficiary both for supporting modelling and exploiting existing models. The research in this direction is hampered by two major problems. First, OCL does not have formal semantics and therefore it is difficult to map it into a formal language. However there are more proposals for a formal semantics for OCL [17]. Second, DAML is not developed to express rules. However there is research going on to develop a rule language for the Semantic Web (DAML-L, RuleML).

Extended support for modelling With the existent transformations it is not guaranteed that transforming a UML model to DAML and then transforming the DAML file back to UML would lead to the same diagrams. A transformation that would preserve all information would allow modelling and inferencing in parallel: a UML model would be transformed to a formal language, new information would be derived and then presented back to the modeler in UML format. This is a desirable functionality for powerful modelling tools.

[3] http://www.rational.com/

[4] http://codip.grci.com/codipsite/wwwlibrary/DUETGuide/DAMLS-UML_Mapping_V1.htm

Another observation is that the elements of class-diagrams are only enough to model light-weight ontologies. Enriching these ontologies with rules and axioms would be possible with OCL. Therefore, to build complex ontologies using UML the modelling tools have to support transformations both for class-diagrams and OCL rules.

Acknowledgements

We would like to thank Lynda Hardman, Martin Gogolla and Borys Omelayenko for valuable comments for improvement and Frank van Harmelen for suggestions in early stages of the work.

References

[1] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 35–43

[2] Cranefield, S.: UML and the Semantic Web. In: Proceedings of the International Semantic Web Working Symposium (SWWS). (2001) `http://www.semanticweb.org/SWWS/program/full/paper1.pdf`.

[3] Object Management Group: Unified Modeling Language (UML). (`http://www.omg.org/uml/`)

[4] Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics. In Brewka, G., ed.: Principles of Knowledge Representation. Studies in Logic, Language and Information. CSLI Publications (1996) 193–238

[5] Fensel, D., Horrocks, I., van Harmelen, F., McGuinness, D.L., Patel-Schneider, P.F.: OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems **16** (2001) 38–44

[6] Fensel, D., van Harmelen, F., Ding, Y., Klein, M., Akkermans, H., Broekstra, J., Kampman, A., van der Meer, J., Sure, Y., Studer, R., Krohn, U., Davies, J., Engels, R., Iosif, V., Kiryakov, A., Lau, T., Reimer, U., Horrocks, I.: On-To-Knowledge in a Nutshell. IEEE Computer (2002)

[7] Hendler, J., McGuinness, D.L.: The DARPA Agent Markup Language. IEEE Intelligent Systems **15** (2000) 67–73

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

[9] van Harmelen, F., Horrocks, I.: Reference description of the DAML+OIL ontology markup language. `http://www.daml.org/2000/12/reference.html` (2000)

[10] Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Web Ontology Language (OWL) Reference Version 1.0. Working draft, W3C (2002) `http://www.w3.org/TR/owl-ref/`.

[11] Baclawski, K., Kokar, M., Kogut, P., Hart, L., Smith, J., Holmes, W., Letkowski, J., Aronson, M.: Extending UML to Support Ontology Engineering for the Semantic Web. In: Fourth International Conference on UML, Toronto (2001)

[12] Felfernig, A., Friedrich, G., Jannach, D.: UML as domain specific language for the construction of knowledge based configurations systems. International Journal on Software Engineering and Knowledge Engineering **10** (2000) 449–470

[13] Falkovych, K.: Ontology Extraction from UML Diagrams. Master's thesis, Vrije Universiteit Amsterdam (2002)

[14] Object Managemet Group: Meta-Object Facility (MOF) Specification. `http://cgi.omg.org/docs/formal/00-04-03.pdf` (2002)

[15] Baclawski, K., Kokar, M., Waldinger, R., Kogut, P.: Consistency Checking of Semantic Web Ontologies. In: First International Semantic Web Conference (ISWC 2002), Sardinia, Italy (2002)

[16] Horrocks, I.: The FaCT system. In de Swart, H., ed.: Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98. Number 1397 in Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin (1998) 307–312

[17] Cengarle, M., Knapp, A.: A Formal Semantics for OCL 1.4. In Gogolla, M., Kobryn, C., eds.: UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference Proceedings. Volume 2185 of LNCS., Toronto, Canada, Springer (2001) 118–133

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Knowledge Representation and Transformation in Ontology-based Data Integration

Silvana Castano Alfio Ferrara

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

Abstract. This chapter describes an ontology architecture for the integration of heterogeneous XML data sources. In this architecture, the information about DTDs and their contents is represented at a semantic level, by means of a semantic mapping scheme and a mediation scheme. We first describe information transformation techniques for designing such ontology schemes for heterogeneous XML data. Then, we present a rule-based approach for deriving a DAML+OIL representation of the ontology, 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 and that can be used for different intelligent services like information search, retrieval, and transformation. In this chapter, we focus on the use of ontologies for the integration of heterogeneous XML data sources. The advent of the Web and 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 in cases where semantic heterogeneity exists and for the management of semantics that capture the meaning of XML data appropriately [3, 4].

In this chapter, we discuss our reference architecture of a domain ontology for XML data source integration [5], where information about DTDs and their contents is 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 system [6, 7, 8]. In this chapter, we first summarize information transformation issues for abstracting XML DTDs into a high level representation formalism (called *X-formalism* [3]). Then, we describe the schema matching and unification techniques for deriving the semantic mapping scheme and the mediation scheme. Then, we introduce a rule-based approach to represent the ontological knowledge in the DAML+OIL ontology

language standard proposal [9, 10]. In this way, ontological knowledge can be exported to Semantic Web in a uniform and standardized way, ensuring ontology interoperability.

The chapter 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 data source 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, in Section 6 we give our concluding remarks.

2 Ontology-based XML Data Integration

We consider a Web-based scenario, where XML is the standard adopted for data exchange among different data sources in a given domain. We assume that, for each data source, the data to be integrated is described by 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*. This ontology organizes knowledge about (the content of) the DTDs of data source (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 data sources, and is organized as a network of *concepts* and *links*. In the semantic mapping scheme network, we have sets (clusters) of semantically related data source concepts. These concepts are formally represented as *X-classes*, which are defined according to the *X-formalism* that we adopt for abstracting DTDs [3]. In the mediation scheme network, we have global concepts, formally represented as global X-classes, which provide an integrated representation of the underlying data source concepts. A global concept has associated *mapping rules* that specify how to map the structure of the global concept to the structure of the data source concepts in its associated cluster. Semantic links express 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 can express several relationships: 1) an equivalence relationship between similar data source concepts forming a cluster in the semantic mapping layer; 2) a generic semantic relationship between global concepts; 3) 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.

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

3 Ontology Design

The domain ontology is built in a bottom-up process, articulated in the following steps: (1) **DTD abstraction**, where DTDs associated with data sources are abstracted into a set of X-classes (using a reference formalism called X-formalism); (2) **semantic DTD matching**,

```
<!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
```

Figure 1: An example of DTD of a real data source in the TV entertainment domain

where X-classes of different DTDs are clustered based on semantic mappings that are established between them; (3) **mediation scheme design**, where X-classes which belong to the same cluster are abstracted and unified into *global X-classes*. In this section, we first summarize the 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 [3, 7, 8].

3.1 Reference Formalism for DTD Abstraction

To build the domain ontology, we assume that the XML data that has to be integrated is associated with 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]. This formalism allows one to describe the contents of a given DTD at a high abstraction level, by capturing the semantics of various elements within the DTD through the concepts of *X-class*, *property*, *attribute*, *link*, and *referenced X-class*. In particular, 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}, $t_p \in$ {`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 ($P_{xc} \cup R_{xc}$).

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

```
<!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
```

Figure 2: An example of DTD of a real data source in the TV entertainment domain

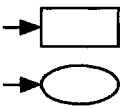
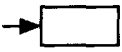
X-formalism Concept	Graphical representation	Description
X-class		Corresponds to an element declaration with element content in a DTD.
Property		Corresponds to an element with PCDATA, any, or empty content in a DTD.
Attribute		Corresponds to an attribute defined in the attribute list declaration of an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

Table 1: Basic X-formalism concepts

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 chapter, 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. A description of the basic X-formalism concepts and of their graphical representation is shown in Table 1. The X-formalism representation of an X-class, a property, and an attribute is a rectangular, oval, and double oval node, respectively. A link is represented as an oval (or rectangular) node depending on its type, labeled with the name of the link and connected by a double directed arc to the appropriate X-class node. An XML DTD union structure ('|') is represented as an `or`-labeled dashed arc crossing the arcs entering the class/property nodes involved in the union. An XML DTD sequence structure with a cardinality constraint that applies to the whole structure is represented as an `and`-labeled arc that crosses the arcs entering the involved X-class/property nodes. Finally, cardinality constraints associated with properties, links, attributes, and referenced X-classes are represented explicitly — if different from $(1, 1)$ — as labels attached to arcs; the cardinality $(1, 1)$ is implicitly taken as a default.

As an example, consider the DTDs illustrated in Figure 1 and 2, which describe information about two real TV entertainment data sources. Their X-formalism graphical representation is shown in Figure 3 and in Figure 4, respectively. In particular, 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`.

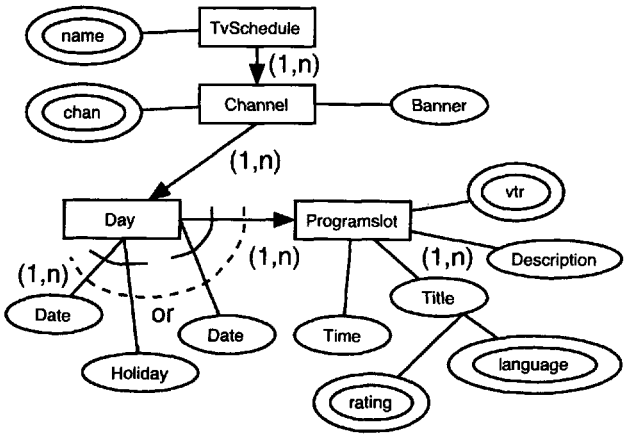


Figure 3: X-formalism representation of the DTD in Fig. 1

3.2 *Ontology Design Techniques*

Schema matching techniques. The first step in the construction of the ontology consists of defining the semantic mapping scheme. This involves the comparison of X-classes of DTDs associated with different data sources to find similarities between them. To this purpose, ARTEMIS provides schema matching techniques based on the concept of *affinity* to identify X-classes that have a semantic relationship. Affinity evaluation in ARTEMIS exploits a thesaurus of weighted terminological relationships (e.g., synonymy, hyponymy) which express inter-schema knowledge and characterize source DTDs. In particular, a validated thesaurus [7] is defined to contain the following kinds of weighted terminological relationships: i) automatically extracted relationships from WordNet for names of X-classes and features (basic ontological knowledge); ii) automatically extracted relationships about X-classes and structure (schema knowledge); iii) manually supplied relationships 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 overlapping). 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 higher 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 is referred to [7, 8] for a detailed discussion of affinity-based schema matching and some experimental results. For instance, with a reference to the DTDs in Figures 3 and 4, 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`}, while the remaining X-classes are placed in singleton clusters.

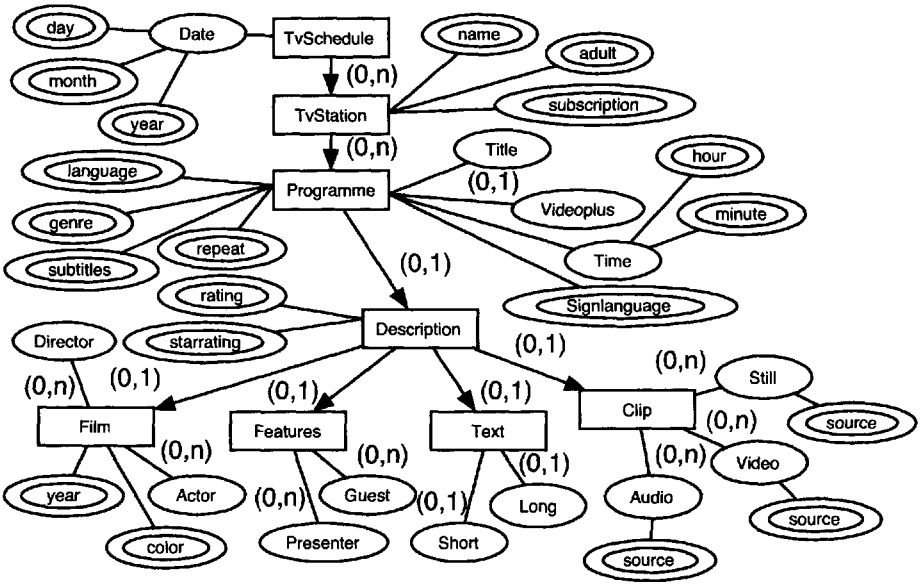


Figure 4: X-formalism representation of the DTD in Fig. 2

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. In this stage, we identify for each global X-class its basic features, namely, attributes, properties, and links. The second phase of the mediation scheme design process is devoted to the revision of preliminary global X-classes and to the identification of their structure and links to referenced X-class. Finally, mapping rules between global X-classes and corresponding local X-classes that belongs 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 the properties, attributes, and referenced classes of the X-classes that belong to it by applying a set of basic *reconciliation rules*. The purpose of these rules 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, and subsequently considers non-homologous features. A detailed description of the reconciliation rules for mediation scheme design is described in [5]. As an example, the global X-class

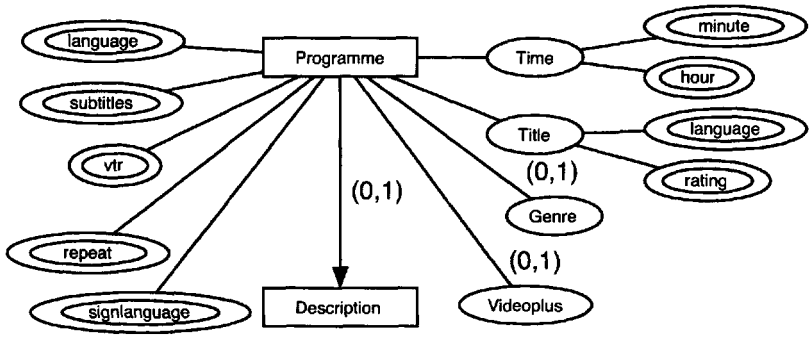


Figure 5: Example of global X-class

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

Programme obtained from the reconciliation of the X-classes of cluster cl_2 is shown in Figure 5. Finally, to complete the global X-class definition, *mapping rules* are defined that specify how to map global X-classes onto the structure of original X-classes. For each global X-class, a persistent *mapping-table* that stores all the mapping rules is generated; it is a table which columns represent the set of the local X-classes that belong to the cluster associated with a given global X-class and which 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 a set of mapping rules for the global X-class Programme is shown in Table 2.

4 Knowledge Representation

In order to increase ontology interoperability for Semantic Web, we provide a DAML+OIL-based [10] representation of our ontology. In particular, we provide a set of rules to automatically derive the DAML+OIL representation of the semantic mapping scheme and of the mediation scheme, starting from the X-formalism specification of their contents.

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

The basic 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 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, respectively. We also 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. A summary of these basic DAML+OIL concepts is given in Table 3.

4.2 The Rules for Representing the Semantic Mapping Scheme

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 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.

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

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 a 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 translated 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 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 class `Property`. According to the same process, properties `hasLink` and `hasAttribute` are used for each link $l_i \in L_{xc}$, and for each attribute $a_i \in A_{xc}$. Finally, property `hasRefX-class` is used

```
<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>
```

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

```
<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>
```

Figure 7: DAML+OIL representation on the cluster cl_2

for each referenced X-class $r_i \in R_{xc}$ [2]. An example of DAML+OIL specification of the referenced X-class `S2.Programme` is shown in Figure 6.

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

4.3 The Rules for Representing the Mediation Scheme

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 refer-

[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`.

enced X-class of a global X-class gxc, the corresponding feature (if any) of each local X-class of the cluster associated with gxc. We define a basic DAML+OIL object property `MappingRule` to represent a mapping rule. For each property, link, attribute, and referenced X-class of gxc, a `MappingRule` property is specified with two object properties, `GlobalFeature` and `SourceFeature`, representing the feature of gxc and its corresponding feature of each involved local X-class, respectively. 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), no local feature is listed in the `SourceFeature` for such a local X-class.

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 defined 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` from Figure 5, considering the mapping rules reported in Table 2.

5 Related Work

A number of relevant approaches has been recently reported in the area of information integration. The information integration architectures based on mediator/middleware data models and ontologies have been proposed [12, 13, 14, 15]. More recently, some work has been reported on semantic representation and integration of XML data sources.

Cupid [1] is a schema matching system, which provides different matching techniques for discovering mappings between the elements of heterogeneous schema descriptions. Metadata representation is graph-based, using an extended Entity-Relationship model. Cupid computes similarity coefficients among the elements of different schemas using a combination of linguistic and structural matching techniques. In this system the focus is kept on schema matching techniques rather than on data source integration.

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. Then, semantic mappings between the elements of different DTDs are identified. Metadata representation at the global level is provided by the so-called *abstract DTDs* modeled as a tree. The proposed solution relies on human interaction in identifying mappings between DTD paths. Machine-learning techniques can be used for matching source DTDs against a pre-defined global schema in form of a DTD as described in [4]. The system asks the user to provide semantic mappings for a small set of data sources. A set of instance matchers, called *learners*, is constructed from these 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 based on XML schema trees. This system follows a 'local-as-view' approach to integration, where 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 medi-

```
<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>
```

Figure 8: Example of DAML+OIL specification for the global X-class `Programme` of Figure 5

ation 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 means of Description Logics. The integration is performed according to the local-as-view approach, and is focused on the representation of the mappings between the source schemas and the global schema.

The main contribution of our work is the following: (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 describing DTDs contents and their role following a data model approach (i.e., by distinguishing concepts like X-class, property, link); (iii) the use of DAML+OIL for ontology representation to ensure its compatibility and interoperability with the Semantic Web.

6 Concluding Remarks

In this chapter we have presented an approach for ontology-based integration of heterogeneous XML data sources, where ontology is organized into a semantic mapping scheme and a mediation scheme. We have described the ontology design techniques for deriving clusters of semantically related source concepts (semantic mapping scheme) and for deriving global concepts and links (mediation scheme). These techniques rely on the X-formalism, our representation of source DTDs for mapping DTD contents to the X-class constructs, properties, attributes, links, and referenced X-class, and on a thesaurus of terminological relationships specifying interschema knowledge. We use DAML+OIL to represent our ontology on the Semantic Web.

Future research work will be devoted to the development of a CORBA module for producing DAML+OIL specification of the ontology, in the framework of the ARTEMIS tool environment. Actually, ARTEMIS implements the ontology using ODL_{I^3}, an extension of the ODL data model [7]. We are studying how to derive the DAML+OIL representation of the ontology knowledge by considering directly ODL_{I^3} specifications.

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, Italy (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., De Antonellis, V., De Capitani di Vimercati, S.: 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 2002 Workshop, Aix-en-Provence, France (2002) 103–110

[6] The ARTEMIS Project home page: `http://isserver.usr.dsi.unimi.it/artemis/d2i/` (2001)

[7] Bergamaschi, S., Castano, S., Vincini, M., Beneventano, D.: Semantic integration of heterogeneous information sources. Data and Knowledge Engineering **36** (2001)

[8] Castano, S., De Antonellis, V., De Capitani di Vimercati, S.: Global viewing of heterogeneous data sources. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 277–297

[9] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L.: Daml+oil (march 2001) reference description (2001) `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) 2–13

[11] Castano, S., De Antonellis, V., De Capitani di Vimercati, S., Melchiori, M.: Semi-automated extraction of ontological knowledge from XML datasources. In: Proc. of DEXA WEBH Workshop, Aix-en-Provence, France (2002) 852–860

[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

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

A Logic Programming Approach to RDF Document and Query Transformation

Joachim Peer

Institute for Media and Communications Management,
University of St.Gallen, Blumenbergplatz 9, 9000 St. Gallen
Switzerland
joachim.peer@unisg.ch

Abstract. The Resource Description Framework (RDF) is an attractive tool for managing any kind of data, especially if multiple heterogenous vocabularies are involved. However, a standard way of either transforming RDF documents or transforming RDF queries does not exist yet. Several concepts have been proposed so far, each concentrating on different goals. In this chapter we present a concept following a Logic Programming approach, providing the amount of expressivity needed to build generic services for management of RDF data. We will focus on several transformation problems not satisfactorily discussed yet and we will provide an overview of the algorithms and data structures needed to solve these problems.

1 Introduction

Technical advancements and the success of the Internet and the World Wide Web in the last years have led 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 designed to be capable of explicitly expressing semantics of data from ground up. RDF does so by incorporating a very generic model of knowledge representation: It permits the formulation of *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 these definitions. 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 large number of 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 the 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 on both sides can lead to more efficient transformations.

However, XML is in use much more frequently 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 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 in the data model layer to each other to establish (domain-) specific conceptualizations. 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 chapter 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 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, specific algorithms for transforming RDF documents and RDF queries. The chapter is concluded by a short summary of findings and an outline of future work.

2 Related Work

Currently no standard way of transforming RDF-encoded documents or RDF queries exists, 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 individuals with individual. 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 towards 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

An approach to try to model RDF query and transformatione languages very closely to the XML transformation tools (XPath and XSLT) is taken by RDFPath/RDFT as 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 of 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*). To define the characteristics of an association made 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 the 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 to suppose 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 upheld. 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 of meaning between knowledge bases even 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* $\Rightarrow$ *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 with some of the heavyweight 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 and which facilitates a Logic Programming[1] approach is presented in [13].

A transformation between two documents is defined by a mapping definition M: M consists of a set of mapping rules T $\Rightarrow$ 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

[1]here we use the classical definition of this term, primarily referring to Horn Logic-based languages like Prolog and Datalog.

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 introducing 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 dealt with yet is the transformation of queries.

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 somewhere in the range between these two concepts, building on the findings presented in [13]. In addition 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

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

Firstly, we need to define the objects of interest for our transformation concept and how they are represented.

As already described in several papers (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 advice of [19] and define it as a ternary relation (which we will call *triple* throughout this chapter)

```
triple(Subject,Predicate,Object).
```

How do we presenting 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 to RDF querying fits the abilities of Logic Programming well. Queries can be expressed as (Horn-) rules, which may be asserted 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 specific 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) :-
    ruleheadl(arguments), ...
    ruleheadN(arguments).
query(arguments) :-
    ruleheadl(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).

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 based upon these facts. In general we can distinguish two top-level 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-relationship provided by RDF Schema (further examples for such rules are presented in [27]). Another type of rules may encapsulate *domain specific* knowledge. Individual rule 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 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 meta-model (i.e. RDF) we deal with translations of much lower inherent complexity. This allows us to formulate mappings not only as unidirectional implications (denoted as '⇒'), but also as equivalences (denoted as '⇔').

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 ⇔. 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 of a tranformation of e-business documents. We will use this example throughout this chapter for illustration purposes. The example makes use of two ontologies c and d, derived from XML-based e-commerce standards.

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 if instance data between 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 "syntactic 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 in order to more easily enable 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 inthe 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 transformation 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 into 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 into 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').
```

Once 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 above example, both query stores are processed according to our algorithm, 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, 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 one 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 kinds 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 for their specific bindings; the same is true for the second kind of rules. However, there are rules that can not 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 these semantics individually.

As an example let us consider the recursive defined rule `type` which reflects semantics of RDF Schema, particulary 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 the context of a query the modified semantics of the construct can lead to unwanted consequences, such as 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 can 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.

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 (these equivalences can be written down much more concisely 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)
<=>
```

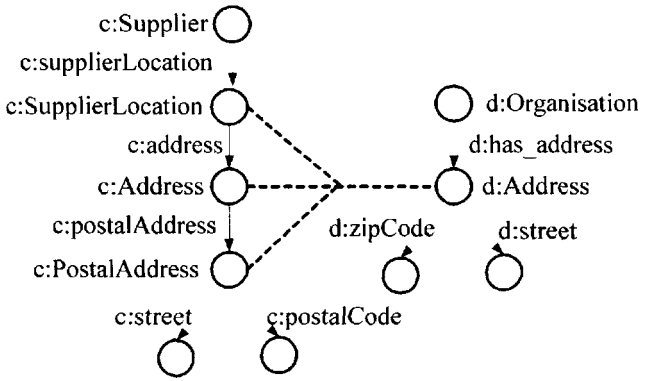


Figure 1: Example of structural differences

```
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 present our solution, we will briefly discuss the problems outlined 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 we agree 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 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 as indicated above. 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 transformation 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 would be inappropriate.

Each equivalence per se is logically correct (i.e. the relationships of the variables are correctly formulated in each equivalence), but in conjunction with other equivalences 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 between the logic variables used in the production rules.

To solve the problem of branching in the example above, we need to define the relationships between the potentially conflicting variables. This can be achieved using a fact (e.g. `dependence`) as follows:

```
dependence('c', ['X', 'SL', 'A', 'PA']).
```

This fact tells the translation engine that on 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 the variables it depends on (e.g. `X`, `SL`) with the 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 chosen rather than 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 to see 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 in the first version of our algorithm in Section 4.1.

In either 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

In this chapter we have described how a system of Horn-based matching and production rules can be used to transform arbitrary RDF documents and queries and we have introduced the algorithms required for correct transformation. We also discussed advanced issues i.e. the problems arising when transforming (RDF-) documents with substantial structural differences, in particular the problems of branching and conflation. We have shown the algorithms and data structures needed for such advanced cases.

As a proof of concept we have developed a first version of a prototype incorporating the concepts presented in this paper. Please contact the author of this chapter for further information.

However, there are many other issues related to transforming RDF documents that have not been discussed in this chapter; among these problems are the correct transformation of RDF collections and reified statements. These problems will be addressed by further research.

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) 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) 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)) 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)

[19] Hayes, P.: RDF model theory. (W3C Working Draft) 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: Proc. of the 2002 International Semantic Web Conference (ISWC 2002)

[26] Conen, W., Klasping, R.: A logical interpretation of RDF. (Electronic Transactions on Artificial Intelligence (ETAI) (under review))

[27] Zou, Y.: DAML XSB rule version 0.3. (working paper) http://www.cs.umbc.edu/~yzou1/daml/damlxsbrule.pdf.

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

RDFT: A Mapping Meta-Ontology for Web Service Integration

Borys Omelayenko

Department of Computer Science
Vrije Universiteit, De Boelelaan 1081, 1081hv,
Amsterdam, The Netherlands
borys@cs.vu.nl

Abstract. The Semantic Web needs to possess advanced techniques for mapping various web services to enable open enterprize integration on the Semantic Web. These services contain message sequences and messages derived from substantially different underlying domain models and cannot be straightforwardly mapped or converted. In this chapter we propose a mapping meta-ontology built on top of RDF Schema and targeted for describing and inferencing about the links between different services. The ontology contains the main web service concepts needed for the integration: events, messages, XML constructs. They are linked with bridges representing different relations between the concepts and several other mapping constructs. The few relations described in the chapter allow the most essential mappings and validation tasks to be performed.

1 Introduction

Historically business integration has been performed within the Electronic Document Exchange[1] (EDI) paradigm via costly Value-Added Networks that use private exchange protocols and provide a 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 enterprizes searching for fast and inexpensive solutions and expecting them to come from the Semantic Web area.

With some reasonable assumptions a pair of companies can be seen as a pair of Web Services producing and expecting certain messages as reactions to internal and external events. Most likely these services would be described on the Web in a unified way in the Web Service Definition Language[2] WSDL that is becoming a widely accepted standard. WSDL specifies a service as a set of input-output message sequences called `PortTypes`. `PortTypes` can be either request-response-failure, or response-request-failure, as well as one-way request or response. These messages themselves are essentially XML documents with their structure specified in XML Schema[3] included in WSDL service descriptions as message `Types`.[4]

[1] `http://www.x12.org`
[2] `http://www.w3.org/TR/wsdl`
[3] `http://www.w3.org/XML/Schema`
[4] WSDL also defines bindings of messages to event sequences and linking them to specific addresses and protocols that is beyond the focus of this chapter

Independent companies playing different roles and conducting various types of business are likely to support different message sequences with messages of different structures and containing different information. Building a mediator capable of receiving all the output messages of the companies and producing all the input messages required by these companies seems to be a standard solution path. The mediator needs to support two tasks: mapping WSDL message sequences and mapping XML Schemas of the messages bodies. The first task requires the mediator to represent the mappings between the messages and to relate these mappings to temporal constraints imposed by the companies. The second task requires the mediator to be capable of representing conceptual models of the XML Schemas and hence to be at least as expressive as XML Schema itself. The mediator needs to use an open and Web-friendly modelling language to serve as an open integration approach. There are numerous conceptual modelling languages, quite a few of which have the potential to become a world-wide standard for representing conceptual models on the Web. The lucky two are RDF Schema [1] and the Web Ontology Language OWL[5] built on top of RDF Schema. These two seem to be sufficiently suited the second task, while neither of them contains axioms required for the first task. We use Prolog as an axiom language to represent all the axioms and to inference over data models represented in RDF following a number of successful practices.[6]

The task of integrating different message sequences is relevant to the area of interorganizational workflow management [2] that concentrates on mapping rich workflow models. However, these models are normally not publicly available [3] and in this case the tools capable of mapping them are of little relevance. Moreover, the workflow integration solutions tend to be separated from the message integration process and special effort is needed to link them together [4].

Mapping XML Schemas of messages requires interpreting labelled trees of XML documents in terms of objects and their attributes. Several chapters of this book are devoted to different aspects of this problem. Some other recent work on mapping domain models in the Semantic Web context includes generic mapping framework [5] and a discussion of various important properties of the mappings [6].

There exist numerous commercial tools that approach the problem from the practical side. The MS BizTalk[7] 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 BizTalk Mapper that supports a user in designing the mappings and generates XSLT scripts translating instance XML documents according to the maps. Finally, the target database record or EDI document is then created from the target XML document.

Capestudio[8] contains an XML mapping tool helping the user to map a pair of XML schemas and automatically generating a correspondent XSLT file, similar to the Biztalk's Mapper. In addition, it provides advanced means for working with online services and WSDL service descriptions (the same functionality is achieved with wrappers and an SQL server in BizTalk's database integration scenario).

The new generation of tools, e.g. the Unicorn[9] toolkit, uses ontologies as structured vo-

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

[6] http://www.google.com/search?q=using+prolog+rdf

[7] http://www.biztalk.org

[8] http://www.capeclear.com/products/capestudio/

[9] http://www.unicorn.com/

cabularies that help the users to develop the mapping rules. In this case a user maps document elements to ontological concepts and uses hierarchies of ontological terms to navigate the elements. These terms are only used to provide mediating names for database attributes, and do not constitute a mediating conceptual model. For example, the tool cannot support a case when in the source database a single object is stored in a single record while in the target database the same object is split up into several records.

In this Chapter we first focus on modelling different kinds of objects: events, documents, and vocabularies as presented in Section 2. Then in Section 3 we describe our mapping meta-ontology for linking these objects followed by a sketch of available tool support and inference tasks in Section 4. The Chapter ends with a discussion of several open issues and final conclusions.

2 The Things to be Mapped

To integrate two or more services described in WSDL one needs to map WSDL events corresponding to activities, conceptual models of XML messages attached to these events, and different terminologies used in these documents [7]. A model for events, documents, and terminologies is presented in Figure 1 and discussed in the rest of this section.

2.1 Events

A WSDL service description contains explicit references to activities (the input-output sequences) and events (the messages) supported by a service provider. We need to model these explicitly and link them to a formal temporal theory to be able to reason about them and validate their appearance from a temporal perspective.

We model WSDL concepts in RDF Schema by directly re-encoding their representation in the standard WSDL XML Schema. These concepts are presented in Figure 2 and denoted with the `wsdl:` prefix. In the figure one can easily see that WSDL `PortTypes` link `wsdl:Messages` via the `wsdl:input` and `wsdl:output` properties.

We linked our WSDL concepts to the Process Specification Language[10] (PSL) temporal ontology that provides a formal theory of objects in time. PSL includes several main classes: `activity`, `activity_occurrence`, `timepoint`, and `objects` as well as a number of axioms. Activities are performed during certain time intervals marked with timepoints, and each activity occurrence specifies the status of that activity at a moment in time. The objects are defined as things that are not timepoints, not activities, and not activity occurrences, and hence do not possess temporal properties. They may participate in activities at certain timepoints as defined by the `activity-timepoint-object` relation.[11] PSL axioms provide the basic temporal semantics of time intervals and timepoints, as well as a number of temporal constraints.

Some of the PSL classes are presented in Figure 2, where PSL concepts are denoted with an appropriate prefix. In PSL activities are treated as higher-level concepts similar to WSDL

[10] `http://www.mel.nist.gov/psl/`

[11] RDF Schema is bound to binary relations and provides no means to specify the ternary `activity-timepoint-object` relation. To model it we have to introduce a special class `psl:activity_at_timepoint_relation` attached to `psl:object` and linked to both `psl:activity` and `psl:timepoint`.

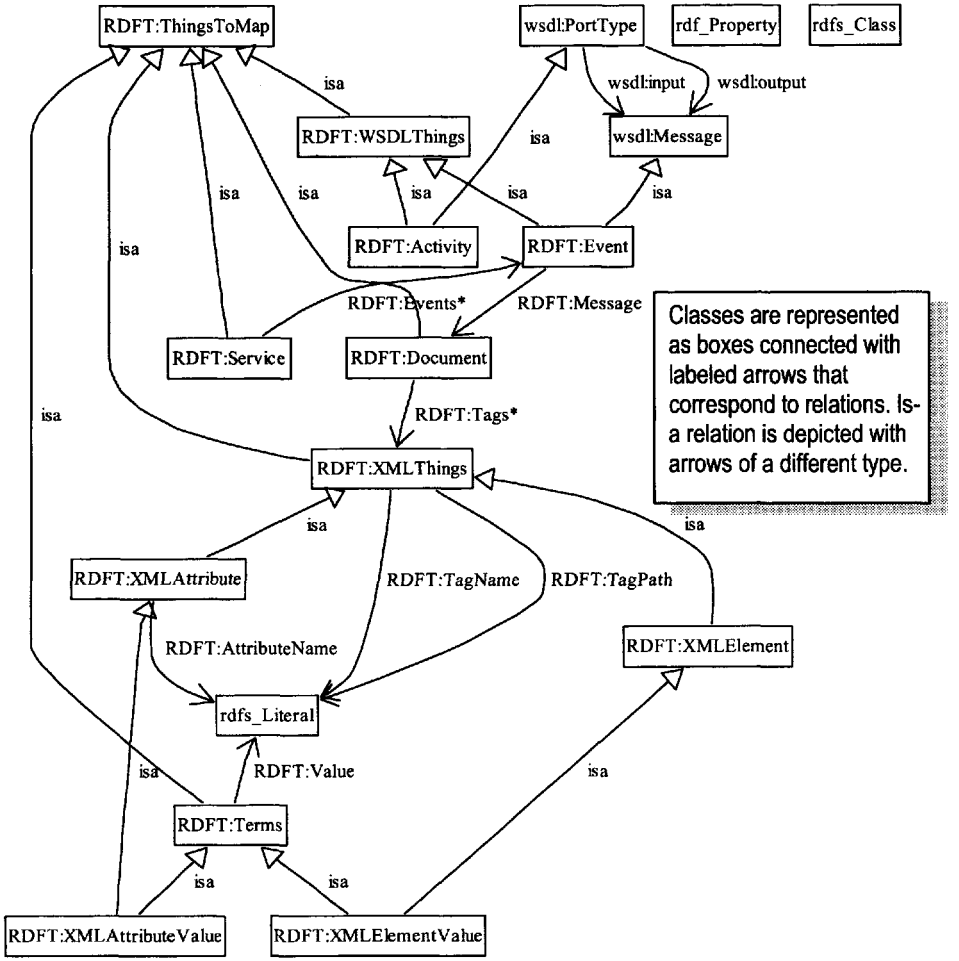


Figure 1: Things to map. The figure shows how we include both WSDL and XML Schema constructs in a single model prior the integration. We distinguish three types of `ThingsToMap`: generic RDF Schema classes and properties, `WSDLThings` that corresponds to WSDL events and activities (`RDFT:Activity` is a subclass of `wsdl:PortType` and `RDFT:Message` is a subclass of `wsdl:Message`), `XMLThings` that represent XML element and attribute definitions, `Terms` that correspond to string constants belonging to regular vocabularies.

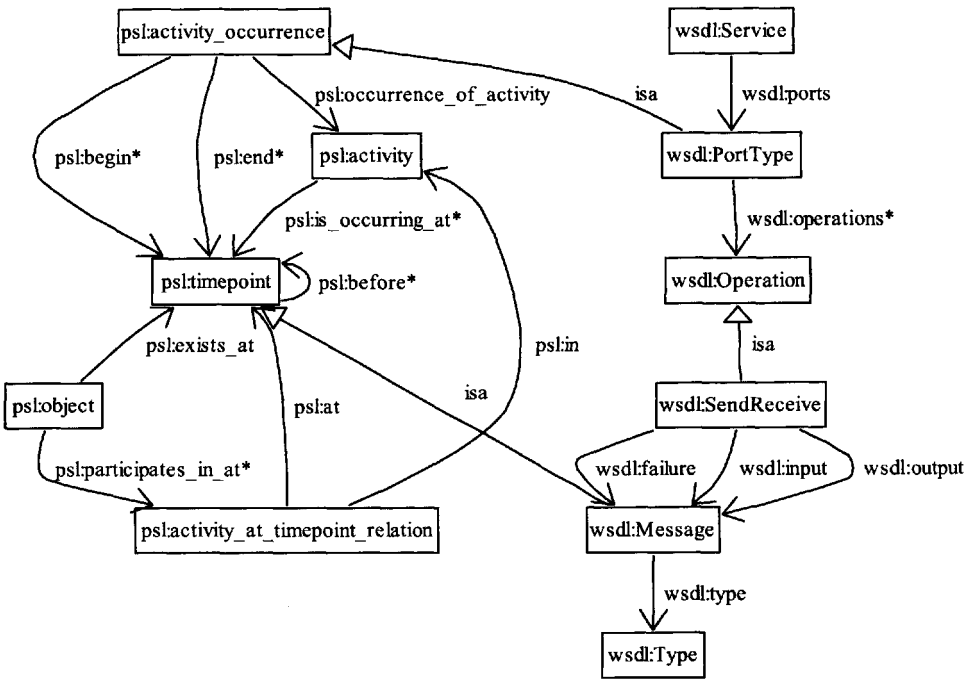


Figure 2: WSDL extension. The figure shows how the classes of our WSDL model are connected to the PSL temporal ontology.

portTypes, and activity occurrences correspond to activities running within a limited time interval that naturally fits WSDL operations. Accordingly, the begin and end points of an activity correspond to WSDL messages bounding operations and the messages are defined as subclasses of PSL timepoints. The objects referenced in the messages are thus defined as PSL objects.

WSDL activities (the input-output sequences) from a WSDL specification of a service are not related to each other. However, they could be easily annotated with, for example, the psl:before relation to form a complete chain of events and increase the number of tasks that can be solved with inference.

The most salient type of correspondence between different WSDL services is the 'request-response' correspondence that represents the fact that one (or more) events are a response to another one (or more) events.[12] These mappings may be incomplete when the messages attached to the request events are not sufficient to form the response and some additional requests need to be made to fill-in certain fields in the messages. These mappings have to be computed from the explicit 'request-response' mappings and the mappings associated with the messages. The mappings need to be validated against available temporal axioms and other mappings to ensure that they can be executed in real time.

Such a solution for shallow service annotation with temporal concepts is not sufficient to merit recommendation as a generic recipe. Most likely, future research on representing temporal knowledge on the Semantic Web will center on more elaborated initiatives like the Web Service Flow Language[13] (WSFL) built on top of WSDL, DAML-S[14] and other efforts aimed at representing workflow and domain semantics.

2.2 *Documents*

A service description in WSDL contains an attached XML Schema for each message to be produced or expected by the service. A well-defined XML Schema captures most part-of relations between the domain objects and/or literals referred in the document and it can be used as a rich source of background knowledge for making conceptual models of these objects.

XML Schema and RDF Schema use similar encoding and representation means [8] and partially XML Schema can be interpreted in RDF Schema [9]. However these two represent conceptually different information and it seems to be impossible to come up with a non-ambiguous and consensual interpretation of XML Schema in terms of RDF Schema [10]. One illustrative example from Figure 3 shows how different modelling layers can be mixed up in XML Schemas that should be avoided in RDF Schemas.

In addition, different companies tend to aggregate their data according to different criteria even if the whole set of data strings remains the same. An obvious example includes a supplier grouping its orders according to manufacturers' names and a delivery company grouping them according to delivery addresses.

Domain object being represented or referenced within XML Schemas and can be converted to document conceptual models. In case of well-elaborated schemas for well-structured domains, these models represent a decent part of the original conceptual model that was used

[12] For a number of pragmatic reasons we restrict this relation to one-to-many or many-to-one only prohibiting many-to-many mappings.

[13] http://xml.coverpages.org/wsfl.html

[14] http://www.daml.org/services/

```
<Address>
  <PhysicalAddress>The Address</PhysicalAddress>
</Address>
<Address>
  <LegalAddress>The Address</LegalAddress>
</Address>
```

(a) Using nested XML tags

```
<Address type="Physical">The Address</Address>
<Address type="Legal">The Address</Address>
```

(b) Using XML attributes with fixed values

Figure 3: Different ways to represent is-a relationship used in different XML serializations for the same document. These two fragments of real-life documents represent the fact that an address can be either a physical address or a legal address. In (a) nested XML tags are used to explicitly represent subclasses of `Address`. XML tags correspond to classes and properties, i.e. to the schema layer, and tag values correspond to instance data strings. In (b) attribute values are used for the same purpose. Now XML tags correspond to meta-classes encoding the fact that a concept has an attached `type` name that is instantiated with the type name. The tags are then directly instantiated with instance data strings, somehow suppressing the schema layer.

to construct the schema by the developers of the service.

A rule-based approach for extracting conceptual models from DTDs [11] 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, object attributes, cardinality constraints, etc. XML Schemas are more expressive than DTDs and include all the modelling primitives of DTDs. As a result an arbitrary DTD can be automatically converted to an XML Schema without any information loss. Accordingly, the rules developed for DTDs in [11] can easily be applied to XML Schemas. We illustrate this with a small example in Figure 4.

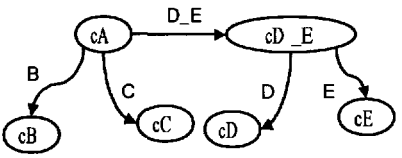
In practice, element and attribute names are frequently reused in the same schema and this causes a number of problems. For example when attribute `value` is assigned to tag `credit` it has different meaning from when it is assigned to tag `debit`. In this case two separate attributes `credit.value` and `debit.value` need to be created and interpreted in different ways.

We model XML element and attribute declarations as subclasses of the class `XMLThings` that contains two properties: `TagName` and `TagPath` identifying the place of the tag in the original XML Schema. `XMLElements` correspond to XML element declarations and `XMLAttributes` correspond to XML attributes and have an extra property `AttributeName`. `XMLThings` are associated with certain `wsdl:Messages` via the `Tags` property of the class `Documents` attached to the messages.

The most essential mapping tasks concerning the documents include normalizing the models by mapping each modelling primitive that occurs in an XML Schema to the mediating RDF Schema concept of the proper level, and resolving the problem of multiple data aggregations.

```
<xsd:element name="A">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="B"/>
      <xsd:element ref="C"/>
      <xsd:choice>
        <xsd:element ref="D"/>
        <xsd:element ref="E"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

(a) XML Schema



(b) Conceptual Model

Figure 4: An example of how XML Schema (a) can be converted to conceptual model (b). The `<xsd:choice>` of D or E is processed with rule `C(Choice)` from [11] to create 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. 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 from [11] concerning repeating elements, cardinality, and complicated composite elements.

2.3 *Vocabularies*

Quite often XML tags take their values not as unrestricted strings but as terms taken from fixed vocabularies. The numerous examples include country and currency codes, units of measurement and product categories in product encoding standards.

We model vocabularies as instances of class `Terms` presented in Figure 1 and containing the string property `Value` that defines its value as it appears in XML documents. Different terms can then be grouped into hierarchies corresponding to vocabularies. The XML tag or attribute values explicitly represented in XML Schemas are then encoded with classes `XMLAttributeValue` and `XMLElementValue` subclassed from `Terms` and thus with the inherited property `Value`.

The vocabulary mapping task consists of mapping terms to vocabularies and mapping equivalent terms.

3 Mapping Meta-Ontology

The RDFT (RDF Transformation) mapping meta-ontology[15] specifies a small language for mapping RDF Schemas with special extensions for mapping WSDL constructs and XML document models. We used the Common Warehouse Model (CWM) Specification [12] provided by the Object Management Group[16] that describes a general architecture for a mapping ontology to be adopted by each specific mapping application. The architecture contains three kinds of `TransformationMaps`: `ClassifierMaps` bridging classes, `FeatureMaps` bridging properties, and `ClassifierFeatureMaps` linking classes to properties (Figure 5). We use RDF Schema to represent RDFT classes and properties and Prolog to represent all the axioms.

[15] `http://www.cs.vu.nl/~borys/RDFT`
[16] `http://www.omg.org/`

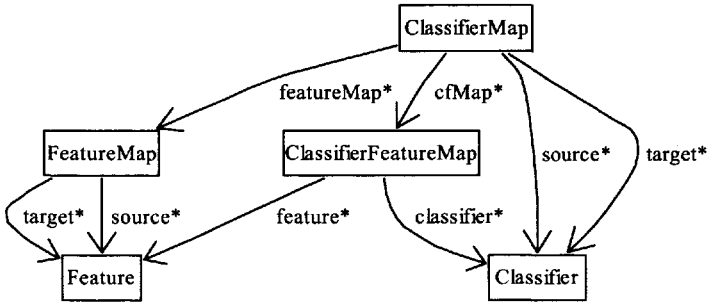


Figure 5: CWM Transformation Package.

We should mention the layering considerations that lead us to making RDFT a meta-ontology. XML instance documents naturally belong to the instance level to be represented in RDF. The same holds for the instances of services bounded to certain addresses and protocols. XML Schemas and service structures in WSDL are represented at the schema level as far as they describe the structure of the instances. Accordingly, the maps connecting the schema concepts are located at the schema level as well. The mapping language that specifies how we should construct the maps and the structure of the maps should belong to the third, higher, meta-level. From the RDF Schema point of view, the maps introduced in RDFT do not belong to the set of RDF Schema classes nor to RDF Schema properties because they are not a part of the domain. Instead, the maps constitute a third type of resources used to bridge the other RDF Schema resource (eventually, classes, properties, and literal constants).

RDF properties, called features in CWM, are defined as disjoint to resources, called classifiers in CWM, and thus cannot be mapped without serious discussion about the semantics of such a map. Hence, we adopt only two types of bridges: `ClassifierMap` and `FeatureMap`. In addition, we have to include special means for mapping service descriptions, XML and RDF Schemas into our RDFT ontology.

RDFT is a meta-ontology that means that a user is allowed to create mapping constructs by instantiating RDFT meta-classes. These constructs then become ordinary RDF Schema classes and properties. RDFT maps connect domain-specific classes and properties and correspond to the schema level. The RDFT ontology containing statements about these bridges that should naturally belong to the meta-level. The class diagram for the basic top-level meta-classes is presented in Figure 6 using the same notation as used in Figure 2.

3.1 Bridges

The basic RDFT class is `Bridge` that connects two concepts (similar to the CWM's `ClassifierMap`) and describes the common properties of bridges. As depicted in Figure 6 `Bridge` has two properties with multiple cardinality: `Source` and `Target` that point to the concepts being connected by the bridge. We only allow one-to-many and many-to-one bridges[17] by an extra axiom.

[17]As opposed to the CWM specification where many-to-many mappings are also allowed. We restrict our language for pragmatic purposes: allowing many-to-many bridges leads to serious difficulties in using them for transforming instance data.

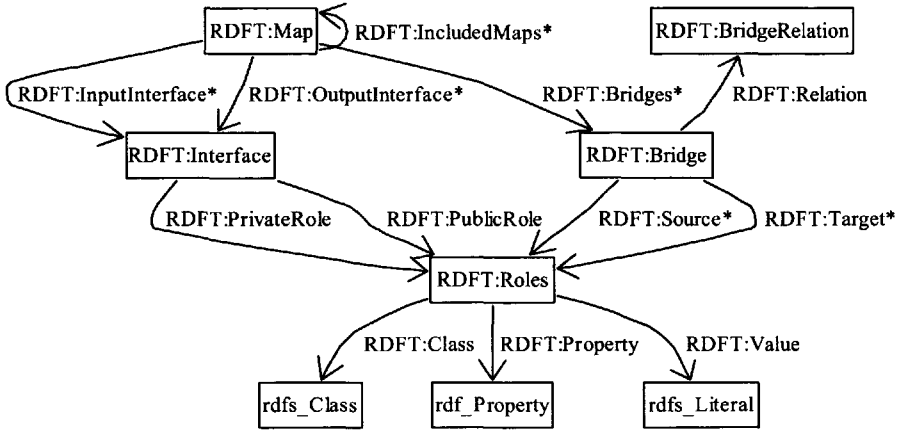


Figure 6: Main RDFT classes: Bridge, Map, Interface, BridgeRelation, and Roles.

Actual interpretation of each bridge is performed according to the `Relation` property attached to each `Bridge`. Each relation is defined as a subclass of abstract class `BridgeRelation`. We introduce a minimal set of relations for mapping different objects:

XML data elements are linked by the relation `Necessary` if an instance of the source concept(s) is necessary to generate an instance of the target concept(s), and `Sufficient` if an instance of the source concept(s) is sufficient to generate the target concept. The `Equivalent` relation is defined as both necessary and sufficient. A special axiom specifies that these three relations are applicable only to bridges linking the concepts instantiated from the `XMLThings` class and thus referring to XML Schema elements. This axiom is represented in Prolog as all other axioms we are using.

Events are linked by the relation `RequestResponse` that defines the target event of a bridge as a response to the source event. This relation is applicable to the bridges linking instances of `Events` that are in turn subclassed from `WSDLThings`.

RDF Schema elements and terms are linked with two generic relations `sameClassAs` and `samePropertyAs`. These two specify equivalent classes and properties, and are applicable only to the bridges linking `Terms`.

Specific applications of RDFT that deal with objects other than those covered by RDFT `BridgeRelations` may extend RDFT by creating private subclasses of `BridgeRelation` and defining specific axioms interpreting new relations. The recent wave of interest in ontology versioning [13, 14] may result in such an extension with the `version` relation specifying that the target set of elements forms an updated version of the source set of elements.

Sometimes inference-based bridge interpretation is not sufficient for instance data transformation. For this case we attach the property `Expression` to `Bridge`. `Expression`

specifies an XPath [15] 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`). These expressions are applicable only to bridges linking XML elements and affect only the concepts linked by a single bridge. The users are free to interpret them in any application-specific way.

3.2 Roles and Interfaces

Collections of bridges serving a single purpose are grouped together to form a `Map`. Each map serves three objectives: (i) aggregating bridges into logical units; (ii) interfacing between bridges inside a map and the concepts being linked with the map; and (iii) allowing the aggregation and reuse of different maps.

The concepts linked by bridges are either classes, properties or literal values. Each property assigned to several objects can have several meanings and should be mapped separately. The `Bridges` do not link them directly but via intermediate `Roles`. Each `Role` points to a `Class`, `Property`, or a `Literal` value with the properties named accordingly (see Figure 6 for a plot).

An `Interface` defines a mapping between different roles linked with the `PrivateRole` and `PublicRole` properties. These roles are necessary for bridge reuse and are incorporated in bridge `Maps`.

3.3 Maps

The bridges are grouped into `Maps` where each `Map` serves as a container for a collection of bridges serving a single purpose. These bridges are linked to a map with the `Bridges` property. The maps are identified by their names[18] and form minimal reusable modules of RDFT bridges.

Each `Map` may include other maps via the `IncludedMaps` property. Inclusion is interpreted as textual inclusion of the RDFT code of another map.

The bridges inside a map connect some local concepts, but they may be reused with other RDF concepts. Each `Map` may contain an `Interface` linking the roles used inside the map to some other roles outside the map, with which the map can be used. Conceptually, each interface is a one-to-one bridge stating that two roles are interchangeable.

4 Using RDFT

RDFT is a meta-ontology and all the properties of the classes defined in RDFT become resources properties of RDFT instances, while these instances themselves correspond to class definitions. This is illustrated in Figure 7 that depicts a user bridge `Bridge_001` linking three classes `S0`, `S1`, and `T0` via three roles.

[18] In this case we have a collision between resource identifiers (URIs) that are used in RDF Schema and represent a logical name without any associated physical object and resource locators (URLs) that point to files somewhere on the Web. The maps are identified by their URIs but should be accessible via their URLs as reusable files with bridges.

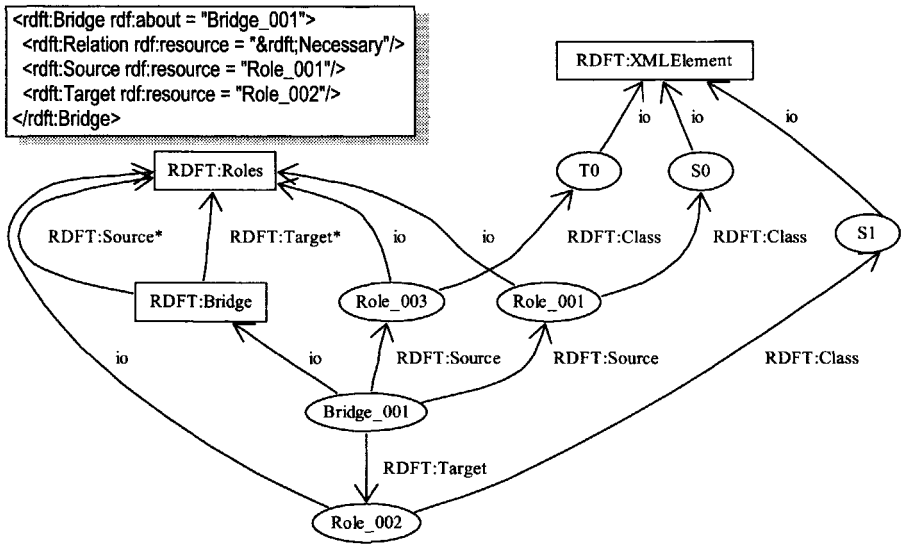


Figure 7: The use of RDFT as a meta-ontology: the definitions of the properties of the RDFT classes (`RDFT:Bridge` in the figure) are instantialized with ordinary properties of RDF resources, while the resource itself may correspond to a class definition, e.g. `Bridge_001` described in RDF in the code sample. Meta-concepts are represented by boxes and class-level concepts are depicted as circles.

4.1 Tool Support

We have developed a prototype tool supporting RDFT and implementing a number of ultimately useful features needed to develop and inference with RDFT maps. The tool is designed to support various service integration subtasks discussed in [7] and uses RDFT to represent all the mappings. In this section we are only focusing on the RDFT editing and inferencing functions provided by the tool.

A screenshot presented in Figure 9 gives an impression of the basic editing functionality provided by the tool. The tool allows a user to browse two WSDL service descriptions, examine their event sequences and XML Schemas of the messages. As mentioned in Subsection 2.1 WSDL event sequences are not related to each other and the tool provides an interface to annotate them with the `psl:before` relation. The tool allows the user to visually create and edit RDFT bridges.

The tool allows exporting RDF (Schema) ontologies in Prolog to enable Prolog inference with the bridges. Indeed, RDFT and its specific instantiations specified by the user are represented in RDF Schema encoded in RDF, and can be treated at the RDF level as simple collections of triples. We convert all these triples to Prolog: a triple `(object, property, value)` where `value` takes `rdfs:Literal` strings is represented with Prolog fact `l_triple(object,property,value)`, a triple with an `rdf:Resource` value is translated into fact `o_triple(object,property,value)`. RDF Schema theory (subclass-of, instance-of, domain and range relations) can then be easily represented using Prolog predicates.

```
show_double_linked_targets(Target, Bridge1, Bridge2) :-
  bridge_chain(Src1,Bridge1,Target),
  bridge_chain(Src2,Bridge2,Target).

show_unlinked_sources(Target, Bridge) :-
  resource_in_module(Target,'TRG'),
  not(bridge_chain(SrcCls,Bridge,Target)).

% Chaining source class - role - bridge - role - target class
bridge_chain(SourceClass,Bridge,TargetClass):-
  rdft_eq_bridge(Bridge),
  source_bridge_class(Bridge,SourceClass),
  resource_in_module(SourceClass,'SRC'),
  target_bridge_class(Bridge,TargetClass),
  resource_in_module(TargetClass,'TRG').
% Module support
resource_in_module(Resource,Module) :-
   o_triple(Resource,'mapper#InModule',Module).
% Searching for RDFT bridges
rdft_eq_bridge(Bridge) :-
  instanceOf(Bridge,'RDFT#Bridge').
% Analyzing bridges
source_bridge_class(Bridge, Cls) :-
  o_triple(Bridge,'RDFT#Source',Role),
  o_triple(Role,'RDFT#Class',Cls).
target_bridge_class(Bridge, Cls) :-
  o_triple(Bridge,'RDFT#Target',Role),
  o_triple(Role,'RDFT#Class',Cls).
```

Figure 8: Two validation tasks in Prolog: searching for double-linked target classes and searching for unlinked source classes. Two namespaces are used: RDFT `RDFT` and `module` introduced by the RDFT editor. All the models are first converted from RDF to Prolog: `l_triple` represents an RDF triple `object,property,value` where `value` is a literal, `o_triple` corresponds to a triple with resource value. Predicate `instanceOf` returns all RDF instances of a class and its subclasses.

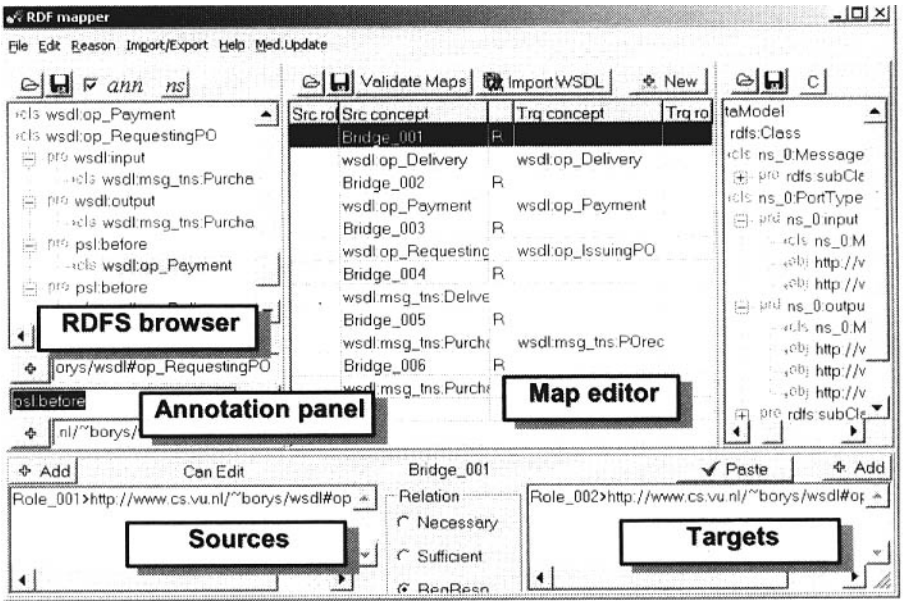


Figure 9: RDFT tool support: the editor.

An example of an inference task to be performed with RDFT bridges is presented in Figure 8 and includes two major predicates. The first `show_double_linked_targets` returns all the concepts of the target ontology that are linked to two or more concepts from the source ontology. The second `show_unlinked_sources` returns all the source concepts that are not linked to the target concepts. Numerous other inference tasks can easily be defined in a similar manner and be treated as a part of the RDFT meta-ontology.

5 Using OWL

The languages for knowledge representation on the Semantic Web are now undergoing rapid development and evolution resulting in the development of a new improved language for representing conceptual models on the Web called OWL. How grounded is the choice of RDF Schema as a central language for our mapping framework and what can we expect in the foreseeable future of the Semantic Web language development?

RDF (and its schema language RDF Schema) was primarily designed for the annotation of existing documents with conceptual models, capturing some *partial* knowledge primarily for information annotation and retrieval purposes. Accordingly, we face a number of problems when trying to representing *complete* conceptual models.

In RDF all the objects are described with object-property-value triples and quite often property names are reused. For example, the property `Address` may indicate billing address when it is attached to `Invoice` and delivery address when attached to `DeliveryOrder`.[19]

[19]The same problem occurs when an attribute is attached to different elements in XML Schemas. There we solve it by creating a separate class for each attribute attached to each element as described in Section 2.2.

In RDF Schema properties are defined as first-class objects with a global attachment to classes. Hence, it is not possible to specify that the `Address` of an `Invoice` differs from the `Address` of a `DeliveryOrder`. A possible solution to this problem lies in attaching the property `Address` to both classes and modifying its interpretation with expressions like: '`Address` that is attached to an instance of `Invoice`...' to separate these two cases.

In doing so, however, we run into a second problem. In RDF Schema each property definition uses the properties `rdfs:domain` and `rdfs:range` to specify the classes to which a certain property may be attached. Multiple occurrence of `rdfs:domain` and `rdfs:range` has conjunctive semantics, that means listing both `Invoice` and `DeliveryOrder` as ranges of `Address` should be interpreted as if `Address` is attached to an instance of *both* `Invoice` and `DeliveryOrder`, and not to an instance of *any* of them. To model our case correctly we need to create an artificial superclass for the two classes and assign this superclass as the domain restriction for `Address`.

OWL Lite allows the explicit representation of local range restrictions with the `allValuesFrom` and `someValuesFrom` property characteristics. In addition the 'heavy' version, OWL Full, would allow local domain restrictions with the `hasValue` constraint. In our current status we are using equivalent constraints introduced in the Protégé[20] environment.

The RDFT bridge `Roles` can be expressed in a more elegant way in OWL by using local restrictions rather than privately interpreted properties `Class`, `Property`, and `Value`. One-to-one equivalent `Bridges` themselves can be partially represented in OWL with OWL's statements `sameClassAs` and `samePropertyAs`. However, RDFT bridges may represent different relations between the source and the target sets of concepts and can be re-written in OWL only in a single specific case of one-to-one equivalent relation, and, for example, are not applicable to stating that a target event is a response to the source event.

Cardinality constraints on properties are missing in RDF Schema even though they are quite useful. They will appear in OWL and we are currently using their equivalents defined in Protégé. Finally, some of the temporal axioms can be re-written using OWL constraints specifying inverse, transitive, and symmetric properties.

There is some evidence that 'lighter' representation languages and simpler modelling constructs tend to be accepted faster than more expressive constructs [16] and we are trying to avoid overloading RDFT with complicated representation means or extensions. Most of the constructs provided by OWL in addition to RDF Schema and needed in RDFT are either supported by Protégé in present or they are easy to implement in our interpretations of RDFT, so we do not expect the upgrade to OWL to be a painful procedure.

6 Conclusions

In this chapter we have presented the RDFT mapping meta-ontology specifically designed for mapping the concepts that occur in the service integration scenarios: events, activities, XML document elements an vocabulary terms. RDFT maps are represented in RDF Schema and are easily converted into Prolog facts suitable for inference. The maps consist of bridges, each of which represents a certain mapping relation. The relations, in turn, are grouped into three categories according to the concepts being mapped: events, document elements, or vocabulary terms, with the correspondent interpretations.

[20] `http://protege.stanford.edu/`

The set of mapping relations corresponds to the most salient mapping problems that we have foreseen to date in the area of procurement services. However, other domains may require different sets of relations. From another side, new relations may increase the expressivity of the RDFT language that would create numerous problems in their deployment.

We are now in the process of conducting a series of case studies to provide some estimate of the adequacy of the set of mapping relations that we have introduced in RDFT and to discover possible (domain-specific) extensions to this set. A balanced version of RDFT containing a minimal set of relations still sufficient for certain domains is still under way. Additional and updated information on RDFT can always be found on the RDFT homepage.[21]

Acknowledgements. The author would like to thank Frank van Harmelen, Dieter Fensel, Christoph Bussler, and Michel Klein for their helpful discussions and comments.

References

[1] Brickley, D., Guha, R.: Resource Description Framework (RDF) Schema Specification 1.0. Technical report, W3C Candidate Recommendation (2000)

[2] van der Aalst, W.: Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow. Information Systems **24** (2001) 639–671

[3] 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), IEEE CS Press (2002)

[4] Bae, H., Kim, Y.: A Document-Process Association Model for Workflow Management. Computers in Industry **47** (2002) 139–154

[5] Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - A MApping FRAmework for Distributed Ontologies. In Gomez-Perez, A., Benjamins, R., eds.: Proceedings of the 13-th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002). Number 2473 in LNCS, Siguenza, Spain, Springer-Verlag (2002) 235–250

[6] Madhavan, J., Bernstein, P., Domingos, P., Halevy, A.: Representing and Reasoning about Mappings between Domain Models. In: Proceedings of the 18-th American Conference on Artificial Intelligence (AAAI-2002), Edmonton, Canada, AAAI Press (2002) 80–86

[7] Omelayenko, B.: Ontology-Mediated Business Integration. In: Proceedings of the 13-th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002). Number xxxx in LNCS, Springer-Verlag (2002)

[8] Gil, Y., Ratnakar, V.: A Comparison of (Semantic) Markup Languages. In: Proceedings of the 15-th International FLAIRS Conference, Pensacola Beach, Florida, AAAI Press, Menlo Park, CA (2002) 413–418

[9] Klein, M.: Interpreting XML via an RDF Schema. In: Proceedings of the Workshop on Semantic Authoring, Annotation and Markup at the 15-th European Conference on Artificial Intelligence ECAI-2002, Lyon, France (2002)

[10] Klein, M., Fensel, D., van Harmelen, F., Horroks, I.: The Relation between Ontologies and XML Schemas. Linkoping Electronic Articles in Computer and Information Science **6** (2001)

[11] 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, Springer (2001) 133–148

[12] CWM: Common Warehouse Model Specification. Technical report, Object Management Group (2001)

[21] `http://www.cs.vu.nl/~borys/RDFT`

[13] Nejdl, W., Siberski, W., Simon, B., Tane, J.: Towards a Modification Exchange Language for Distributed RDF Repositories. In Horroks, I., Hendler, J., eds.: Proceedings of the First International Semantic Web Conference (ISWC-2002). Number 2342 in LNCS, Sardinia, Italy, Springer-Verlag (2002) 236–249

[14] Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology Versioning and Change Detection on the Web. In Gomez-Perez, A., Benjamins, R., eds.: Proceedings of the 13-th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002). Number 2473 in LNCS, Singuenza, Spain, Springer-Verlag (2002) 197–212

[15] Clark, J.: XSL Transformations (XSLT). Technical report, W3C Recommendation (1999)

[16] van Harmelen, F.: The Complexity of the Web Ontology Language. IEEE Intelligent Systems **17** (2002) 71–73

Knowledge Transformation for the Semantic Web
B. Omelayenko and M. Klein (Eds.)
IOS Press, 2003

Transforming UML Domain Descriptions into Configuration Knowledge Bases

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

[1] *Institut fr Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik*
Universitt Klagenfurt, Universittsstr. 65-67,
9020 Klagenfurt, Austria
{felfernig,friedrich,jannach,zanker}@ifit.uni-klu.ac.at

[2] *Advanced Computing Research Center*
University of South Australia
5095 Mawson Lakes (Adelaide), SA, Australia
mst@cs.unisa.edu.au

Abstract. In this chapter we emphasize how we can integrate Web-based sales systems for highly complex customizable products and services by making use of the descriptive representation formalisms of the Semantic Web. Building on the equivalence between the consistency based definition and the description logic based definition of configuration [1], we are capable of transforming our application domain-independent meta-model for modeling configuration knowledge [2] into the emerging standards of the Semantic Web initiative, such as DAML+OIL. Furthermore, we discuss how these standardized description languages can be used to derive capability descriptions for semantic configuration Web services.

1 Introduction

The easy access to vast information resources offered by the World Wide Web (WWW) opens new perspectives for conducting business. One of these is the goal of the research project CAWICOMS to enable configuration systems to deal simultaneously with configurators of multiple suppliers over the Web [3]. The technical approach taken resembles state-of-the-art electronic marketplaces. Many-to-many relationships between customers and different suppliers are enabled, thus replacing inflexible one-to-one relations dating to the pre-internet era of EDI (electronic data interchange). We resolved the problem of heterogeneity of product and catalogue descriptions by imposing a common representation formalism for product models on all market participants based on UML. The textual representation of the graphical product models uses XML Schema definitions. XML serves as a flexible data format definition language that allows to communicate tree structures with a linear syntax. However, single standards for conducting B2B electronic commerce will not exist. As content transformation between those catalog and document standards is far from being a trivial task [4], the Semantic Web offers the perspective of dynamic knowledge transformations by reasoning on semantics.

In this chapter we will outline how configuration systems can flexibly cooperate in an ontology-based approach through the use of the Web service paradigm. The capability of each configuration system can be expressed by the evolving language standards of the Semantic Web initiative [5, 6]. In [1] it is shown that a consistency based and a description logic based configuration knowledge representations are equivalent under certain restrictions. Therefore, product model representations in our UML-based meta-model for configuration knowledge representation can be transformed into OIL resp. DAML+OIL [7]. We showed the transformation of the configuration meta-model into a configuration knowledge base using predicate logic already in [2].

In Section 2 we will give an example configuration model and in Section 3 a description logic based definition of configuration is given. Examples for the translation rules into a corresponding OIL (Section 4) representation and a discussion on semantic configuration Web services (Section 5) finally follow.

2 Configuration Knowledge Base in UML

The Unified Modeling Language (UML) [8] is the result of an integration of object-oriented approaches of [9, 10, 11] 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. [12] 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 [13], resource-based [14], and structure-based [15] 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 [2, 16]:

- **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).

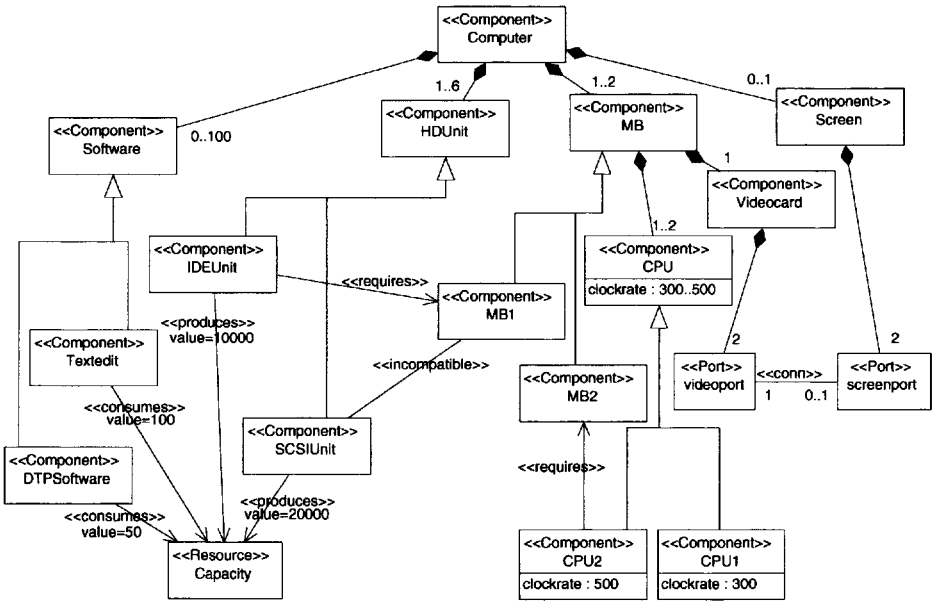


Figure 1: Example configuration model

- **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 [1] serves as a foundation for the formulation of rules for translating UML configuration models into a corresponding OIL representation[1]. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [17], where $\mathcal{RN}$ is a disjunctive union of roles and features.

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 the next section, 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 [18], 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 $CONF$={$\langle CPU1, \{c1, c2\} \rangle$, $\langle CPU2, \{c3\} \rangle$, $\langle MB1, \{m1\} \rangle$, $\langle MB2, \{m2\} \rangle$, $\langle$*mb-of-cpu*, {$\langle m1, c1 \rangle$, $\langle m1, c2 \rangle$, $\langle m2, c2 \rangle$}$\rangle$, ...}.

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

[1] In the following we assume that the reader is familiar with the concepts of OIL. See [7] for an introductory text.

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.

The equivalence between the definitions of valid configurations in description logic (Definition 2) and the definition in first-order predicate logic [19] is shown in [1]. It refers to the translation function $\mathcal{T}\langle\cdot\rangle$ in [17] that translates concepts, roles, terms, and axioms of a description language ($\mathcal{DL}$) without transitive closure into equivalent formulas in the first order predicate logic $\ddot{\mathcal{L}}^3_{CNT}$. Note that $\ddot{\mathcal{L}}^3_{CNT}$ allows only monadic and dyadic predicates and restricts the number of counting quantifiers and subformulas to at most three free variables. Building on this equivalence we are therefore capable of transforming our UML based configuration models [2] into the Semantic Web standards, such as DAML+OIL, that are founded on description logics.

4 Translation of UML Configuration Models into OIL

As already pointed out in the previous section the translation rules give exact semantics to the UML modeling concepts. In [2] translation rules that transform the graphical representation into a predicate logic configuration knowledge base have been already given. In the following, $GREP$ denotes the *graphical representation* of the UML configuration model. For the model elements of $GREP$ (i.e., component types, generalization hierarchies, part-whole relationships, requirement constraints, incompatibility constraints) we propose rules for translating those concepts into a description logic based representation. The definition is based on a schema S=($\mathcal{CN}$, $\mathcal{RN}$, $\mathcal{IN}$) of disjoint sets of names for concepts, roles, and individuals [17], where $\mathcal{RN}$ is a disjunctive union of roles and features.

Rule 1 (Component types): Let c be a component type in $GREP$, then

- DD_{DL} is extended by class-def c.

For all attributes a of c in $GREP$, and d the domain of a in $GREP$,

- DD_{DL} is extended by

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

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_{DL} is extended by

  $d_1, ..., d_n$: 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 [20], [16], [21] it is pointed out that part-whole relationships have quite variable semantics depending on the 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. Note that we do not require acyclicity since particular domains such as software configuration allow cycles on the type level. In the following we discuss two facets of part-whole relationships which are widely used for configuration knowledge representation [16] 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[2]. 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 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 $Partof_{mode} \in \{partof_{composite}, partof_{shared}\}$. A part p can be either a shared part (concept $part_{shared}$) or a composite part (concept $part_{composite}$). Given a part-whole relationship between p and w in $GREP$, then

- DD_{DL} is extended by

  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: subclass-of ($part_{shared}$ or $part_{composite}$).
  p: slot-constraint *w-of-p* min-cardinality lb_w w.

[2]Note that in our *Computer* configuration example we only use composite part-whole relationships - as mentioned in [16], 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.

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.

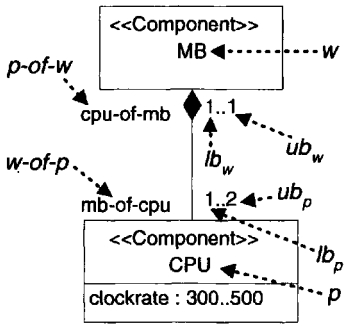


Figure 2: Part-whole relationship (p:part, w: whole)

Remark 1: The following properties have to hold for shared and composite part-whole relationships.

- Each shared part is connected to at least one whole, i.e.,

  (DD_{DL}) $part_{shared}$: slot-constraint $partof_{shared}$ min-cardinality 1 top.

- Each composite part is connected to exactly one whole, i.e.,

  (DD_{DL}) $part_{composite}$: slot-constraint $partof_{composite}$ min-cardinality 1 top.
  slot-constraint $partof_{composite}$ max-cardinality 1 top.

- A shared part cannot be a composite part at the same time, i.e.,

  (DD_{DL}) *disjoint* $part_{shared}$ $part_{composite}$

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: subclass-of ($part_{shared}$ or $part_{composite}$)
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[3] for these components - the components are within the same mereological context [16]. 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)...).

For the translation into DD_{LOG} the macro $navpath(c_1, c_n)$ is defined as

$\exists Y_1, Y_2, ..., Y_{n-1}, Y_n :$
$name_{c1}(Y_1, X) \wedge name_{c2}(Y_2, Y_1) \wedge ... \wedge name_{cn-1}(Y_n, Y_{n-1}) \wedge c_n(Y_n),$

where X is a free variable quantified outside the scope of this expression and represents an instance of concept $c1$. □

Example 4 ($navpath(Computer, CPU1)$)**:**
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[4] 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 $Videocard$. Note that the component type $Computer$ is not a nearest

[3] In Figure 1 the component type *Computer* is the unique common root of *IDEUnit* and *CPU1*.
[4] In this context $partof$ is assumed to be transitive.

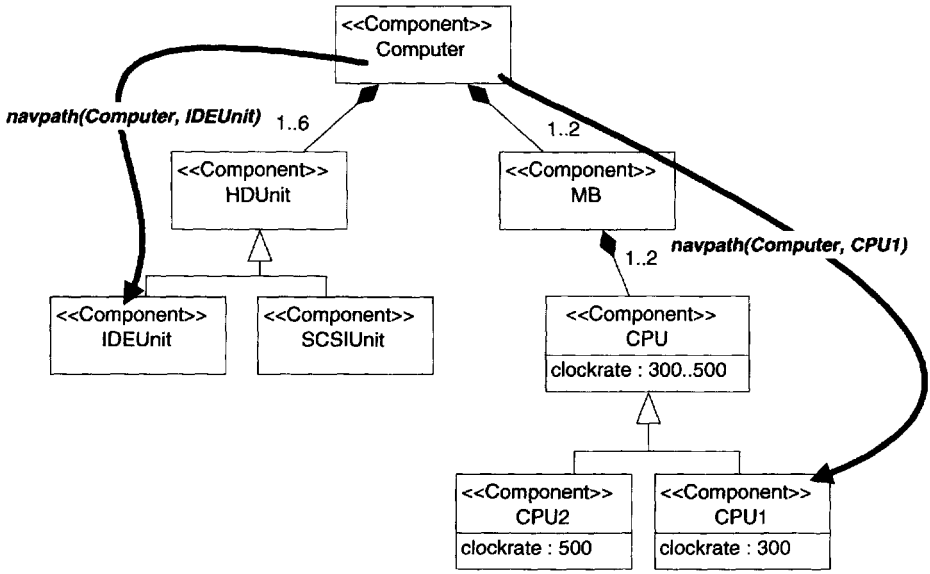


Figure 3: Navigation paths from *Computer* to $CPU1$ and $IDEUnit$

common root of CPU and $Videocard$ but the nearest common root of $CPU1$ and $IDEUnit$ (see Figure 3).

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 an instance of component type c_2 in 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 nearest common root of c_1 and c_2, then

- DD_{DL} is extended by r: ((not($navpath(r, c_1)$)) or ($navpath(r, c_2)$)). □

The condition part of the inner implication is a path from the nearest common root to the component c_1; the consequent is a 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 $MB1$)). □

Incompatibility constraints. An *incompatibility* 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 (Incompatibility constraints): Given an *incompatibility* constraint between a set of component types c=$\{c_1, c_2, ..., c_n\}$ in $GREP$ with r as nearest common root of $\{c_1, c_2, ..., c_n\}$, then

- DD_{DL} is extended by

  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. In order to introduce resource constraints, additional expressivity requirements must be fulfilled - this issue will be discussed in [1].

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 relationships.

In order to represent port connections in OIL, we introduce them via a separate concept *Port*[5]. 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 6 (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 $\{a, b\}$[6], 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_{DL} is extended by

  class-def pa subclass-of *Port*.
  class-def pb subclass-of *Port*.
  pa: slot-constraint *portname* cardinality 1 (one-of pa_1 ... pa_{ma}).[7]
  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.

[5]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.

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

[7]In this context pa_i denotes one pa port.

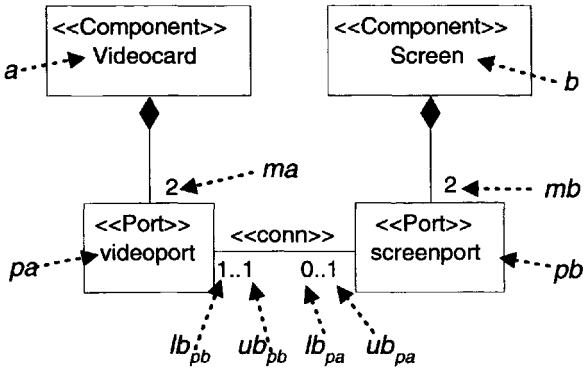


Figure 4: Ports in the configuration model

Example 7 (*Videocard* **connected to** *Screen*)**:**
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 port connection structure defined above, the constraint "a *Videocard* must be connected via $videoport_1$ with a *Screen* via $screenport_1$" can be written as follows.

Example 8:
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 graphical modeling concepts presented in this paper has its limits when building complete configuration knowledge bases. Typically, they do not only include the product structure itself, but also more complex constraints that cannot be represented graphically. Happily, (with some minor restrictions discussed in [1]) we are able to represent these constraints using languages such as OIL or DAML+OIL. UML itself has an integrated constraint language (Object Constraint Language - OCL [22]) 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 [23].

5 Semantic Configuration of Web Services

When it comes to multi-site product configuration, problem solving capabilities are distributed over several business entities that need to cooperate on a customer request for joint service provision. This Peer-to-Peer (P2P) interaction approach among a dynamic set of participants without a clear assignment of *client* and *server* roles asks for applying the paradigm of *Web services* [24]. It stands for encapsulated application logic that is open to accept requests from any peer over the Web.

Basically, a Web Service can be defined as an interface that describes a collection of provided operations. Consequently, we can interpret the application logic that configures a product (i.e. a configurator) as a standardized Web service. In the CAWICOMS approach [3] service requests and their replies are enabled by a *WebConnector* component that owns an object model layer that accesses the internal constraint representation of the configuration engine. This object model layer represents the advertised configuration service descriptions. A service requestor agent can, therefore, impose its service request via an *edit-query* onto the object-model layer and retrieves the configuration service result via a *publish-query*. In our workbench implementation this matchmaking task is, therefore, performed as part of the search process for a configuration solution of a constraint-based configurator engine. For the internal representation of the advertised service models as well as the service requests, an object-oriented framework for constraint variables is employed [25]. Reasoning on service requirements as well as on service decomposition is performed by the underlying Java-based constraint solver.

The offered Web services are employed by interface agents (i.e. the *Frontend* in terms of the CAWICOMS architecture) that interact with human users via a Web interface as well as by configuration agents that outsource configuration services as part of their problem solving capabilities. Formally, when implementing a Web service the following issues need to be addressed [24]:

- *Service publishing* - the provider of a service publishes the description of the service to a service registry which in our case are configuration agents with mediating capabilities. Within this registry the basic properties of the offered configuration service have to be defined in such a way that automated identification of this service is possible.

- *Service identification* - the requestor of a service imposes a set of requirements which serve as the basis for identifying a suitable service. In our case, we have to identify those suppliers that are capable of supplying goods or services that match the specific customer requirements.

- *Service execution* - once a suitable service has been identified, the requirements need to be communicated to the service agent that can be correctly interpreted and executed. UDDI, WSDL, and SOAP are the evolving technological standards that allow the invocation of remote application logic based on XML syntax.

Now, following the vision behind the Semantic Web effort [5, 6], the sharing of semantics is crucial to enable the WWW for applications. In order to have agents automatically searching, selecting and executing remote services, representation standards are needed that allow the annotation of meaning of a Web service which can then be interpreted by agents with the help of ontologies.

In the following, we sketch a Web service scenario that focuses on enabling automated procurement processes for customisable items (see Figure 5). Basically there are two different types of agents, those that only offer configuration services (L) and those that act as suppliers as well as requestors for these services (I). The denotation of agent types derives from viewing the informational supply chain of product configuration as a tree[8] where a configuration system constitutes either an *inner node* (I) or a *leaf node* (L).

Agents of type I have therefore the mediating functionality incorporated that allows the offering agents to advertise their configuration services. Matchmaking for service identification is performed by the mediating capability that is internal to each configurator at an inner node. It is done on the semantic level that is eased by multi-layered ontological commitments (as discussed in the preceding subsection) among participants. It is assumed that suppliers share application domain ontologies that allow them to describe the capabilities of their offered products and services on the semantic level. An approach that abstracts from syntactical specifics and proposes a reasoning on the semantic level also exists for transforming standardized catalog representations in [4].

In the configuration domain an abstract service description can be interpreted as a kind of standardized functional description of the product.[9] Furthermore, agents in the role of customers (service requestors) can impose requirements on a desired product; these requirements can be matched against the functional product description provided by the suppliers (service providers). If one or more supplier descriptions match with the imposed requirements, the corresponding configuration service providers can be contacted in order to finally check the feasibility of the requirements and generate a customized product/service solution.

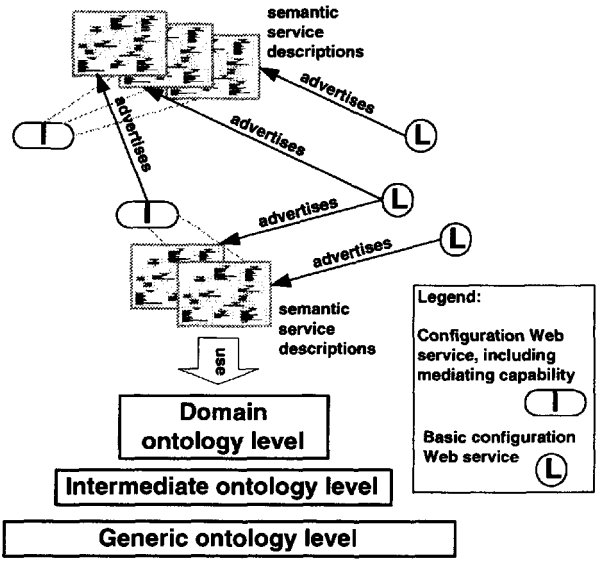


Figure 5: Web service scenario

[8]Note, that only the required configuration services are organized in a tree structure, which must not hold for the involved companies in the value chain of a product.

[9]In [13] this kind of description is denoted as a functional architecture of the configured product.

Consequently, it is necessary to have semantic descriptions of the demanded services that allow us to implement efficient matchmaking between supply and demand. Within these semantic annotations, restrictions on the domain and cardinality of slots, constraints on connections and structure, as well as the possible types of classes are possible. Furthermore, offered component instances can be represented as subconcepts (e.g. read from a catalog) of the classes of the service domain-specific configuration ontology. Additional supplier-specific constraints can be introduced.

Therefore, markup languages are required that enable a standardized representation of service profiles for advertisement of services as well as definitions of the process model. In this way, the task of identifying appropriate services and the decomposition of a service request into several separate requests can be performed by domain independent mediators. Due to the lack of these standards, this mediating functionality is in the current state of the CAWICOMS Workbench performed by application logic integrated into the configuration systems. DAML-S[10] is an example for an effort underway that aims at providing such a standardized semantic markup for Web services that builds on top of DAML+OIL. In this way semantic capability descriptions are possible that will then allow to implement semantic configuration Web services.

6 Summary

On the one hand, a standardized representation language is needed in order to tackle the challenges imposed by heterogeneous representation formalisms of state-of-the-art configuration environments (e.g. description logic or predicate logic based configurators), on the other hand, it is important to integrate the development and maintenance of configuration systems into industrial software development processes. We show how to support both goals by demonstrating the applicability of the Unified Modeling Language (UML) for configuration knowledge acquisition and by providing a set of rules for transforming UML models into configuration knowledge bases specified by languages such as OIL or DAML+OIL which represent the foundation for potential future description standards for Web services.

In [1] we build on the equivalence of a consistency based and a description logic based definition of configuration. Therefore, we are capable of transforming our application domain-independent meta-model for modeling configuration knowledge [2] into the emerging standards of the Semantic Web initiative, such as DAML+OIL, that set the stage for semantic capability descriptions that will be exploited by the next generation of Web services.

References

[1] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., Zanker, M.: Configuration Knowledge Representations for Semantic Web Applications. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM) (to appear) (2003)

[2] 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

[3] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Meyer, M., Petrone, G., Schfer, R., Zanker, M.: A Framework for Rapid Development of Advanced Web-based Configurator Applications.

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

In: Proceedings of the 15^{th} European Conference on Artificial Intelligence - Prestigious Applications of Intelligent Systems (PAIS 2002), Lyon, France (2002)

[4] Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M., Flett, A.: Product Data Integration in B2B E-Commerce. IEEE Intelligent Systems **16** (2001) 54–59

[5] Berners-Lee, T.: Weaving the Web. Harper Business (2000)

[6] Hendler, J.: Agents and the Semantic Web. IEEE Intelligent Systems **16** (2001) 30–37

[7] 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

[8] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)

[9] Booch, G.: Object-Oriented Analysis and Design with Applications. Addison-Wesley Object Technology Series (1994)

[10] Jacobson, I., Christerson, M., vergaard, G.: Object-oriented Software Engineering - A Use-Case Driven Approach. Addison- Wesley (1992)

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall International Editions, New Jersey, USA (1991)

[12] Chandrasekaran, B., Josephson, J., Benjamins, R.: What Are Ontologies, and Why do we Need Them? IEEE Intelligent Systems **14** (1999) 20–26

[13] Mittal, S., Frayman, F.: Towards a Generic Model of Configuration Tasks. In: Proceedings 11^{th} International Joint Conference on Artificial Intelligence (IJCAI), Detroit, MI (1989) 1395–1401

[14] Heinrich, M., Jngst, E.: 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). (1991) 257–264

[15] Stumptner, M.: An overview of knowledge-based configuration. AI Communications **10(2)** (June, 1997)

[16] Soininen, T., Tiihonen, J., Mnnist, T., Sulonen, R.: Towards a General Ontology of Configuration. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM), Special Issue on Configuration Design **12** (1998) 357–372

[17] Borgida, A.: On the relative expressive power of description logics and predicate calculus. Artificial Intelligence **82** (1996) 353–367

[18] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M., Zanker, M.: A Joint Foundation for Configuration in the Semantic Web. 15^{th} European Conference on Artificial - Configuration Workshop (2002)

[19] Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-Based Diagnosis of Configuration Knowledge Bases. In: Proceedings of the 14^{th} European Conference on Artificial Intelligence (ECAI), Berlin, Germany (2000) 146–150

[20] Artale, A., Franconi, E., Guarino, N., Pazzi, L.: Part-Whole Relations in Object-Centered Systems: An Overview. Data & Knowledge Engineering **20** (1996) 347–383

[21] Sattler, U.: Description Logics for the Representation of Aggregated Objects. In: Proceedings of the 14^{th} European Conference on Artificial Intelligence (ECAI). (2000) 239–243

[22] Warmer, J., Kleppe, A.: The Object Constraint Language - Precise Modeling with UML. Addison Wesley Object Technology Series (1999)

[23] 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, Chicago, IL (2000) 284–293

[24] McIlraith, S., Son, T., Zeng, H.: Mobilizing the Semantic Web with DAML-Enabled Web Services. In: 17^{th} International Joint Conference on Artificial Intelligence (IJCAI) - Workshop on E-Business and the Intelligent Web. (2001) 29–39

[25] Junker, U.: QuickXPlain: Conflict Detection for Arbitrary Constraint Propagation Algorithms. In: 17^{th} International Joint Conference on Artificial Intelligence (IJCAI) - Workshop on Modelling and Solving problems with constraint, Seattle, WA (2001)

Author index