FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Legacy Datasource Interoperability for the Semantic Web

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Bernhard Schreder
Matrikelnummer 9425714

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Florian Rosenberg

Wien, 15.07.2013

(Unterschrift Verfasser)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Legacy Datasource Interoperability for the Semantic Web

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Bernhard Schreder
Registration Number 9425714

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dr. Schahram Dustdar
Assistance: Dr. Florian Rosenberg

Vienna, 15.07.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Bernhard Schreder
Bergsteiggasse 12-16/2/13, 1170 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First of all, I would like to express my sincere gratitude to Prof. Dr. Schahram Dustdar, who had given me the opportunity to conduct this thesis at the Distributed Systems Group.

Many thanks also to Dr. Florian Rosenberg, my advisor, who had always taken the time to give me support and advice. Moreover, he provided me with very useful hints to improve my work.

Bernhard and Georg who had taken the time to proofread this work. Thank you both.

Finally, I would like to thank everyone who contributed to this thesis and especially my family for their great support during the years of study.

Abstract

This master thesis discusses the problem of integrating legacy data sources with other connected applications and data sources, one of the key issues of processes such as Enterprise Application Integration. This work examines one possible solution to this problem by leveraging semantic technologies, specifically, by mapping between ontologies which describe the legacy data sources. This solution necessitates the mediation and alignment between a data source's or application's data format and the format prescribed by the central, shared ontology. Utilizing ontologies is assumed to provide a flexible and light-weight solution to the data integration problem.

In this thesis several semantic mediation approaches are surveyed, including the requisite preprocessing steps such as schema lifting and normalization. Using the Mapping Framework of the University of Karlsruhe as the underlying architecture, a comprehensive mediation tool has been developed, which supports the creation and execution of mappings between different ontologies. Several new methods of importing legacy data sources, such as relational databases and XML data, are developed and introduced as new services for the mediation tool. Finally, a number of case studies and application scenarios for these mediation services are detailed and their applicability and flexibility are evaluated.

Kurzfassung

Diese Diplomarbeit behandelt das Problem der Integration von existierenden Datenquellen mit anderen Applikationen und Systemen innerhalb eines Unternehmens. Die Integration heterogener Systeme und Datenquellen stellt eine der zentralen Herausforderungen in Bereichen wie beispielsweise der Enterprise Application Integration dar. Diese Arbeit untersucht einen möglichen Lösungsweg dieses Problems, unter Verwendung semantischer Technologien. Abbildungen zwischen Ontologien, welche die verschiedenen Datenquellen beschreiben, werden genutzt um die Integration der unterschiedlichen Systeme zu ermöglichen. Die Mediation zwischen Ontologien stellt einen flexiblen und leicht wartbaren Weg dar, um unterschiedliche Datenquellen zu verbinden.

In dieser Arbeit werden zunächst verschiedene Mediationstechniken, welche auf semantischen Technologien basieren, untersucht. Unter Berücksichtigung verschiedener Anforderungen werden Entwurf und Umsetzung spezieller Mechanismen für die Integration heterogener Datenquellen geschildert, inklusive der dazu notwendigen Verfahren zur Extraktion und Normalisierung der jeweiligen ursprünglichen Datenmodelle. Eine Implementierung dieser Methoden als eigenständige Transformationsdienste wurde durchgeführt und als Teil eines Mediationstools realisiert, welches auf dem Mapping Frameworks MAFRA der Universität Karlsruhe basiert. Relationale Datenbanken und XML Dokumente sind hierbei die zentral unterstützten Datenformate. Verschiedene Fallstudien und Anwendungsbeispiele für diese Transformationsdienste werden geschildert, und die Anwendbarkeit sowie die Flexibilität dieser Dienste werden evaluiert.

Contents

List of Figures

List of Tables

Listings

CHAPTER 1

Introduction

1.1 Motivation

Semantic technologies are currently entering a new phase in their adoption for distributed computing environments, as they are applicable to a wide range of different problems, including the combination with classic Web technologies in the form of the Semantic Web to make Web content machine-interpretable. The Semantic Web is mainly regarded as the enabler for the next generation Internet, as it presents an evolution of machine-processable Web Sites by expanding their structure with semantic descriptions. While the vision of a "public" Semantic Web, as proclaimed by the authors of the seminal article in the "Scientific American" [5], is still a long-term goal, currently a more realistic approach is to use the technologies for the "corporate" Semantic Web. The new paradigm applied to existing knowledge representation and Web technologies also provides innovative solutions to integration problems in the B2B sector, especially in the fields of Enterprise Application Integration (EAI) and Business Process Management (BPM). Semantic Web technologies will provide diverse tools and frameworks in the coming years, which are going to ease the problems involved in the dynamic integration of information systems in and between businesses (referred to as dynamic B2B integration). Especially Semantic Web Services, as discussed for example in [14], are seen as a means to achieve a new level of flexibility and interoperability in enterprise middleware based on the Service Oriented Architecture (SOA) paradigm [30].

Following these tendencies in the B2B area, the creation and adoption of domain specific ontologies are necessary to describe data and process concepts of one's business. Seen as an evolution of information system architectures, rather than a revolution, B2B integration accentuates the need to develop software tools, which enable the reuse and integration of existing legacy systems and data sources.

In order to allow the reuse of a company's data sources, typical EAI solutions include data connector/adapter components which are used to transform from one application data format into another that is usable by the target application that needs to be integrated. These connectors are quite often hard to manage, not very flexible in their usage and often require an expert's intervention for maintenance. Instead of having to build countless one-to-one connectors between a company's applications and data sources, committing to a baseline data format understandable by all participants provides a more flexible approach to this data integration problem. Ontologies, defined for example by Gruber as providing a shared conceptualization of a domain of discourse [21], play a central role for this purpose.

By transforming their data in compliance with the shared ontology, applications and data sources can more easily interoperate with other systems which commit to the same ontology. This solution necessitates the mediation and alignment between a data source's or application's data format and the format prescribed by the central, shared ontology. Business application developers perform these design-time mediation tasks using dedicated modeling tools, while execution services, deployed on middleware infrastructure such as an Enterprise Service Bus (ESB), are used to act upon the predefined mapping rules at run-time.

1.2 Problem Definition

Mediation in general refers to techniques that aim at overcoming obstacles in the communication of two entities. In the scope of this work, mediation will be used to enable communication and interoperability between different legacy applications or heterogeneous data sources.

Mediation has been a well researched problem for some time, with different strategies proposed for different target applications. In [31] the authors propose a two-layered approach to data mediation. This two-layered mediation forms the basis for the work conducted in this thesis.

The main focus of this thesis is a detailed survey of currently available mediation frameworks leveraging semantic technologies. Additional tools and services are evaluated, allowing the mediation of other heterogeneous data sources, from XML documents to relational databases. Building upon this survey, several new services for the lifting of legacy data sources were developed in the scope of this thesis, while existing services were adapted to better fit real-world use cases.

An integrated mediation solution called SEML (Semantic Mediation of Legacy Datasources) is presented, as work from this thesis has been taken up by this project and applied in several case studies. Results from these case studies can provide valuable insight regarding the applicability and envisioned flexibility of data mediation via ontologies.

Ontologies can be expressed in a large variety of different representation languages, of differing expressivity and created for different application scenarios. In this work we will mainly focus on the Resource Description Framework (RDF) [29] to describe instances of an ontology represented in RDF Schema (RDFS) [13]. RDFS provides the baseline ontology representation elements which were necessary to support in the targeted ontology mediation solution. Many aspects of legacy data models, such as relational databases and XML documents, could be mapped to corresponding RDFS elements. Still, more powerful ontology representation languages are needed to express a larger set of legacy data models, and the inclusion of these languages will be a focus for future work. Specifically OWL DL is a candidate for replacing RDFS as the representation language in the lifting component.

1.3 Organization of this thesis

The thesis started out with a discussion of the problem domain and motivation in this section, continuing with a definition of the terminology used and a detailed survey of semantic mediation and the diverse strategies for data source lifting in Chapter 2. In Chapter 3, the theoretical approach and design for a semantic lifting component is explained in detail. Chapter 4 then explicates the implementation of this lifting component and the various choices made during it's development. This prototype for the lifting component has been used in several case studies, which are described in Chapter 5, together with an evaluation on flexibility, maintainability and performance of the described mediation solution. Related work, similar in scope to the lifting component developed for this thesis, is described in Chapter 6. Finally we conclude with a summary of the work done and an outlook on future work.

CHAPTER 2

State of the Art Review

This chapter contains a detailed survey of the current state of the art in (semantic) mediation frameworks and technologies, including prototypical implementations of lifting techniques for XML data and relational databases. Leading up to this survey are a section on the terminology used in the remainder of this work, as well as a section which contains a collection of functional requirements. These requirements were defined in the scope of the SEML project and were used for the evaluation of suitable mediation technologies to be chosen for the implementation of the project's prototype development. The requirements are also useful for the design and development of the new functionalities conducted as part of this thesis, which are introduced in Chapters 3 and 4, respectively.

2.1 Terminology

In order to clearly define the process of semantic mediation and its inherent complexity and capabilities, a definition of the commonly used terms in this area follows. A major part of this terminology definition section has been taken or adapted from [17], unless otherwise noted.

Ontology: There are diverse definitions for ontologies, but the most commonly used definition from an Artificial Intelligence perspective is the one found in [21]: "An ontology is an explicit specification of a conceptualization", where "a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose". Often this definition is extended by mentioning that the specification is a formal one, and the conceptualization is shared and thus understandable by different domain experts. Ontologies represent one of the main building blocks for the Semantic Web, as envisioned by Tim Berners Lee.

Mediation: Mediation in general refers to the task of resolving heterogeneity problems between two or more entities that were not designed to interact with and understand each other. A mediation process applies techniques to overcome any obstacles, which prevent communication and interoperability between these entities. Entities could be applications, components, processes, ontologies or other data sources and formats. This is not a new problem, and has been a research focus, for example in federated databases and other data integration efforts. In federated databases there are local schemas describing both individual databases or database groups. In both cases, bilateral mappings are provided for each of the databases to each other database or a single global schema is defined to include all of the others, which each database having a mapping to the global one.

Ontology Mediation: "Ontology mediation is the process of reconciling differences between heterogeneous ontologies in order to achieve inter-operation between data sources annotated with and applications using these ontologies. This includes the discovery and specification of *ontology mappings*, as well as the use of these mappings for certain tasks, such as query rewriting and instance transformation. Furthermore, the merging of ontologies also falls under the term ontology mediation." [17] The involved ontologies are assumed to be expressed using the same formalism and the same representation language. Language transformation is considered to be an adaptation problem and is normally not regarded in the scope of the mediation.

Mediation Framework: A mediation framework is a set of services designed to enable the creation of mappings, either automated or in an interactive process with an ontology engineer. The framework also contains methods to execute the mapping rules, to either transform data from a given source format to a target format, or to query the mapped data sources by the application of suitable mapping rules to the query language in question (see query translation).

Ontology Mapping: "An *ontology mapping* M is a (declarative) specification of the semantic overlap between two ontologies O_S and O_T. This mapping can be one-way (injective) or two-way (bijective). In an injective mapping we specify how to express terms in O_T using terms from O_S in a way that is not easily invertible. A bijective mapping works both ways, i.e., a term in O_T is expressed using terms of O_S and the other way around." [17]

Lifting and Lowering: The process of lifting is the execution of mappings to solve the so-called problem of lift, where different, possibly heterogeneous, data sources are mapped into a common representational framework for ease of processing. After lifting a legacy data source, which could be anything from XML documents to relational databases, the data is represented in a way as prescribed by a central, common representational format. In the context of this work, lifting always produces either an ontology or instances of an ontology. The reverse process of lifting is called lowering, which results in the execution of mappings in the reverse direction. From a general perspective, the processes of lifting and lowering (from a concrete source format to a target format) are special cases of the instance transformation use case as defined below, where the source and target data models are specified in different representational languages (e.g., in XML Schema or RDFS).

Query Rewriting: A typical use case for semantic mediation. In a distributed system, different ontologies are used for the representation of the information of multiple data sources. An

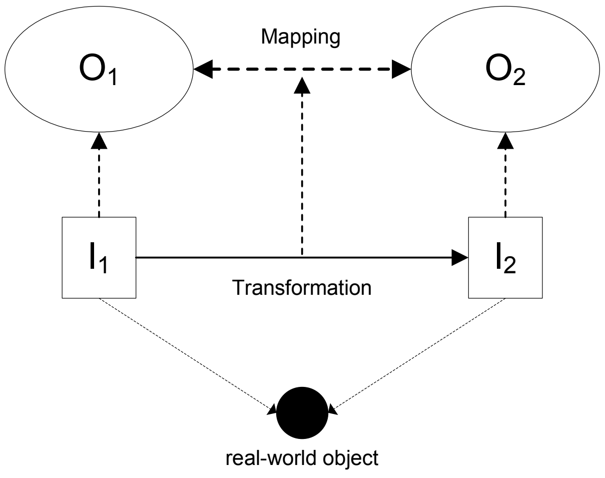


Figure 2.1: Instance Transformation [17]

application A normally uses one data source, whose information is represented in ontology O_A. When this application wants to execute a query on another data source, whose data is represented by ontology O_B, A at first formulates a query Q_A in terms of the known ontology O_A. As this query will be executed on the target data source, the process of query rewriting first needs to transform it as query Q_B, using the terms of ontology O_B. Afterwards, the query results can then be either directly returned to A or again transformed to conform to the representation format of O_A.

Instance Transformation: Instance transformation is another use case for semantic mediation. In this case two or more separate data sources are represented by several different ontologies. An application scenario might need the instances of the data sources merged into a single repository under one common data representation format. Instance transformation now refers to the task of transforming (a selection of) the instances of a source ontology O_S into instances of the target ontology O_T. Figure 2.1, adapted from [17], shows a single step of this transformation process: The source instance I_1, which is described in terms of ontology O_1, is transformed into another instance I_2, which conforms to the representation format set by target ontology O_2. Both instances refer to the same (real-world) object, albeit in different terms. The transformation step itself is guided by a mapping between the two ontologies, which needs to express the correlation between the instances of O_1 and O_2.

Ontology Learning: In [28], ontology learning is defined as the process of constructing ontologies facilitated by the integration of multiple disciplines, in particular the field of machine learning. The process is semi-automatic, requiring human intervention, and is seen as consisting of several distinct steps: the import and reuse of existing structures (e.g., ontologies, database schemas or document schemas); the ontology extraction phase, which utilizes learning technologies; the pruning phase in which the created ontology is adjusted to better suit its purpose; and finally the refinement phase, which produces an even finer grained ontology by using domain specific knowledge. These phases can be applied iteratively after validating the results against a measure provided by the target application.

2.2 Requirements Analysis

This section discusses several requirements, which were collected during the SEML project. These requirements are a mixture of both functional and non-functional constraints on the design and development of a comprehensive mediation toolsuite. As such, this collection of requirements influences the evaluation of current tools which takes up the remainder of this chapter, as well as the actual design for the lifting component, which is described in Chapter 3.

Requirements for a Mediation Framework

The following requirements were collected in order to influence the choice for a generic mediation framework to form the basis of the SEML Modeller. While the lifting component developed in this work is a separate service, these requirements are still important, since the choice of the framework itself constrains the way in which the lifting service can be connected to the overall mediation solution.

- R1.1 The mediation framework MUST support horizontal ontology mediation, that is mediation between two or more ontologies, which use the same representational language.

- R1.2 The framework MUST be able to work with ontologies based on RDF Schema. It SHOULD be possible to use different ontological representation languages, such as OWL or WSML.

- R1.3 The framework MUST provide methods to map at least concepts, attributes and relations. It SHOULD be possible to extend the mappings to axioms, functions and additional ontology language elements.

- R1.4 The actual mapping rules SHOULD be expressible in a dedicated mapping language (e.g., in a declarative mapping language, using a suitably expressive language).

- R1.5 Different transformations MUST be attachable to the mapping rules. These transformations are executed whenever a mapping rule is executed. Examples include data type specific transformations, such as string handling, or simple numerical functions.

- R1.6 The framework itself SHOULD be extensible, and should be highly modular, in order for new transformations, mapping rule constructs and import formats to be added to the system. The framework MUST enable the integration of its core services with a separate modeling environment, which will be designed in the scope of the project.

Requirements for the Lifting of Relational Data

The requirements for the lifting tool itself have been divided into the functionalities required for the support of the two main legacy data formats. Thus the requirements are mentioned first for relational data, then for XML data.

- R2.1 The lifting component MUST support the import of SQL based databases.
- R2.2 The database management systems supported SHOULD at least include the common open source solutions postgreSQL and mySQL, given the fact that these are the most prevalent systems in the envisioned use case scenarios.
- R2.3 The lifting component MUST be able to scan a provided relational schema (e.g., by opening a database connection and analyzing the database metadata).
- R2.4 The scanned database MUST be convertible to a corresponding RDF Schema.
- R2.5 The lifting process MUST convert at least relations, attributes, relationships (i.e., foreign key constraints) and data types for the attributes. The process SHOULD provide support for other database constraints (e.g., NOT NULL and UNIQUE constraints).

Requirements for the Lifting of XML Data

- R3.1 The lifting component MUST support the import of XML Schema documents.
- R3.2 It SHOULD be possible to add support for other schema languages (e.g., Schematron) to the lifting component in a straightforward manner.
- R3.3 The lifting component MUST be able to convert a provided XML Schema document to a corresponding RDF Schema document in an automated manner.
- R3.4 Configuring the transformation process (e.g., by selecting applicable rules) SHOULD be possible for the tool user.

2.3 State of the Art Mediation Frameworks and Tools

A number of mediation frameworks have been proposed in the last years, offering complete support for all mediation needs in the form of stand-alone tools or integrated with ontology management and engineering tool-suites. A detailed survey of diverse frameworks and tools can be found in [17], therefore, this section will concentrate on two important frameworks, one of which was chosen for the SEML Modeller, according to the presented evaluation. The selected framework imposes some additional requirements on the design of the lifting component - these requirements are explicated and linked to design decisions in Chapter 3. Finally future trends, such as ontology representation language independency, are mentioned and their relevance for continued development of an integrated mediation tool is detailed.

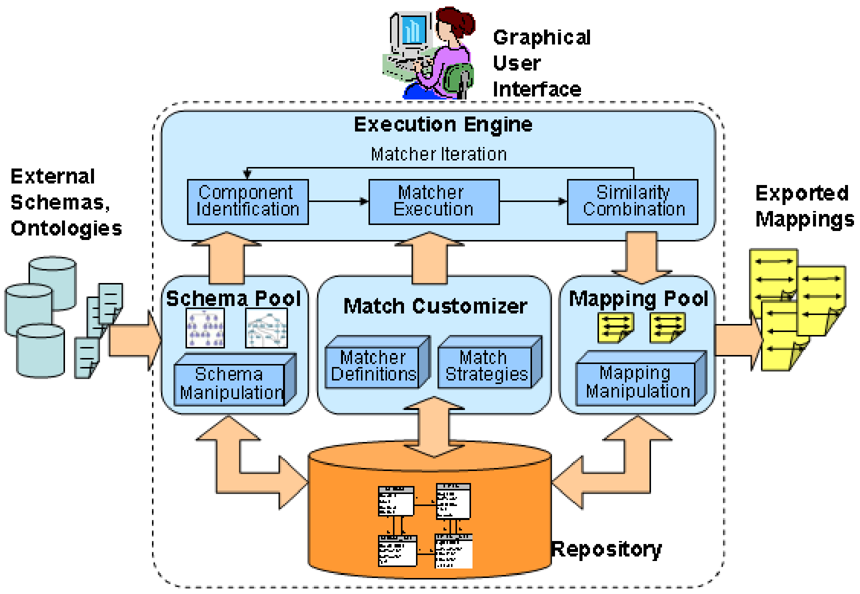


Figure 2.2: COMA++ Architecture (from [2])

COMA++

COMA++ [2] is a fully-fledged schema and ontology matching tool, which allows a user to mediate between various heterogeneous schemas. In the context of the COMA tool, “matching” is understood as the ability to derive suitable mappings between two different schemas. The tool itself is an extension of the previous version called COMA, and has been expanded significantly by providing a mechanism to combine many different schema matching algorithms. The theoretical basis for the tool’s implementation has been discussed in detail in [18], with a prototype and installation instructions available from the institute of computer science at the university of Leipzig[1].

Overview and Architecture of COMA++

The underlying architecture for COMA++ consists of five main components: the *Repository*, which stores all match-related data in a persistent manner, the *Schema Pool* and *Mapping Pool*, which manage schemas, ontologies and mappings in memory, the *Match Customizer* for the configuration of matchers and corresponding strategies and finally the *Execution Engine*, which actually performs the match operations. Figure 2.2[2] shows a comprehensive overview over the COMA++ architecture and its major components.

[1]available at `http://dbs.uni-leipzig.de/Research/coma_index.html`
[2]taken from `http://dbs.uni-leipzig.de/Research/coma.html`

Match Strategies

In addition to a variety of different matchers and match strategies to solve specific schema matching problems, COMA++ also addresses complex match problems by introducing *fragment-based matching* and *reuse-oriented matching*.

fragment-based matching: As reducing problem size has provable benefits to execution time and match quality, this higher-level match strategy identifies similar schema fragments, and then continues to apply a single match operation on each pair of similar fragments. The resulting set of mappings between fragments is finally merged to produce a global match result.

reuse-oriented matching: This global strategy includes several cases where previously determined match results can be reused. This can be achieved by either reusing direct mappings and by joining operations on so-called mapping paths (i.e., given mappings between schemas A and B, and B and C, new mappings from A to C can be derived).

Evaluation

Major features and benefits of the COMA++ approach include:

- A large variety of schema matching algorithms has been included in the system. These can be combined or applied separately to a specific mediation situation.
- A comprehensive Graphical User Interface, focussing on user-friendliness in order to enhance the practicability and effectiveness of the matching process.
- COMA++ uses a generic data model to represent the diverse schemas and ontologies. This allows to support a wide variety of different schema and ontology representation languages, including SQL, W3C XSD and OWL standards.
- COMA++ features a fragment-based match approach, which is a method to decompose large match problems into smaller ones.

Further development of COMA++ is ongoing, but, at the time of this writing, the current prototype is not suitable for integration with the lifting component described in Chapter 3. The main reason is that the architecture of the framework itself is not flexible enough for our purposes, which - according to requirement R1.6 - includes the integration of the mediation framework with a fully-fledged semantic mediation front-end of our own design.

MAFRA

During the design phase several key requirements of the mediation toolkits' components were defined (described in Section 2.2). Based on these requirements an evaluation of several available mediation approaches and their implementations was done, considering the basic objectives mentioned in the previous section. Several surveys (for a detailed look at diverse mediation strategies, frameworks and tools see [17]) showed the ontology mapping framework MAFRA

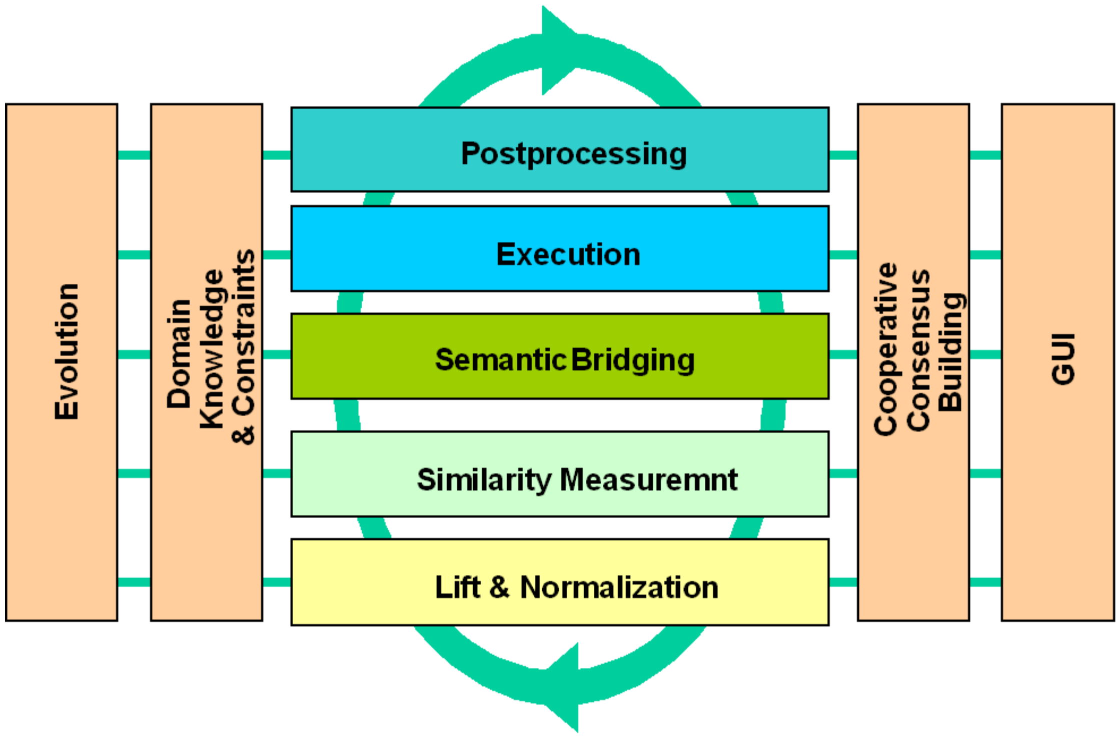


Figure 2.3: MAFRA conceptual framework [27]

to closely match the design requirements and objectives, while at the same time providing the necessary flexibility with its service oriented architecture for the integration of additional components and services. A general overview on MAFRA's architecture and its relevance to the work done in Chapters 3 and 4 follows.

MAFRA - the MApping FRAmework for Distributed Ontologies [27] has been developed by the FZI Karlsruhe[3]. The MAFRA approach and conceptual framework provides a generic view onto the overall distributed mapping process and supports all parts of the ontology mapping life-cycle. This life-cycle correlates to the phases for ontology learning, as explained in Section 2.1. The phases of the life-cycle are described in the next section. A toolkit implementing a specific architecture for the framework, called hMAFRA, is available and has already been used for the Harmonise project[4]. The MAFRA conceptual framework is outlined in the next section.

Conceptual Framework

According to [27], the conceptual framework of MAFRA, as shown in Figure 2.3, has been assembled from the set of observed commonalities of the different approaches for schema mediation in literature. It features five horizontal modules, which relate to distinct phases occurring

[3]Forschungszentrum Informatik at University of Karlsruhe, http://www.fzi.de

[4]http://www.harmo-ten.info/

during the mediation process. In addition a set of four vertical components support the whole mediation process during its entire lifecycle. The horizontal modules are briefly explained next, as these components are directly related to the core functionalities of the integrated mediation toolkit developed in SEML.

Lift & Normalization: The first step in the mediation lifecycle as defined by the MAFRA conceptual framework, this module's task is to cope with syntactical, structural and language heterogeneity. All data, both from the source and the target data source, needs to be mapped to a single representation. MAFRA assumes the data source's schema to be lifted to an RDFS ontology, and the corresponding data available as RDF instances. This phase of the mediation lifecycle is the central part of the work conducted in this thesis. The lifting component described in Chapter 3 corresponds directly to this mediation phase. The second part of this phase, the normalization, transforms ontology elements (e.g., by entity tokenization and acronym expansion), in order to prepare for the following similarity measurement phase. Normalization is not considered in this work, but Chapter 3 will point out where specific normalization tasks can be added to the lifting component.

Similarity: In order to support the discovery of applicable mapping rules between the source and target ontologies, this module uses a set of similarity measurement algorithms to establish similarities between the ontology elements. MAFRA proposes a sequence of different similarity measurement strategies, starting from the application of the WordNet service[5] to capture lexical similarity between the elements, to property, bottom-up and top-down similarity measurement algorithms.

Semantic Bridging: MAFRA assumes a semi-automated approach to actually create the mappings between different ontology elements, called semantic bridges. Results from the similarity measurements are reused to establish both concept and property bridges, which are then refined in an interactive manner with a tool user. MAFRA specifies its own Semantic Bridging Ontology (SBO), represented in the language DAML+OIL[6], which contains a set of ontology elements describing possible bridges and transformation details. Actual semantic bridges between a specific source and target ontology are then created as instances of this SBO. Further details on this semi-automatic approach can be found in [37].

Execution: This module is responsible for actually executing the mapping rules (the semantic bridges) defined in the last phase. MAFRA defines both an offline - called static or one-time transformation - execution, and an online execution - a dynamic, continuous mapping between source and target. This definition of execution can be related to the two main use cases for semantic mediation, as discussed in Section 2.1: instance transformation (the offline execution of mapping rules) and query rewriting (the corresponding online execution).

[5]available at `http://wordnet.princeton.edu/`

[6]a precursor of the Web Ontology Language (OWL)

Post-processing: Results from the previous phase are refined and different mechanisms for quality improvement are applied in this module. The authors of the MAFRA conceptual framework acknowledge that the establishment of object identity constitutes the most challenging task in this lifecycle phase. This task is equivalent to the notion of instance unification, as defined in Section 2.1.

Service-oriented Approach

MAFRA's service-oriented approach to ontology mediation is a major advantage for its use in an integrated mediation tool. In [36], the authors have observed that a static, centralized approach to the implementation of the conceptual framework defined above is virtually impossible to accomplish, given the large set of different transformation requirements inherent to the mediation tasks. The current implementation of MAFRA instead utilizes a decentralized, modular approach, taking up several characteristics of a Service Oriented Architecture, in order to achieve a lightweight and easily extensible solution. Independent transformation modules are attached to the system core modules, which provide the main functionalities, i.e., bridging, execution and evolution. These transformation modules are called *Services* in MAFRA, and each is described through a simple service ontology, allowing the core modules to request the features provided by those services.

Evaluation

MAFRA can be applied to diverse application scenarios, where data integration and reuse of heterogeneous data sources is deemed important. A typical example would be the integration of databases in a Customer Relation Management (CRM) tool, where data from the customer database is combined with data from other sources, like the order provisioning and fulfillment system. Besides allowing the transformation of datasets into new formats, new MAFRA services could support dynamic querying of source data without having to make changes to the original data sources and their models.

Due to its service-oriented approach and the support of all phases of the mediation lifecycle through its conceptual framework, MAFRA was found to fulfil the requirements needed for the basic underlying architecture of an integrated mediation toolkit, as noted in Section 2.2. Therefore MAFRA services have been used for the implementation of SEML, which is constraining the design of the lifting component conducted in this work as detailed in Chapter 3.

Besides the service-oriented approach, which allows for the flexible extension to the functionalities provided by the MAFRA, the second major reason to adopt this technology for the mediation toolkit was the existence of a freely available implementation of the conceptual framework. A first version of a MAFRA implementation has been available since 2003, and provided the semantic bridging and execution modules, together with a Graphical User Interface. While this MAFRA toolkit[7] supports only the mediation of RDFS ontologies, the hMAFRA toolkit (Harmonise Mapping Framework Suite[8]) adds an additional component for XML schema im-

[7]available at `http://mafra-toolkit.sourceforge.net/`

[8]available at `http://sourceforge.net/projects/hmafra`

port, lifting the schema to an RDF representation. This XML lifting component is examined in 2.5. hMAFRA has already been applied to several real world projects.

Trends and Current Development

Continuous development in the area of ontology mediation led to the inclusion of several new features in current tool development. Examples for these trends include:

- Ontology representation language independency: Currently a large set of ontology representation languages is available, from the W3C supported Web Ontology Language to frame logic based languages such as F-Logic. While MAFRA assumes all participating ontologies to be represented in RDFS, this automatically excludes many higher-level language constructs from being mappable. Solutions such as the DERI Ontology Management Environment (DOME), currently under development by the Ontology Management Working Group [9], have been conceived with the goal of achieving ontology representation language independency. DOME features a language-neutral ontology API by specifying a language-neutral meta model for defining ontologies.

- Versioning support: Ontology development has been noted to be a *shared* process, as well as a process undergoing continuous evolution, implying that some form of versioning support has to be adopted. Specifically, problems include how to use instance of earlier versions of an ontology with later versions. As an example, the DOME tool utilizes an approach to versioning similar to code versioning systems such as CVS or Subversion.

In Chapter 3, we will examine which of these functionalities can be supported by the lifting component developed in this work or, more generally, by the complete mediation toolkit developed in SEML.

2.4 Relational Database Lifting

The need of accessing legacy data sources for the Semantic Web became evident in the last years, since it cannot be expected that systems and applications provide entirely new data based on ontological representation formalisms. Usually stored in relational database management systems (RDBMS), the data will be lifted to an ontological level as needed. Several differing approaches to make this data accessible for Semantic Web enabled applications have appeared in the recent years. The following section first mentions several challenges for the design and implementation of a relational lifting tool, then evaluates a few of the currently available tools and compares their viability in an integrated Legacy Data Source Mediation solution.

Main Challenges

A description of the relational model [15] and how it is (not) applied in modern day relational database management systems has been provided in literature (e.g., in [16]). For the purposes

[9] available at `http://www.omwg.org/`

of the challenges of lifting data stored in a RDBMS, the following summary, taken from [4], is presented:

- Relational data is organized in relations, or tables as the RDBMS world terms it, with each record or row of a relation consisting of the same set of attributes (or fields).

- RDBMS utilize primary and foreign keys to create joins between tables, that is, to express relationships between the entities described by the contents of two table rows.

It has been noted that the interpretation of a database schema from a semantic point of view – that is, deducing what is "meant" with certain schema constructs – is complicated by the database engineer's design decisions. Furthermore, in production environments databases are often denormalized to optimize them for performance. These factors add to the challenge of extracting the hidden semantics from a database schema and choosing the most suitable ontology constructs to represent these semantics. While a suitable solution would involve the consultation of the original Entity Relationship model used to define the database model, this is not often a possibility. Another approach is to implement the database lifting as an interactive approach, allowing a human user to decide how certain relations and database constrains are to be interpreted.

A simple, first approach to generate RDF data from a relational database – in order to be able to further process the data with RDF tools and applications – doesn't take the creation of a suitable ontology into account. Used by many database to RDF export tools, the mechanism involves the creation of RDF nodes for every row of every relation. After this first step, property arcs are attached to the generated nodes. For each column in the relation, a property is created with the corresponding field content as the property value.

While usable for the development of a triple-based API to relational databases, this approach is not suitable to an ontology lifting tool, since it doesn't expose the conceptual entities related to the database schema, but rather expresses an RDF description of the underlying database. Especially, since the relational model – as typically implemented in a RDBMS – is not necessarily a suitable means to express various concepts of the problem domain. Also taking the changes due to optimization efforts and inherent assumptions into account, a fully automated lifting process will usually yield dissatisfactory results. Therefore, additional user interaction is encouraged, or additional processing phases involving other mechanisms are introduced to generate a conceptual model with a higher relevance to the real world problem domain.

Still, the lifting component that has been developed in the course of this work uses an automated process, since the user interaction takes place at the ontology mapping level. The intent of the tool user would be to work on one level of abstraction, without being subjected to the specific characteristics of the underlying data models. Thus, different data schemas, from heterogeneous data sources, can be presented to the tool user in a homogeneous manner.

According to [4], the following features of a modern RDBMS present additional challenges when considering an automated database lifting component.

Complex Field Types

A wide variety of field types is used by the different database management systems available today. When creating a property in an ontology from a given column, often the column's field type can be directly mapped to a datatype, preexisting in the ontology representation language, most of which are related to XML Schema datatypes (e.g., the numeric types available in the PostgreSQL database management system include smallint, integer and bigint, all of which can be mapped to the xsd:integer datatype). Some field types might have no matching XSD datatype however, which often is solved by mapping to an untyped literal, thus discarding some information provided by the database system.

Related to this problem are compound types for fields as provided by some database management systems (e.g., an array of integers). An application of the RDF container or collection language elements might be possible, but this is usually discouraged in the ontology engineering field, due to the unclear semantics of these RDF language constructs (which has been noted in the RDF Semantics specification itself [22]).

Object-relational Databases

Some database systems have been introducing features, which take up an "object-relational" approach, such as the inheritance of properties from one relation to another. Class inheritance features might be suitably modeled in RDFS, even though the details of such features might vary between different providers' products.

The Use of NULL Values

While the NULL value did not directly appear in relational algebra originally, most SQL compliant database systems introduced the NULL value as necessary for an efficient operation. Regardless, the NULL value is used nowadays to express a variety of meanings, from "not appropriate", "not available", "don't know", to a means of expressing implicit information, such as "this person is an adult" (when appearing in a "date-of-birth" field). The implicit semantics for a particular NULL value are often only known by the database engineer or hidden away in business logic. As no automatic lifting rule can be applied to the many diverse uses of the NULL value, in many cases no statement is generated at all by the database schema lifting algorithm, when the object of such a statement would be NULL.

Current Approaches

In the following section a selection of currently available research projects and prototypes in the area of relational database schema lifting is presented and evaluated in the context of integrating them in a centralized mediation toolkit, as envisioned in Section 2.3. A common limitation of these approaches is the necessity of scanning the database schema in order to be able to apply the diverse mapping rules defined for the respective approach. Usually the scanning process is dependant on the abilities of the used database/JDBC driver, which has to provide information about the tables existing in the database and about which columns a table consists of. Furthermore information about the foreign and primary keys of the database's relation is necessary to deduce the applicable mappings. If the driver cannot provide this information, a mapping tool user usually has to provide this himself.

KAON Reverse

Reverse[10] is a prototype for mapping relational database content to ontologies enabling both storage of instance data in such databases and querying the database through the conceptualization of the database. Reverse is an early prototype, a refined version will be merged with the hMafra Tool, effectively combining the results of both the EU-IST Harmonise and WonderWeb projects.

Reverse defines two distinct kinds of mappings: *TableMappings* and *(Table)ColumnMappings*. *ColumnMappings* can only be defined in the context of an existing *TableMapping*.

TableMappings are further subdivided into *Table-To-Concept-Mappings*, which map each relation to an existing concept of the provided ontology, and *Detailed-Table-To-Concept-Mappings*, where each row of a table should generate its own concept. Tables can be mapped to multiple concepts, and concepts can be mapped to multiple tables.

ColumnMappings again come in two distinct types: The first type is the *Column-To-Attribute-Mapping*, which maps a concrete value represented by the column to an attribute of the concept referenced by the *TableMapping*. Foreign keys cannot be mapped in this way. The second kind is the *Column-To-Relation-Mapping*, which is created when a column representing a table's foreign key is mapped to a target relation of a concept.

Reverse cannot be easily integrated in a comprehensive Ontology Mapping and Mediation solution, as it depends on the existence of an ontology to define its mappings to database constructs. Thus the tool would have to be used stand-alone to generate mapping rules between the database schema and the ontology structure, as no possibility exists to automatically extract an ontology from the database schema.

D2RQ

D2RQ [11] is a declarative language to describe mappings between relational database schemas and OWL/RDFS ontologies. The mappings allow RDF applications to access the content of huge, non-RDF databases using Semantic Web query languages like SPARQL [33]. Currently the eigth version of the language specification is available[11].

With D2RQ one can query a non-RDF database using an RDF query language like RDQL or SPARQL. Furthermore it is possible to publish the contents of a non-RDF database on the Semantic Web using the Joseki RDF server[12]. Information can also be accessed in a non-RDF database by using the Jena model API[13]. In addition to querying the data as a virtual RDF graph, RDFS and OWL inferencing can be done over the content of a non-RDF database using the Jena

[10]The prototype of KAON Reverse is available at `http://kaon.semanticweb.org/alphaworld/reverse/`

[11]at `http://d2rq.org/d2rq-language`

[12]a SPARQL server for Jena, available at http://www.joseki.org/

[13]part of the Jena Semantic Web Framework, available at http://jena.sourceforge.net/

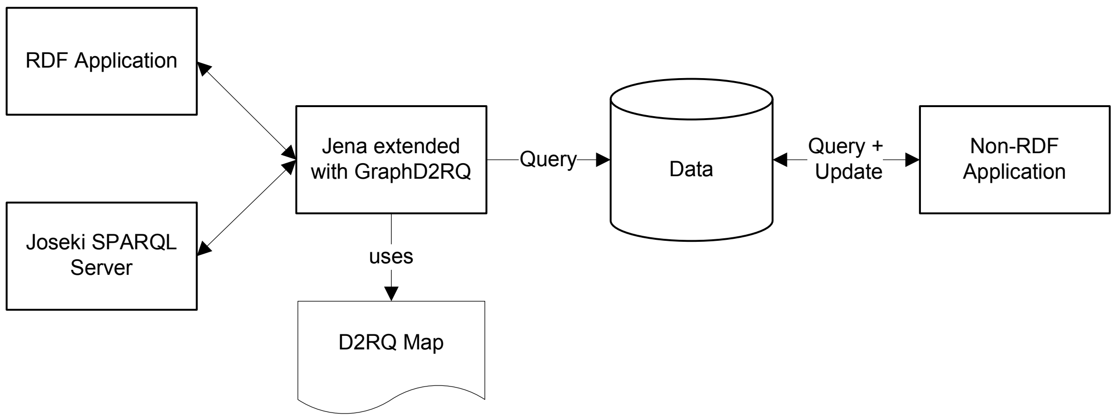


Figure 2.4: D2RQ application scenario (adapted from [11])

ontology API. Finally, since version 0.4 of D2RQ, it is also possible to access information in a non-RDF database by using the Sesame[14] repository API.

In the Jena Semantic Web framework the basic information representation object is a graph. D2RQ is implemented on top of such a Jena graph, wrapping a set of relational databases as a virtual, read-only RDF graph. Access to this graph is possible either by Jena API calls, such as the find(subject, predicate, object) method, or by rewriting queries from RDQL (and SPARQL since version 0.5) to corresponding SQL queries. Result sets generated by the execution of said SQL queries are then transformed into RDF statements, which are passed back to the Jena framework. Also included with D2RQ is the Sesame wrapper, which has been added to the framework in version 0.4, wrapping the D2RQ implementation for Jena behind the Sesame repository interface.

Important aspects of the D2RQ architecture and language specification, as currently defined in [8], are explained in detail in the following sections.

D2RQ consists of a declarative mapping language, which can be used to describe the relation between an ontology and a relational data model, and GraphD2RQ, which is a plug-in for the Jena Semantic Web framework. This plug-in uses the mappings described in the D2RQ language to rewrite Jena API calls to SQL queries and also to transform query results back to RDF triples. Returned triples are then passed on to the higher layers of either the Jena framework or the Sesame framework. Figure 2.4 shows the overall application scenario for D2RQ. A legacy database - which is still being queried and updated by another (non-RDF) application - is queried by Jena with GraphD2RQ, by utilizing a previously generated D2RQ mapping document. Queries are generated and the returned RDF triples are consumed by either a stand-alone RDF application or a specialized RDF server - such as the Joseki SPARQL server.

The D2RQ mapping language provides two central elements to map an ontology to a database schema: `d2rq:ClassMaps` and `d2rq:PropertyBridges`. Each Class Map represents a class or a group of similar classes of the ontology. Class Maps also indicate how instances of the mapped class are to be identified. Each Class Map has a set of Property Bridges - which can be

[14]open source RDF database, available at http://www.openrdf.org/

either plain literals or URIs and blank nodes that relate the resource to other resources. Property Bridges specify how the properties of an instance are created.

Shown below is an example fragment of a D2RQ map. The mappings included in this example relate the database table “Conferences” to the class “Conference” in an ontology.

```
map:Conference a d2rq:ClassMap;
    d2rq:dataStorage map:Database1.
    d2rq:class :Conference;
    d2rq:uriPattern "http://conferences.org/comp/confno@@Conferences.ConfID@@";
    .

map:eventTitle a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Conference;
    d2rq:property :eventTitle;
    d2rq:column "Conferences.Name";
    d2rq:datatype xsd:string;
    .

map:location a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Conference;
    d2rq:property :location;
    d2rq:column "Conferences.Location";
    d2rq:datatype xsd:string;
    .
```

Listing 2.1: D2RQ Mapping Sample

The most important D2RQ language constructs, as defined in version 0.8 of the language specification, are described in detail below.

d2rq:Database: The d2rq:Database construct defines a connection to a relational database, which has to support either access via JDBC or ODBC. Properties of this element further specify the type of the database columns which are to be used by D2RQ. Multiple d2rq:Database elements are allowed, to connect to separate databases from the same D2RQ map. A number of important properties for d2rq:Database are shown in Table 2.1.

d2rq:ClassMap: A Class Map in D2RQ defines how instances of a single class - or a group of classes - are identified, given an existing class in an RDFS or OWL ontology. For attributes and relations of the class, additional d2rq:PropertyBridges are used, as detailed below. Four different identity definition mechanisms are provided by D2RQ, some of which are explained together with other important properties of d2rq:ClassMap in Table 2.2.

d2rq:PropertyBridge: Property Bridges are used to create properties for instances generated by a Class Map. Both attributes (i.e., properties whose values are literals) and relations (i.e., properties whose values are either URIs or blank nodes, relating the resource to another resource in the ontology) are supported. Table 2.3 shows some of the more important properties which can be used for the definition of a d2rq:PropertyBridge.

d2rq:AdditionalProperty: This element can be used to add fixed statements to all instance created by a connected Class Map. Two properties are used for this bridge, the name of

Property name	Description
d2rq:jdbcDSN	JDBC database URL, in the form jdbc:*subprotocol*:*subname*
d2rq:jdbcDriver	JDBC driver class name (e.g., com.mysql.jdbc.Driver)
d2rq:username	The username with which to connect to the database
d2rq:password	The password to be used for the username
d2rq:textColumn d2rq:numericColumn d2rq:dateColumn	These properties can be used to declare the data type of database columns

Table 2.1: Important Properties of d2rq:Database

Property name	Description
d2rq:dataStorage	Reference to the d2rq:Database used for this Class Map
d2rq:class	An RDFS or OWL class, of which instances are generated
d2rq:uriPattern	Instances are identified according to this URI pattern
d2rq:uriColumn	Specified database column is used as an alternative identification mechanism
d2rq:translateWith	Values from d2rq:uriPattern (or d2rq:uriColumn) are translated by the assigned d2rq:TranslationTable before resources are generated
d2rq:additionalProperty	Specified property is attached to all instances of this class
d2rq:condition	A SQL WHERE condition constrains which database rows are used for resource generation

Table 2.2: Important Properties of d2rq:ClassMap

Property name	**Description**
d2rq:belongsToClassMap	Specifies to which Class Map this Property Bridge belongs to
d2rq:property	The RDF property used to connect the instance with the object or literal value created by the bridge
d2rq:column	Used for attributes, this specifies the column used for the literal value
d2rq:pattern	A combination of database values can be used for a literal attribute using this property
d2rq:datatype	Specifies the datatype of literals
d2rq:uriColumn	For relations, this specifies the database column containing URI values
d2rq:uriPattern	For relations, used in a similar manner as the corresponding Class Map property
d2rq:refersToClassMap	Foreign keys can be mapped with this property, which refers to another Class Map
d2rq:join	Used to create literal or object values from database tables' columns outside the Class Map's scope
d2rq:condition	SQL WHERE condition, constrains selected values (e.g., to suppress empty literal values)
d2rq:translateWith	Link to a d2rq:TranslationTable, to translate values from d2rq:column or pattern

Table 2.3: Important Properties of d2rq:PropertyBridge

the property to be added to all instances, and the value. d2rq:AdditionalProperty elements are useful for generic statements, such as rdfs:seeAlso properties.

d2rq:TranslationTable: A d2rq:TranslationTable is used for an additional transformation step between a database value and a corresponding RDF value. These translation have to be specified in one of three ways, either as a Java class implementing the Translator interface, as a CSV file containing translations, or as a collection of single translations with the d2rq:Translation element. Only translations which are unique in both directions can be used, in order to allow for bidirectional mappings.

An older version of the D2RQ mapping language, D2R Map [9], included the ability to directly add a SQL query to a ClassMap, selecting the data from the database which should be instantiated. D2RQ removed this feature, but the addition of conditions, which are basically SQL WHERE conditions, and the join attribute to PropertyBridges allows to achieve similar results. The website for D2R Map[15] features a downloadable processor for the language, as well as the complete language specification and schema.

The W3C Semantic Web Advanced Development for Europe (SWAD-Europe)

The SWAD-Europe project aims to support W3C's Semantic Web initiative in Europe, providing targeted research, demonstrations and outreach to ensure Semantic Web technologies move into the mainstream of networked computing. The project aims to support the development and deployment of W3C Semantic Web specifications through implementation, research and testing activities. The Project Fact Sheet[16] further states the following as the project rationale:

> Semantic Web Advanced Development for Europe (SWAD-Europe) aims to play a key role in the evolution of the Semantic Web, through education and outreach to developers, organizations and content creators; through Open Source implementation and testing, and through pre-consensus technology development to drive and inform the creation of new Semantic Web standards.

As part of this project, one of the project's workpackages (WP10: Tools for Semantic Web Scalability and Storage) produced a report on Mapping Semantic Web Data with RDBMSes [4]. Existing work on mapping triple stores and databases is surveyed and an overview of the major implementations and their schemas is given. Furthermore, this report specifically concerns itself with the mapping of RDBMS schemas to RDF, examining the difficulties in naming and identifying entities, choosing property names, dealing with ordering and sequences and the use of RDF datatyping, among other things. Most of these contents have already been mentioned in the previous section on the challenges for a relational database lifting tool. The discussion concludes with three observations, as quoted from the report in the following paragraph:

[15]available at http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rmap/D2Rmap.htm

[16]available at http://www.w3.org/2001/sw/Europe/factsheet/

> [. . .] This discussion has been principally from the point of view of exposing relational data in a format suitable for import and processing by RDF tools. However, similar considerations apply when wrapping a legacy RDBMS for use with an RDF triple-matching API.
>
> - At the meta level, RDFS is somewhat limited in its ability to capture constraints present in a relational schema.
> - Both the relational model and RDF(S) permit idiomatic approximations of a conceptual problem-domain model. These are close enough that the export of data automatically is possible; however, some knowledge of the problem domain is usually required for a more "natural" expression of the data in RDF.
> - Further, more formal, work in this area is needed.

The first point can be addressed by the choice of an alternative ontological representation language. The use of OWL or WSML would allow for the definition of constraints of an equivalent expressiveness as those used in a relational schema. In a similar vein, other concepts of these representation languages would allow one to address some other complications arising in the lifting of a relational schema to RDFS.

Regarding the second statement about the need for knowledge of the problem domain, the solution for the relational lifting problem discussed in this work does not aim at capturing all hidden semantics of the legacy relational schema. Rather the resulting, automatically created ontology can be used as an input to the following mediation phase with a centralized, shared ontology - as discussed previously in the context of an integrated ontology mediation tool. While this (semi-automatic) mapping creation phase would need the help of a knowledge engineer or domain expert, the preprocessing step of lifting the schema does not.

Finally, since the completion of this report, additional, formal work in this area has been conducted, such as the work on reverse engineering of databases discussed in the following section.

Reverse Engineering of Relational Databases to Ontologies

In a paper by the Estonian researcher Irina Astrova from the Tallinn University of Technology [1], the author considers exclusively ontologies as the target of reverse engineering tasks from relational databases. While other approaches often only analyze key correlations of relational schemas, the author proposes to also base the reverse engineering process on data and attribute correlations, and on combinations of these correlations.

The Reverse Engineering method described is separated in two distinct processes, schema transformation and data migration, of which only the schema transformation process is more closely examined herein. The process uses a relational database in third normal form (3NF) as its main input. Five steps are conducted to deduce a semantically equivalent ontology from the relational schema. These steps are shown in Table 2.4.

Each of these steps is explained in detail in the following sections.

Classification of relations. The Schema Transformation process classifies three different categories of relations. Base relations are independent of any other relation in the database schema,

Schema Transformation steps
1. classifying the schema's relations
2. mapping the relations
3. mapping the attributes
4. mapping the relationships
5. mapping the constraints

Table 2.4: The Schema Transformation Process

formally a relation $r \in R$ is a base relation, if $\neg\exists r_1 \in R$ such that $K_1 \subset K$, where $K = key(r)$ and $K_1 = key(r_1)$. Dependent relations are relations whose primary key depends on another relation's primary key. Formally, a relation $r \in R$ is a dependent relation, if $\exists r_1, r_2, \ldots, r_n \in R$ such that $K_1 \subseteq K_2 \ldots \subseteq K_n \subseteq K$, where $K = key(r)$, $K_i = key(r_i)$, $i \in 1 \ldots n$ and $n \geq 1$. All other relations, thus all relations which are neither base nor dependent, are classified as composite relations.

Mapping relations. In this step all relations are mapped to concepts, with each relation mapped to a concrete concept, except composite relations, which cannot be easily mapped. Composite Relations could denote the necessity to introduce an inheritance relationship if their attributes consist only of primary and foreign keys, but for the automatic transformation phase they can be mapped to either concepts or attributes, depending on their structure.

Mapping Attributes. Each attribute in a relation can be mapped to a property of the corresponding concept, except for foreign keys and primary/foreign keys, which represent relationships and are mapped in the next step.

Mapping Relationships. Relationships, which are represented by foreign keys and primary/-foreign keys in database schemas, can be either mapped to a concept, property or an inheritance relationship. The decision on how to map relationships depends on the types of key, data and attribute correlations. The analysis of key correlation is used for deriving semantics from a relational database (e.g., when key disjointedness holds on two relations with a binary relationship, this relationship can be mapped to properties). Data and attribute correlations can also hold important information on the semantics of a database schema. One of many possible examples would be the case of key equality, data equality and attribute disjointedness holding on a relationship. This denotes a kind of vertical partitioning in the database, where attributes of a single relation have been split into two relations, but can be recombined to a single concept in the resulting ontology. A set of other interpretations of correlations is examined in the paper.

Mapping Constraints. The final step in the Schema Transformation process concerns itself with specific constraints defined in SQL like PRIMARY KEY, UNIQUE, NOT NULL, etc. These can be mapped directly into an Ontology's axioms, thus resulting in logical expressions which preserve all the semantics embedded within a relational database.

2.5 XML Schema Lifting

One of the basic legacy data sources to be used in a mediation scenario is an XML document. In order to capture the conceptual model underlying XML data, an XML schema can be created. This XML Schema Defintion (XSD) is the basic data model used by a lifting tool to create an ontology. The XML data is then used during the execution of the lifting to produce corresponding ontology instances. The lifting component for the SEML Modeller should support this functionality, therefore several current approaches to the lifting of XML schema are surveyed in this section.

Challenges for an XML Schema Lifting Tool

Different approaches to the lifting of an XML schema usually concentrate on different challenges related to XML and XSD, or feature a specific design philosophy influencing its possible inclusion in a comprehensive mediation tool.

- The set of language constructs supported by the lifting mechanism. This defines the amount of the implicit conceptual knowledge which can be expressed explicitly in the created ontology. If the lifting mechanism doesn't support a specific XSD construct, some knowledge may get lost during an automated lifting process. Some language elements are inherently difficult to translate to a suitable ontology construct, therefore an interaction with a user to define an alternative mapping might be necessary by certain tools.
- How mapping rules are defined by the lifting mechanism, i.e., whether there is a declarative language which expresses the mapping rules, or whether mappings are stored internally in the tool. Also part of this issue, is the question if the representation language used for the mapping rules is human-readable.
- The supported target representational format is of further importance, since this also depends on the possible ontology to ontology mediation taking place in later lifecycle phases of an ontology mediation.

The State of the Art solutions presented below are evaluated according to these criteria.

Current Approaches

In the following section a selection of currently available research projects and prototypes in the area of XML Schema lifting is presented and evaluated in the context of integrating them in a comprehensive mediation tool, as envisioned in Section 2.3.

Normkit

The Normalization Toolkit (henceforth called Normkit), is a stand-alone application with which the lift problem, as defined in Section 2.1, can be solved. The tool consists of two distinct phases: the first phase is called C-Normalization, and lifts a supplied XML Schema document to a corresponding RDF Schema, resulting in a Normalization Map which contains the mappings between the two schemas. This Normalization Map is utilized in the following D-Normalization phase, where XML documents conforming to the XSD from the first phase can be transformed to RDF documents, which contain instances conforming to the RDF Schema developed in the previous phase.

Normkit is also a part of the hMAFRA tool-suite and directly integrated in the mapping framework of this tool-suite. As the normalization process can be completed fully automatically without any user interaction, it is transparent to the user, who only has to define the mappings on a horizontal layer (i.e., between two different ontologies).

The approach to the lifting process used by Normkit and corresponding design decisions have been detailed in [20] and are more closely examined in the following section.

Conceptual Normalization

The reverse engineering process of creating an RDFS Ontology from a legacy XML Schema is called Conceptual Normalization by the authors of Normkit. The process is seen as semi-automatic, as certain decisions by a user between competing mappings might be necessary. A set of *Normalization Heuristics* define the possible semantic interpretations of XML Schema language elements. In this context, the notion of a normalization heuristic is meant to signify a suitable, intuitive match between elements. These heuristics steer the normalization process, which results in the generation of a *Normalization Map* containing the explicit mapping rules between XSD and RDFS. The mapping rules themselves are instances of a separate mapping ontology - the Conceptual Normalization Ontology (CNO), containing node and path bridges.

Normalization Heuristics

An XML Schema can be represented in terms of an abstract data model, using a set of schema components as categorized by the W3C in [38]. Examples for these categories include *Type Defintion Components* or *Model Group Components*. Each such component in a legacy XML Schema is evaluated in its specific context and a suitable heuristic is applied to it. This will result in the creation of an ontology element, like a class, property or constraint.

For an example of the set of heuristics applicable to Type Definition Components, see Table 2.5.

Normalization Map

The Normalization Map contains a set of Normalization Bridges, relating elements of the created RDFS Ontology to the structures of the legacy XML data model. The following depicts a sample fragment from a Normalization map:

```
<a:NodeBridge rdf:ID="1043067566943-1925540250">
  <a:relatesXPathNode="/Address">
  <a:relatesRDFSNode rdf:resource="&n;#Address_class"/>
  <a:hasBridge rdf:resource="#1281265636"/>
</a:NodeBridge>

<a:PathBridge rdf:ID="1281265636">
  <a:relatesXPath="Street_and_Nr/text()">
  <a:relatesRDFSPath rdf:resource="#859349052"/>
</a:PathBridge>
```

	XSD component Precondition	Action Postcondition
ED1	Element declaration	create property
	typed	type heuristic determines property range
ED2	Element declaration	create property
	any content	property range is XML Literal
AD1	Attribute declaration	create property
		type heuristic determines property range
EP1	Element particle	add domain constraint
	local declaration or reference; ED1 applied	parent component heuristic determines property domain
AU1	Attribute use	add domain constraint
	local declaration or reference; AD1 applied	parent component heuristic determines property domain

Table 2.5: Element and Attribute Normalization Heuristics

```
<a:PropertyNode rdf:ID="">
  <a:relatesConcept rdf:resource="&n;Address_class"/>
  <a:relatesProperty rdf:resource="&n;Street_and_Nr"/>
  <a:relatesObject rdf:resource="&rdfs;Literal"/>
</a:PropertyNode>
```

Listing 2.2: Normalization Map Sample

This sample mapping links the Address data of an XML document to a corresponding RDFS Address class. XML data is modeled using a subset of the XQuery and XPath data model, interpreting the XML document as a tree consisting of several different types of nodes and links.

SWWS Lift API

Semantic Web Enabled Web Services (SWWS)[17] was a European IST project under the 5th Framework Programme of the European Commission, and has been running from 2002 to 2005. One of the goals of SWWS was to provide a scalable Web Service mediation middleware. As XML continues to be the primary data format for e-Commerce, it was planned to gain independence from any particular serialization by modeling the data with RDF. While this can be achieved by describing ad-hoc XSL transformations from XML into RDF/XML, the reverse transformation is problematic, because the transformation rules depend on the direction of the mapping. One of the deliverables of this project therefore describes the “lift API” to be used by the envisioned mediation middleware. (The concept of lifting was introduced in Section 2.1)

[17]Semantic Web Enabled Web Services, project website at http://swws.semanticweb.org

This API provides several methods which allow for a direct interpretation of an XML document with an RDF model. The API only supports the vertical mapping between the concrete XML documents and the instances prescribed by the domain ontology. Horizontal mappings (mappings between different ontologies) are not supported.

The Lifting process itself consists of a two-stage approach. The first stage of the XML to RDF lift produces an RDF model of the document itself. This model is isomorphic with the document structure, which means that every element and attribute of the source XML document corresponds to an RDF property. This so-called document model can then be transformed in a second stage to a form that is consistent with a given domain ontology, thus producing a model of the document content, rather than its structure. The main difference between these two models is that the document model makes use of the terminology of the source XML schema, whereas for the content model no such restrictions exist.

In the following paragraphs, several design approaches and decisions for the first stage of the lifting process are examined. The second stage is not thoroughly described, as it plays no part in the context of an integrated mediation toolkit. The horizontal mappings of MAFRA can be used to map the document model to any target schema or ontology; therefore it is not necessary to explicitly build a content model from the document model.

Simple and Complex Type Mappings

The first stage of the lifting process as implemented by the lift API is based on the central idea that every element and every attribute of the XML document map to an RDF property, viewing the XML structure as a relational model between parent nodes and their children. Generally this means that hidden semantics are not made explicit by the lifting process, which only aims at bringing the data into the RDF realm. The interpretation of intended meanings of the document structure (i.e., elements could rather represent classes, and only XML attributes could be therefore regarded as the classes' properties) is intentionally left out of this stage of the lifting process. One reason is that generally the intended meaning of the XML is not uniformly applied in all possible documents, and even when it is this cannot be deduced from the provided XML Schema. The second process stage can be applied to allow for subsequent inferencing using post-mapping rules in order to bring out the intended semantics.

The basic mappings are created by examining the XML Schema's type definitions. Simple data-types are mapped onto a relation between a resource and a typed literal. Untyped simple types are interpreted as plain literals. Special cases are derived simple types, for which only the base type is used for the property definition. Property restrictions can be used to preserve the content of enumerations.

In XML Schema, complex types are constructed by using the sequence, all or choice compositors. For each complex type, a new resource is generated by the lifting process, which represents the class used to define the rdf:type of corresponding instances.

Finally IDs and IDREFs are not preserved as their own datatype, but are rather interpreted as URIs, defined relative to the document base. Thus IDREFs are an exception, where a simple type does not denote the reference to a literal value.

XML Schema	RDF
element	property
attribute (except ID)	property
attribute (ID)	resource URI
predefined simpleType (not IDREF)	typed literal
untyped simpleType	plain literal
complexType (named)	resource type
sequence compositor	rdf:Seq
literals in mixed content	rdf:value properties
restriction (base type)	typed literal
attribute group	resource type

Table 2.6: RDF/XML Mapping

Sequence
XML documents implicitly denote the sequencing of the elements, which would be lost in a naive translation of XML into RDF. Even so, XML sequencing is often redundant; important sequencing is in reality only defined by the corresponding XML Schema, with the sequence compositor. As RDF lists as a way to define sequences are problematic - especially empty lists with properties - the authors choose to rely on RDF containers for their approach to represent XML sequences. This approach also has its own limitations, such as the weakly defined semantics of RDF containers (discussed in more detail in the previous section on the Normalization Toolkit).

Besides the inherent problems of RDF containers, another problem emerges when trying to use the same approach for lifting XML documents to OWL. In OWL DL container membership properties would need to be differentiated between Object and Datatype properties. In the SWWS approach, sequencing is seen as a data structuring issue, separate from the ontological details, and thus kept apart from the relational model generated by the lifting process.

Schemas and WSDLs
The lift methods defined in the lift API depend on the supplied XML Schema, which is used for the lifting process itself. In reality it is often difficult to know in advance which XML Schema should be used at run-time, particularly for messages received. An application can receive messages described by a broad variety of schemas, which would have to be collected together and referenced from a single schema using the schema import mechanism. As long as the document element of an XML message which should be lifted is globally defined in one of the imported schemas, the lifting process can work with this message.

Another possibility for the usage of the lifting process is to supply a reference to a WSDL file, instead of a reference to an XML Schema document. In this case only the XML Schema part either directly embedded in the WSDL file (or imported into the file as described above) will be used for the lifting process.

Mapping Summary
Table 2.6 is taken from the SWWS deliverable describing the lift API and denotes the RDF constructs mapped to the XML schema elements. A comparison to the similar table 2.5 from the previous section shows that several different decisions have been taken by the authors of this lifting approach.

CHAPTER 3

Theoretical Approach

This chapter will examine several lifting strategies which will be implemented as functionalities of the lifting component for the SEML Modeller. The actual implementation will be described in Chapter 4. The following two main sections examine the lifting of relational databases, and XML Schema, respectively. The lifting of other legacy data sources is briefly mentioned at the end of this chapter.

3.1 Design for a Relational Database Lifting Component

As mentioned in the survey on the state of the art of relational database lifting in Section, there are already various approaches to this problem, mainly in the scientific community. Quite a lot of these approaches are only suitable for a narrow application area or tailored for a specific scenario. One of the main design goals for the SEML Modeller was to provide a tool applicable to a wide area of integration problems.

Lifting Strategies

In the design of the database lifting described in the remainder of this section, a combination of the D2RQ approach, as discussed in Section 2.4, together with a set of modifications has been chosen. The D2RQ mapping language is expressive enough to fulfill all the requirements for lifting relational databases by mapping standard database models to an RDFS ontology, as discussed in the previous chapter. The design of a generator for such mappings is shown next, while the already available engine to execute D2RQ mappings will be used in the mediation tool with some modifications. D2RQ fulfills some of the requirements presented in Section 2.2. R2.1 and R2.2 are supported by the features of the D2RQ mapping language, while R2.3 and R2.4 are fulfilled by the lifting tool implemented in this work. For requirement R2.5, the support for all basic database properties is given, though some database constraints, including uniqueness, are not currently supported by D2RQ.

Figure 3.1 shows an activity diagram for the main functionalities of the database part of the lifting component, including the generation of an RDFS ontology and suitable D2RQ mappings from the database schema.

Two swimlanes are shown on this activity diagram, representing the Lifting Component itself, as well as the SEML Modeller tool user, who wishes to map an existing database (or a set of existing databases) to another ontology or data source. The activity diagram assumes the Lifting Component to be integrated with the SEML Modeller, and the concrete lifting of one or more relational databases to take place whenever the Modeller's user selects the source and target ontologies for a mapping project. Therefore the first action shown on the activity diagram is the provisioning of the database connection parameters by the tool's user. This integration with the Modeller's functionalities will be demonstrated in the next chapter, which is concerned with the actual implementation of the component.

After the lifting component has established the necessary database connections to each of the databases in the set provided by the user, every database is scanned in turn. The database schema is extracted (i.e., by using the relevant JDBC API methods to work with the `DatabaseMetaData` object), and this schema is made available to the remainder of the lifting mechanism as instances of the database object model. Figure 3.2 shows just the most important methods of the `DatabaseMetaData` object, as used for this process. Each JDBC driver has to implement these methods, adapting them to the actual database management system in question. The database object model itself is further explained in Section 4.2.

If only a single database is going to be lifted by the Modeller, the necessary artifacts for the lifting process are immediately created. An RDFS ontology is generated by applying a set of rules and mappings to the elements of the database model, in order to create the relevant ontology elements.

Afterwards a similar process is conducted to create a set of D2RQ mappings, which represent the connection between the database's elements (i.e., relations, attributes and constraints) and the previously created ontology elements (i.e., concepts and properties).

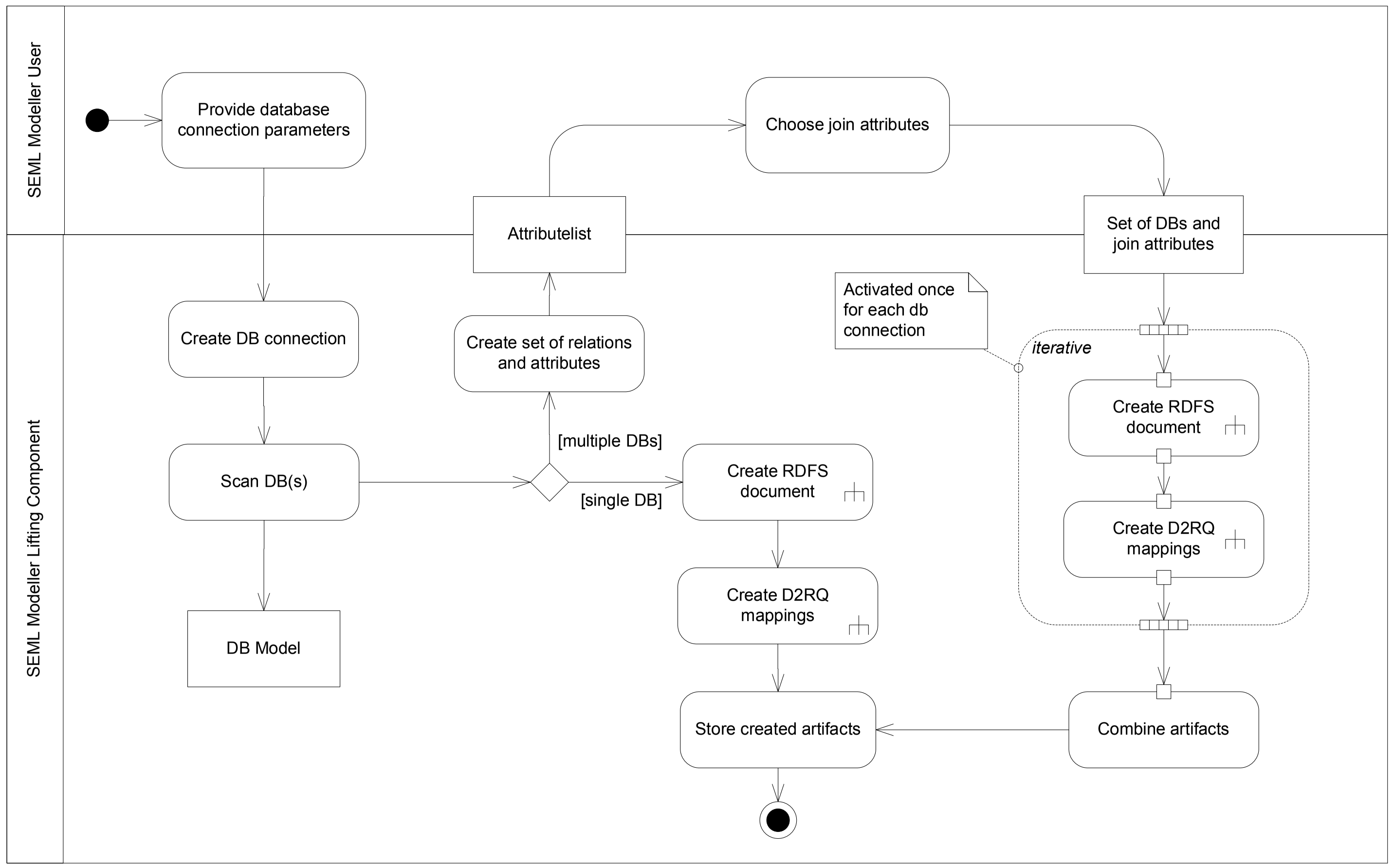


Figure 3.1: Relational Database Lifting - Overview

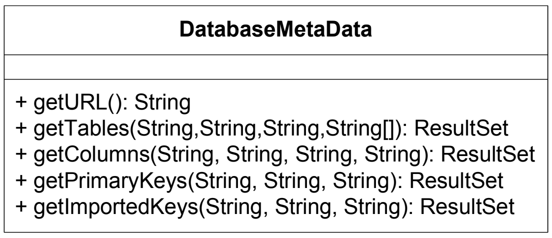

DatabaseMetaData
+ getURL(): String
+ getTables(String,String,String,String[]): ResultSet
+ getColumns(String, String, String, String): ResultSet
+ getPrimaryKeys(String, String, String): ResultSet
+ getImportedKeys(String, String, String): ResultSet

Figure 3.2: DatabaseMetaData Class

Finally both generated models are stored in a repository of the SEML Modeller, which contains other necessary information on the current mapping project. Additionally the models can be serialized and stored as separate files, according to the tool user's preferences, as determined by a set of configurable parameters in the SEML Modeller.

In the case of several databases to be lifted to a single ontology at the same time (which has been mentioned as a requirement for an integrated mediation tool in Chapter 2), an additional interaction with the user becomes necessary, before the generation of the RDFS and D2RQ models can commence. The collection of database relations (and their attributes) are provided to the SEML Modeller frontend, where they are visualized and presented to the user. The user then has the ability of selecting additional constraints across database boundaries (i.e., she can select foreign keys to be added to the already existing constraints of the selected databases). The constraints are supplied back to the lifting component, and the generation of a set of RDFS ontologies and corresponding D2RQ mappings starts similar to the single database lifting process, albeit in an iterative manner. The cross database constraints are finally used to combine all the generated artifacts to both a single RDFS ontology and a related D2RQ mapping rule model. These models are then stored as explained above. Further considerations concerning the case of lifting multiple databases are detailed in the section on "Additional design decisions" below.

Two of the most important actions from the main activity diagram, "Create RDFS document" and "Create D2RQ mappings", are further defined in the specific activity diagrams shown in Figure 3.3 and Figure 3.4, respectively.

Both activities share a sequence of similar actions, as both are basically concerned with the creation of ontological classes and properties. For the RDFS creation, a new ontological model is created as the first action inside this activity, using the provided namespace selected by the SEML Modeller tool user when first selecting the database(s) to be lifted. The collection of database relations (the tables represented in an object model), is used by an iterative process to create a suitable ontology concept (a class) for each relation in turn. After this a second iterative sequence of actions first selects each of the created classes, and then generates all appropriate ontological properties for it. These can include both attributes and relations (i.e., connections to other classes). Finally the created ontology is stored as a serialization in RDF-XML.

The D2RQ creation activity includes several additional preprocessing actions, after the creation of a new ontological model, as for the RDFS creation before. For the D2RQ model two namespace parameters are necessary, both of which are provided by the lifting component itself. The ontology namespace is the same as for the RDFS creation before, while the database

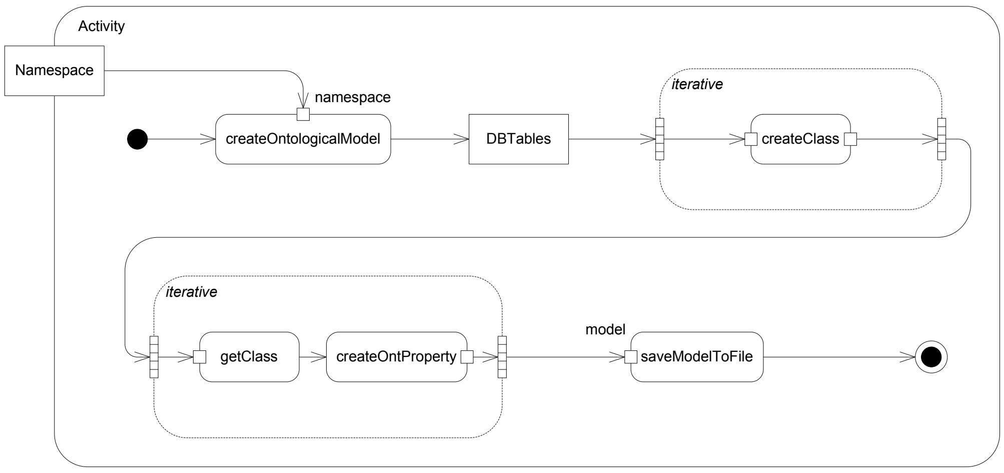


Figure 3.3: Relational Database Lifting - RDFS creation

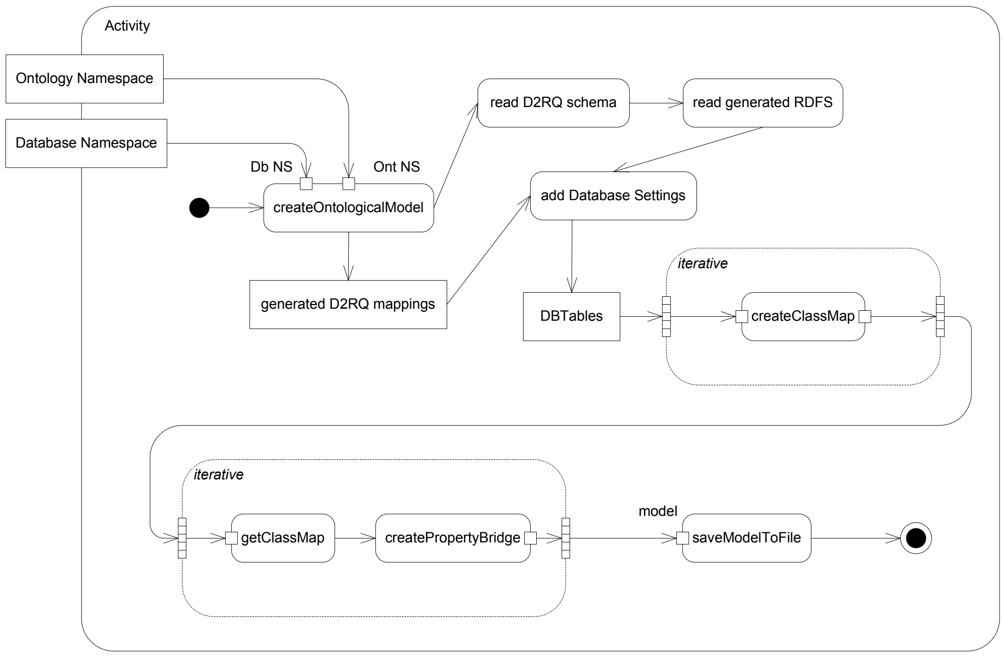


Figure 3.4: Relational Database Lifting - D2RQ creation

namespace is automatically generated by using the database name extracted from the database meta data.

After the empty model has been generated, both the D2RQ language schema itself, as well as the previously generated RDFS model, are read by two consecutive actions of the lifting component. The database settings for the particular database currently being lifted are added to the model as a set of specific resources. Finally, according to the D2RQ language specification explained in the survey on D2RQ functionalities, instances of `ClassMap` concepts are generated for each database relation iteratively, then suitable `ObjectPropertyBridges` are instantiated and attached to the `ClassMaps`. This activity concludes by again storing a serialized version of the generated model.

The specific implementation for these behaviors will be described in Section 4.2.

Additional Design Decisions

This section examines additional challenges encountered during the design and motivates the decisions made to resolve them.

Multiple Domains and Multiple Ranges

Since August 2001 it is possible to define multiple domain and range restrictions for properties in RDFS. The semantics of multiple domains and ranges are denoted to be conjunctive [22], meaning that a property is constrained by the conjunction ("and") of its domain/range constraints.

An example where this could pose a problem for the automated lifting to RDFS is presented next. The following is a basic entailment rule in RDFS, represented in first order logic syntax:

$rdfs : domain(p, s) \bigwedge p(x, y) \Rightarrow rdf : type(x, s)$

where p is a property, x is a source node in the RDF graph, y is a target node and s is the class for the domain of this property.

The above states that, given a property p, whose domain is the class s and a resource x which has a relation to a resource y via the property, it holds that the resource x is of type s.

The following sample database should be lifted to the RDFS layer (for the sake of readability, no namespaces are shown for RDF resources): A relation "books" has the attribute "hasAuthor" and a relation "movies" which has an attribute with the exact same name. The base lifting algorithm described in the previous sections would generate a property "hasAuthor", which has multiple domains: $rdfs : domain(hasAuthor, Book)$ and $rdfs : domain(hasAuthor, Movie)$. A common sense interpretation of these two domain constraints would be that both movies and books can have authors.

Given this, let us further say that the property instance

$hasAuthor(OWL\_Web\_Ontology\_Language\_Reference, Ian\_Horrocks)$

could be created from data stored in the "books" relation. The obvious interpretation of this statement would be that the book "OWL_Web_Ontology_Language_Reference" has the author "Ian_Horrocks".

Due to the conjunctive semantics of the domain (and range) restrictions in RDFS, the following is also inferred:

$rdf : type(OWL\_Web\_Ontology\_Language\_Reference, Book)$

and
$rdf : type(OWL\_Web\_Ontology\_Language\_Reference, Movie)$,
effectively stating that the resource "OWL_Web_Ontology_Language_Reference" is both a book and a movie.

This kind of entailment is perfectly correct with regard to the RDF Semantics, as this basically means that the domain constraint for the given property is an intersection of the classes Book and Movie ($Book \bigcap Movie$). Still, for the automated lifting this proves to be counterintuitive, and to prevent this behavior from occurring, the lifting component introduces a naming convention for lifted properties. An obvious solution is to just add the name of the relation as a prefix to the name of the attribute, e.g., in the above example two properties would need to be generated - "book_hasAuthor" and "movie_hasAuthor". Each of these properties would then have a single domain constraint. While this effectively prevents the occurrence of the type inferencing described above, a side effect would be that some of the hidden semantics stored in the relational schema is lost (i.e., the notion that a property like "hasAuthor" can be applied to multiple differing resources).

The naming convention has been defined as configurable and can be switched off by setting the necessary parameters within the SEML Modeller. While this might be actively desired by a mediation tool user, it is a definitive requirement once lifting is conducted to produce an ontology represented in an OWL dialect (as further explained in Section 7.2).

Lifting Multiple Databases to a Single Ontology

One of the main requirements defined for the relational lifting component among those collected in Chapter 2 is the possibility to lift a set of legacy databases to a single ontology. This is generally feasible by the use of D2RQ as the declarative mapping language, as it already contains primitives which allow to specify for each mapping rule to which database it applies. An automated lifting component has to generate both, the connection to the databases, as well as the respective mapping rule parameters. Also, the lifting component has to combine all separately generated artifacts to provide a single mapping file and thus access to a single virtual graph. In Figure 3.1 these considerations are already depicted and will be explicated in the following section.

The lifting component evaluates the database key constraints to correctly generate property bridges and the necessary parameters. As discussed in the section on D2RQ, a `d2rq:join` property is used to join together the table that was used to generate the containing `ClassMap` and the table(s) containing the columns to create the literal values or objects. Multiple databases cannot be joined in this way since no foreign key constraints usually exist across database boundaries[1]. Therefore, the automated lifting of multiple schemas to a single virtual RDF graph would result in several disconnected graphs.

As a solution to this, user interaction is necessary to choose the correct join attributes. Once the user has selected more than one database to be mediated to an existing ontology, the user is presented with a view of the two (or more) schemas and can use the SEML Modeller to define

[1]Of course many commercial database systems provide mechanisms to solve this, for example Oracle systems specifically allow to query multiple databases via a separate higher-level abstraction layer

the attributes to join. The lifting component will then use these selected attributes to add the missing property bridges and corresponding `d2rq:join` properties.

As has been noted by the author in [26], multiple relational databases have to be joined in D2RQ by specifying the join attributes with "dummy" `ClassMaps`. The reason is that the `ClassMaps` linked by a Property Bridge have to refer to the same data source (i.e., the `d2rq:dataStorage` property of the `ClassMaps` have to match). An example for this is given in Listing 3.1[2], which shows a fragment of a D2RQ map (The complete D2RQ mapping for this running example can be found in the Appendix A.3). The `webshop:TrackingClassMap`, whose dataStorage property refers to `webshop: Database2`, is not directly linked to the `customer:OrderClassMap`, which instead refers to `webshop:Database3`. Therefore, a separate `ClassMap` is introduced, called `webshop:TrackingOrderDummyClassMap` in the example. This `ClassMap` is linked to the `webshop:TrackingClassMap` by an Object Property Bridge. As both the "dummy" `ClassMap` and the `customer: OrderClassMap` use a URI pattern which will create an instance by appending the "orderid" attribute to the same namespace, the generated triples will refer to the same resource. Thus the resulting RDF graph will actually contain the correct relations, as was intended by joining the data sources.

```
webshop:TrackingClassMap rdf:type d2rq:ClassMap;
            d2rq:class :TrackingKeyword;
            d2rq:uriPattern "http://www.niwa.at/statistik/concepts/Tracking@@tracking2.
                trackingid@@";
            d2rq:dataStorage webshop:Database2;
            .

webshop:TrackingOrderDummyClassMap rdf:type d2rq:ClassMap;
            d2rq:class :TrackingOrderDummy;
            d2rq:uriPattern "http://www.niwa.at/statistik/concepts/Order@@tracking2.
                orderid@@";
            d2rq:dataStorage webshop:Database2;
            .

webshop:TrackingOrderToOrder rdf:type d2rq:ObjectPropertyBridge;
            d2rq:property :hasOrder;
            d2rq:condition "tracking2.orderid>0";
            d2rq:belongsToClassMap webshop:TrackingClassMap;
            d2rq:refersToClassMap webshop:TrackingOrderDummyClassMap;
            .

customer:OrderClassMap rdf:type d2rq:ClassMap;
            d2rq:class :Order;
            d2rq:uriPattern "http://www.niwa.at/statistik/concepts/Order@@t_order.
                orderid@@";
            d2rq:dataStorage customer:Database3;
            .
```

Listing 3.1: D2RQ Mapping for a Join Across Database Boundaries

This technique will be introduced to the lifting component. The selected join attributes will result in the generation of "dummy" `ClassMaps` and corresponding Property Bridges as needed.

[2]The mappings are shown in Notation 3 (N3), a human readable RDF syntax.

3.2 Design of an XML Data Lifting Component

The second major legacy data source taken into consideration in the scope of this work are XML Schema documents. The functionalities of the lifting component where extended suitably, in order to support the import of XML Schema as ontologies in the SEML Modeller tool.

Lifting Strategies

As described in the state of the art survey on XML lifting techniques in Section 2.5, the Normalization Toolkit has been selected as the component to enable the lifting of XML documents, by lifting XML Schema documents to an RDFS ontology. Regarding the set of requirements listed for the lifting of XML documents, Normkit directly fulfils requirements R3.1 and R3.3. Furthermore it is possible to adapt the lifting mechanism for additional schema languages (requirement R3.2), though this would necessitate a new Conceptual Normalization Ontology (CNO) for the characteristics of the schema language. R3.4 is not currently fulfilled, but could conceivable be added to the automated lifting tool, as it is possible to define new normalization rules and to enable the tool user to choose from those applicable to a given mapping situation. Finally, due to the tools integration with the hMAFRA implementation of the MAFRA framework, it is easily adoptable to the needs of the SEML mediation tool based on MAFRA.

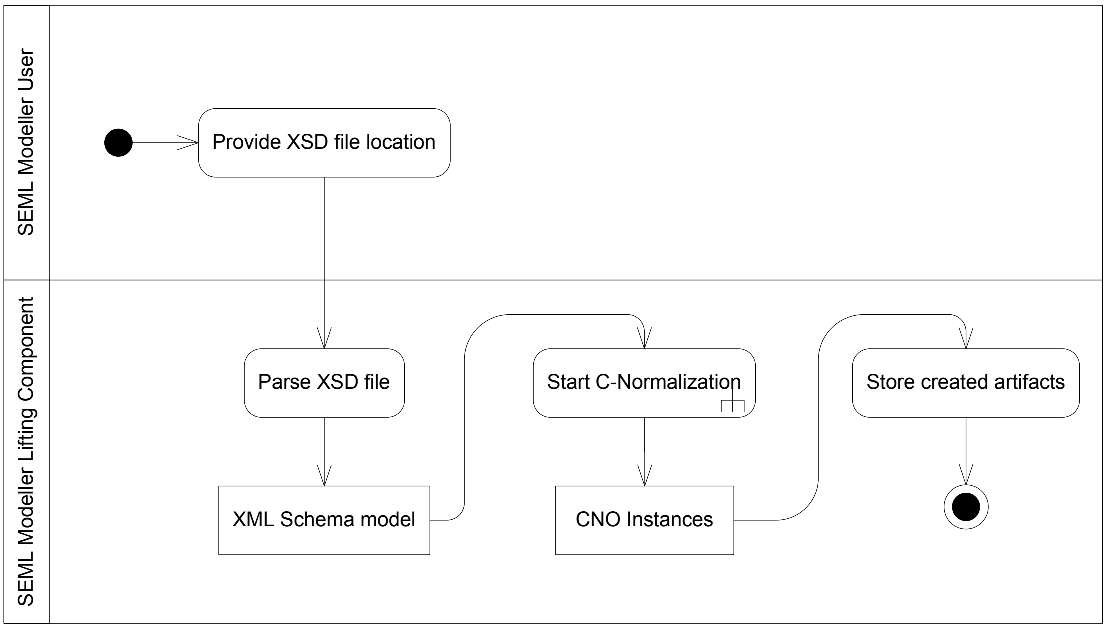


Figure 3.5: XML Schema Lifting - Overview

Figure 3.5 shows the activity diagram for the actions carried out by the Normalization toolkit, including the generation of an RDFS ontology and the generation of the mappings, which are instances of the CNO, as discussed previously. The remainder of this section will explore modifications and extensions which were carried out in order to integrate the Normalization Toolkit with the SEML Modeller.

The core action of this activity diagram, the C-Normalization process, is shown in more detail in Figure 3.6. After two preparation steps, including the creation of the target model and the initialization of a new normalization map, the first phase includes the parsing of the schema's top level complex types. For each globally defined complex type definition, the set of heuristic rules corresponding to the complex type in question is applied. The next action during C-Normalization conducts the same process for globally defined element and attribute declarations, again executing all applicable normalization heuristic rules. After the execution of these rules for the top level entities, the content of first the complex types (i.e., nested elements and attributes) and then the global elements (i.e., the context of the element, consisting of the element's type and scope of it's declaration) is evaluated and matching rules are applied. Finally the rest of the elements of the XML Schema document are traversed and processed. As a result of these actions, both the target model is created and a set of instances of the CNO are generated, which describe the correlation between the target ontology's elements and the XSD elements.

Extensions to the Normalization Toolkit

While the basic functionalities delivered by the Normalization Toolkit have been reused for the XML document lifting component, a few extensions and changes were necessary to better fit the tool to the SEML Modeller. These modifications are detailed in this section.

A first modification was to constrain the list of applicable heuristics, as the process of conceptual normalization should be conducted fully automatically. While the authors of [20] state that it is actually not conceivable to fully automate this process of normalization without loosing important information hidden in the structure of the XML Schema, the usage of the lifting component inside an ontology mediation tool mitigates these problems. The normal lifting procedure set forth by the Normkit, as described in Section 2.5 consists of:

- The reverse engineering of the XML Schema, utilizing the set of generic mapping rules, called Normalization Heuristics. Human interaction is expected, by providing choices between applicable heuristics.
- A post-processing phase, again involving human interaction for the optimization of the generated Ontology.
- The execution of the mapping rules themselves, migrating the XML data to a corresponding RDF repository.

Thus, the Normalization Toolkit promotes direct interaction with an user at two steps in the lifting procedure, first while choosing heuristics and then again during post-processing. In an expanded scenario, where mediation is conducted between diverse legacy data sources, the post-processing phase is superseded by the interaction of the tool user with the SEML Modeller, after the preliminary lifting to an ontological layer has taken place. Therefore the necessary semantical information can still be extracted from the legacy data source by applying suitable mapping rules between the source and target ontologies.

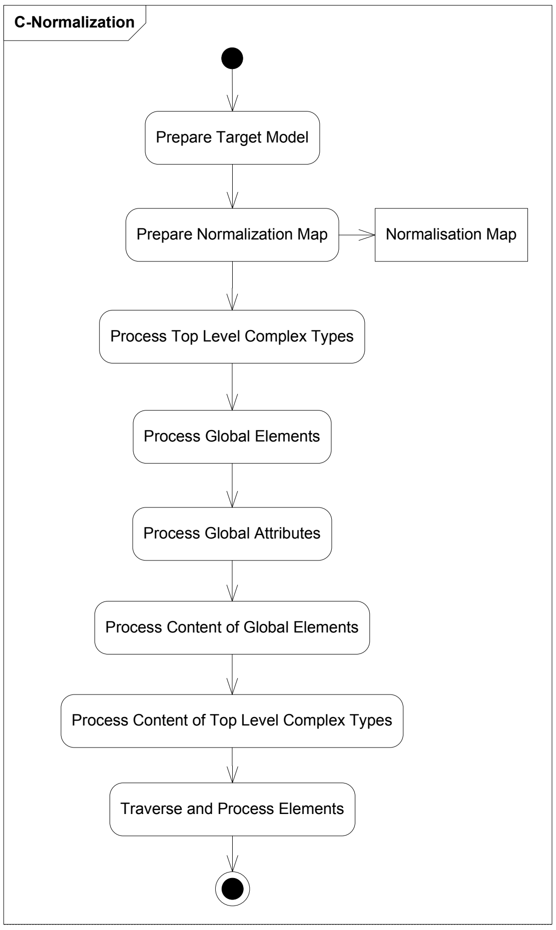

C-Normalization
Prepare Target Model
Prepare Normalization Map
Normalisation Map
Process Top Level Complex Types
Process Global Elements
Process Global Attributes
Process Content of Global Elements
Process Content of Top Level Complex Types
Traverse and Process Elements

Figure 3.6: XML Schema Lifting - C-Normalization

In a similar vein, the possibilities inherent in the mapping of ontology elements and different transformation services supported by the SEML Modeller do not necessitate additional lifting decisions by the tool user. Thus, all heuristics which might need user interaction have been removed from the standard XSD lifting component added to the SEML Modeller. Also, this means that if a selection of different heuristical rules applicable to a single situation during the lifting of a given XML Schema, a default rule is instead chosen and applied.

Finally, an additional wrapper for the Normalization Toolkit has been included with the SEML Modeller. The wrapper makes the necessary methods of the C-Normalization step available to the UI objects of the Modeller. Additionally the execution phase (i.e., the D-Normalization) is made available, and used by the mapping execution functionality of the tool. Chapter 4 contains further details on this integration of the execution component with the SEML Modeller.

3.3 Mediation Support For Additional Data Sources

Besides data stored in relational databases or in XML documents described by an XML Schema, several other legacy data sources might exist in a heterogeneous environment. While not the focus of this work, these additional data sources could be integrated with the mediation tool as well, given that there is a way to automate the lifting of the underlying data model to an ontology in order to provide the user of the SEML Modeller a unified view of all data sources. Examples where such a strategy would be feasible are any kind of data sources, which are at least semi-structured, such as data extracted from HTML pages, EDI[3] data etc. Currently a major topic, the proliferation of community-created schemas and widespread adoption of so-called microformats[4] offers another venue for possible data sources, which can be easily lifted to simple ontologies.

Another common case would be the need to integrate data provided by a trusted third party. Usually data from external sources will not be available in a completely open way (i.e., there will be no direct access to a third party provider's databases). Rather such a provider will grant access to his data by the way of Web services, with the format of the messages being described by a WSDL document, or by other public interfaces[5]. This Web service description will therefore contain an XML Schema definition part for the messages (or at least link to an external schema document), thus in effect constituting the underlying data model for the messages exchanged. An example fragment for such a WSDL description and the message format used by the Web service described therein is presented in Listing 3.2.

```
<definitions name='OrderManager'
xmlns='http://schemas.xmlsoap.org/wsdl/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <types>
   <schema targetNamespace='http://webhosting.niwa.at'>
   <complexType name='DetailedOrder'>
    <sequence>
     <element name='accountid' nillable='true' type='long'/>
     <element name='actualcancellationdate' nillable='true' type='dateTime'/>
     <element name='invoiceid' nillable='true' type='long'/>
     <element maxOccurs='unbounded' minOccurs='0' name='payments' nillable='true' type=
         'Payment'/>
     <element name='resellerid' nillable='true' type='long'/>
    </sequence>
   </complexType>
  </schema>
  <!-- type definitions for other message parts -->
 </types>
 <message name='OrderManagerEndpoint_finalizeOrder'>
  <part name='Long_1' type='xsd:long'/>
  <part name='Long_2' type='xsd:long'/>
  <part name='DetailedOrder_3' type='tns:DetailedOrder'/>
 </message>
 <portType name='OrderManagerEndpoint'>
   <operation name='finalizeOrder' parameterOrder='Long_1␣Long_2␣DetailedOrder_3'>
   <input message='tns:OrderManagerEndpoint_finalizeOrder'/>
```

[3]Electronic Data Interchange

[4]detailed for example at `http://microformats.org/`

[5]Another example would be the Extensible Provisioning Protocol (EPP), used by many domain registrars

```
   <output message='tns:OrderManagerEndpoint_finalizeOrderResponse'/>
  </operation>
    ...
 </portType>
 <binding name='OrderManagerEndpointBinding' type='tns:OrderManagerEndpoint'>
    ...
 </binding>
 <service name='OrderManager'>
    ...
 </service>
</definitions>
```

Listing 3.2: WSDL Fragment

As can be seen from the listed WSDL fragment, the data model underlying the messages exchanged with the Web service is described in the definition of the *types* and *messages* sections of the WSDL document. In order to integrate this data it is therefore not necessary to directly access the original data model used by the third party provider, but rather the relevant parts of the WSDL document can be extracted and lifted to an RDFS ontology. Major parts of the lifting component for XML documents could be reused for this envisioned lifting component for Web services, which will be considered for future work to be conducted.

CHAPTER 4

Implementation Details

In this chapter, the implementation of the XML Schema and the relational database lifting components as presented in the previous chapter is examined. The first section will describe the SEML Modeller, the integrated lifting tool which has been developed in the SEML project. Following this introduction, the next two chapters describe implementation challenges and the resulting components for the lifting of relational databases, and for XML documents, respectively.

4.1 SEML Modeller

The SEML project was concluded in the fall of 2006 and resulted in the prototypical implementation of two separate software deliverables. The SEML Modeller was conceived as the design-time tool to be used by a domain expert, to map the lifted ontology generated from diverse data sources to a suitable target ontology (i.e., a central ontology designed for the interoperability of various data sources and applications). Furthermore, the SEML Transformation Services consist of a set of related services which interpret and execute the mapping rules created during the design phase. The services were designed to support both typical use cases of schema lifting and ontology mapping (as defined in Section 2.1), therefore methods for both instance migration and query rewriting are provided.

As the lifting components resulting from this work form a part of the SEML Modeller, and provide the necessary preprocessing steps needed by the tool, the Modeller itself is briefly described next. Possibilities for extension and the manner in which the user interacts with the lifting component are specifically highlighted.

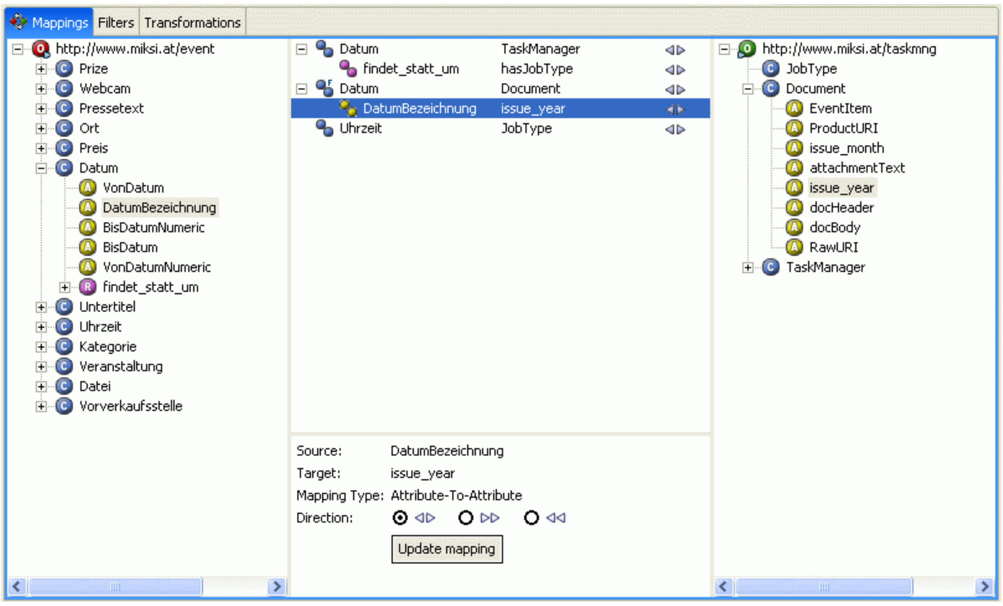
Modeller Overview

The SEML Modeller has been designed as a stand-alone application based on the Eclipse Rich Client Platform[1], not only benefitting from the support of an easily extensible application pro-

[1]the Eclipse project, at `http://www.eclipse.org/`

gramming framework, but also resulting in a platform independent solution.

The SEML Modeller's basic functionality concerns the creation of mappings between elements of source and target ontologies. Once source and target ontologies are imported into the project, the mapping process can be started. Therefore, the ontologies must be loaded into the Mapping View, the core user interface component of the SEML Modeller. Figure 4.1 shows the layout of a Mapping View, where a mapping project has already been created. It displays the selected ontologies (source on the left, target on the right) in the form of browse-able trees. Each node specifies a resource, whose type can be identified by a small icon. The Mapping Browser is situated between the ontologies. It allows to browse already created mappings, and shows source and target elements as well as the mapping directionality. Below the Mapping Browser the Mapping Editor allows for the editing of mapping attributes.



Datum	TaskManager
findet_statt_um	hasJobType
Datum	Document
DatumBezeichnung	issue_year
Uhrzeit	JobType

Source: DatumBezeichnung
Target: issue_year
Mapping Type: Attribute-To-Attribute
Direction: [✓ ◁▷] ▷▷ ◁◁

Figure 4.1: SEML Modeller Mapping View

Importing Data Sources

As already mentioned, a SEML mapping project can include several source ontologies and one target ontology. Importing an ontology is provided by the Ontology Import Wizard. The first page of the wizard allows the selection of the desired data source. By default, the SEML Modeller receives it's ontology data by parsing RDFS files, but it also has the abilities to allow the import of XML Schema descriptions, as well as a relational database schema. Figure 4.2 depicts the Import Wizard.

The RDFS import procedure begins with the selection of an adequate document. By clicking on the "Finish" button, the SEML Modeller starts to parse the specified file. No other user interaction is needed, thus the import process will finish and the ontology will be added to the project. However, the parser is not always able to find a non-ambiguous namespace reference for the specified RDFS file, which can happen if the RDFS file contains multiple top level elements. In that case, a selection dialogue will be displayed, where the user can select or define a namespace.

In case of an XML Schema import, the user has to perform similar steps. First, an XSD file must be selected. Once the import procedure starts, the user may be asked for the schema's root element (only if is ambiguous) and, as in case of the RDFS import, for the ontology's namespace. Usually XML Schema documents do not include any self-describing namespace references, so – in almost every case – the users will be requested to provide their own namespace definition.

If the user wants to import a relational database into the SEML Modeller, two different options are available. Either a single database schema can be lifted to an ontology, or multiple database schemas can be combined by the way of cross database joins. After specifying the necessary foreign key constraints, the databases will be presented as a single ontology in the mapping view.

Several parameters have to be provided by the tool user, once the import wizard for a single database has been started. Besides the necessary connection parameters, such as the database URL, the database name, username and password, other options include the database driver to be used and the namespace for the generated ontology. Finally, the user can specify a pattern to be evaluated against the relations of the database schema, which will constrain the relations that have to be lifted, as they are filtered according to the supplied pattern. Also, the "tabletype" parameter allows the user to specify which kind of database tables should be lifted by the Modeller (e.g., only normal tables, no system tables).

Figure 4.3 shows the Import Wizard page for the lifting of multiple databases. According to the design in Section 3.1, several additional pages have been added to the basic database Import Wizard. These are described in the corresponding sections of the implementation of the lifting

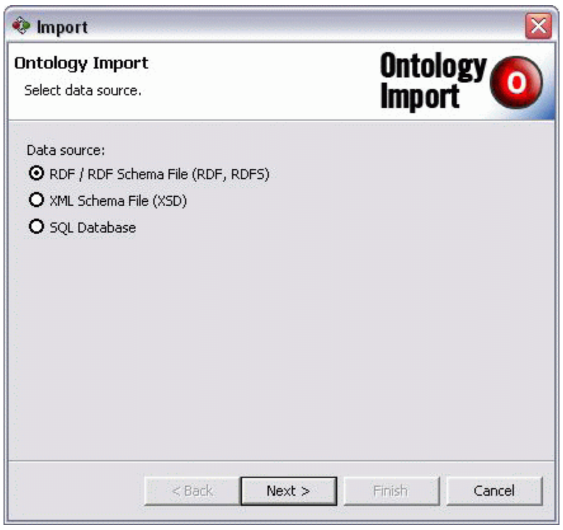


Figure 4.2: Ontology Import Wizard

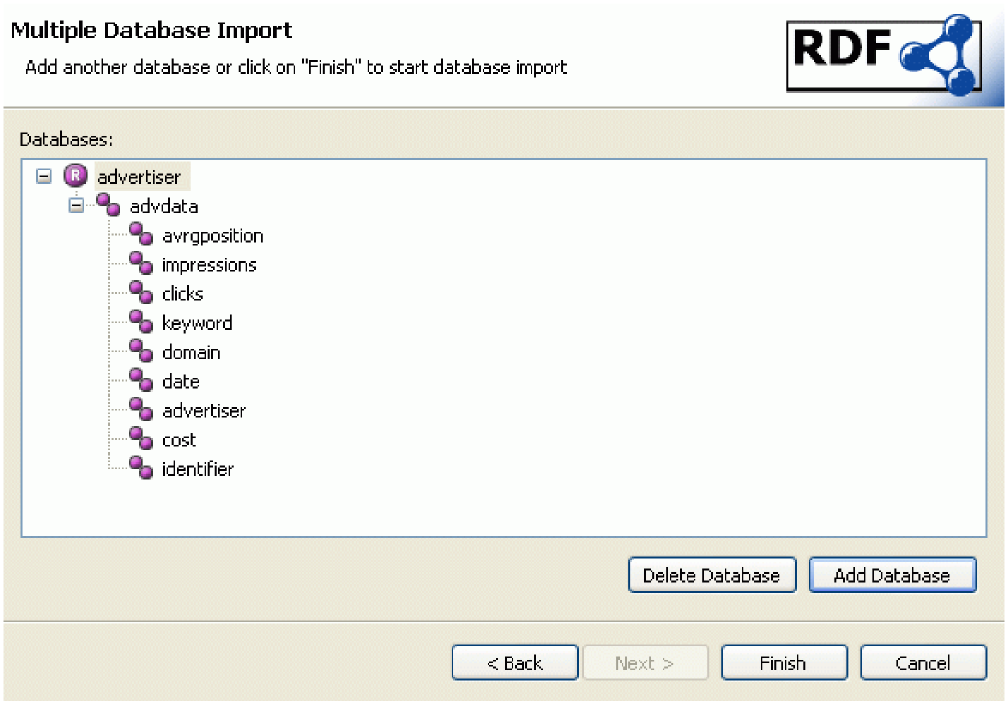


Figure 4.3: Database Import Wizard

component for relational databases.

As explained, the wizard package is the main collection of classes responsible for the lifting and import of the various data sources supported by the SEML Modeller. Figure 4.4 depicts the internal set-up of the relevant classes contained within. Among these, the different wizards are responsible for a set of one or more pages each. A page is an actual user dialog, consisting of such interface elements as input text fields, buttons and various views. The basic set-up is provided and constrained by the Eclipse Modelling Framework.

4.2 Lifting Component for Relational Databases

This section contains the UML class diagrams for all lifting classes and the additions to the SEML Modeler; in addition different implementation challenges are highlighted, in correspondence with the findings of Chapter 3.

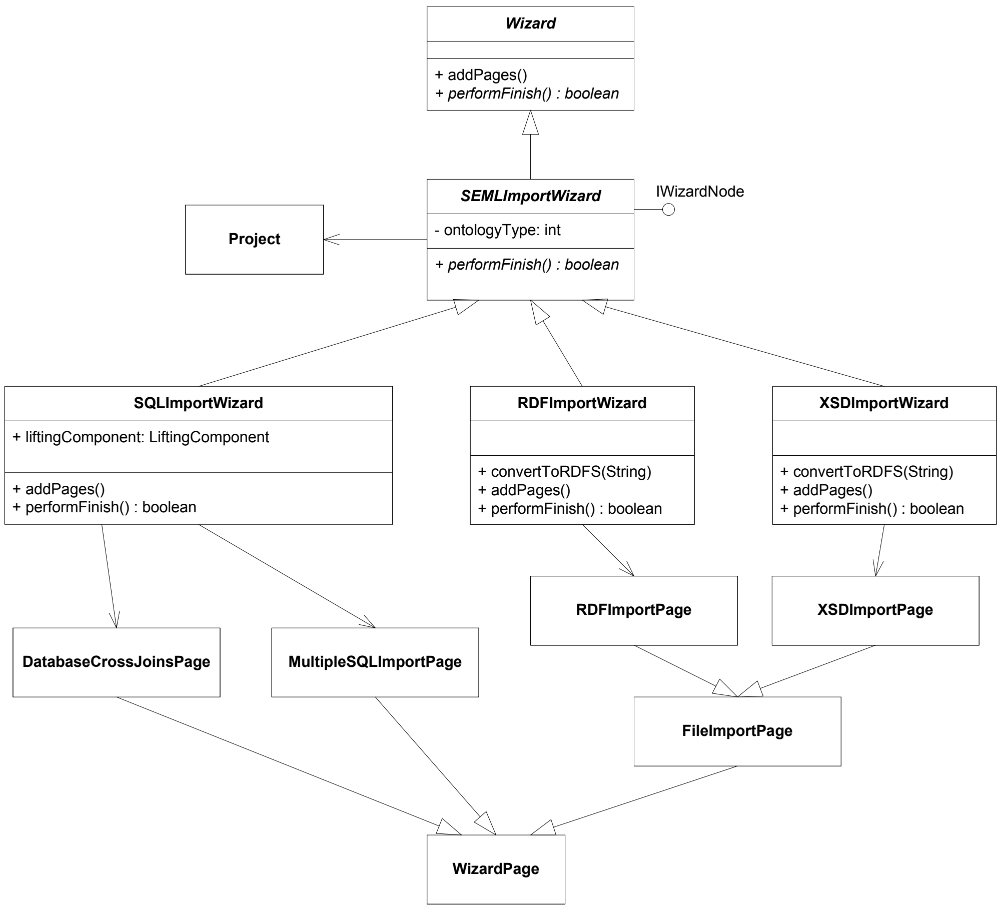


Figure 4.4: SEML Modeller UI classes

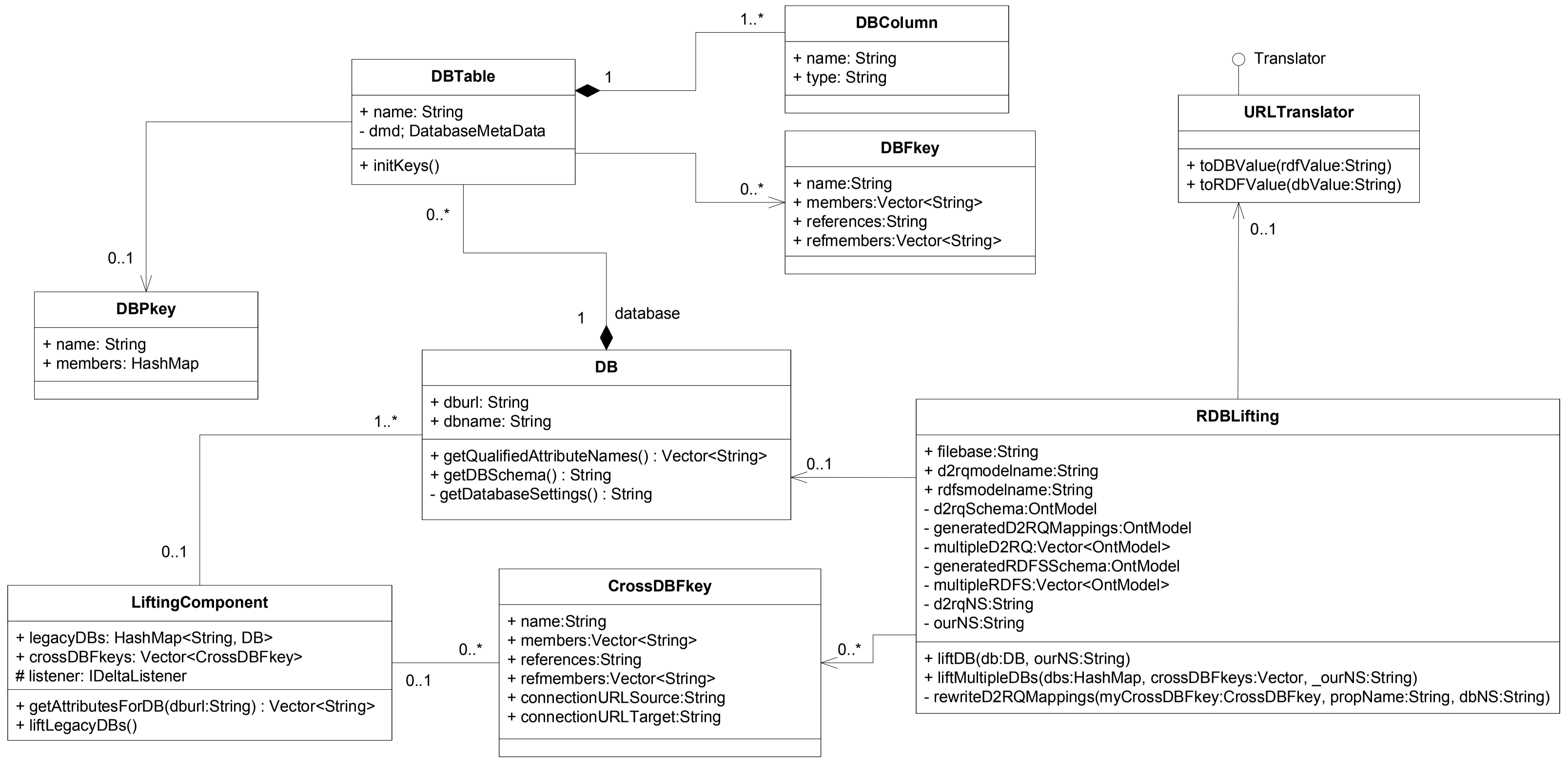

DBTable
+ name: String
- dmd; DatabaseMetaData
+ initKeys()
DBColumn
+ name: String
+ type: String
DBFkey
+ name:String
+ members:Vector<String>
+ references:String
+ refmembers:Vector<String>
DBPkey
+ name: String
+ members: HashMap
DB
+ dburl: String
+ dbname: String
+ getQualifiedAttributeNames() : Vector<String>
+ getDBSchema() : String
- getDatabaseSettings() : String
Translator
URLTranslator
+ toDBValue(rdfValue:String)
+ toRDFValue(dbValue:String)
RDBLifting
+ filebase:String
+ d2rqmodelname:String
+ rdfsmodelname:String
- d2rqSchema:OntModel
- generatedD2RQMappings:OntModel
- multipleD2RQ:Vector<OntModel>
- generatedRDFSSchema:OntModel
- multipleRDFS:Vector<OntModel>
- d2rqNS:String
- ourNS:String
+ liftDB(db:DB, ourNS:String)
+ liftMultipleDBs(dbs:HashMap, crossDBFkeys:Vector, _ourNS:String)
- rewriteD2RQMappings(myCrossDBFkey:CrossDBFkey, propName:String, dbNS:String)
LiftingComponent
+ legacyDBs: HashMap<String, DB>
+ crossDBFkeys: Vector<CrossDBFkey>
# listener: IDeltaListener
+ getAttributesForDB(dburl:String) : Vector<String>
+ liftLegacyDBs()
CrossDBFkey
+ name:String
+ members:Vector<String>
+ references:String
+ refmembers:Vector<String>
+ connectionURLSource:String
+ connectionURLTarget:String
1
1..*
0..*
0..*
0..1
1
database
1..*
0..1
0..1
0..1
0..*
0..1
0..*

Figure 4.5: Database Class Diagram

Creating RDFS Elements

Creating the RDFS ontology for the supplied database schema(s) involves the following steps.

First a new RDFS Ontology Model is created using the Jena API's createOntologyModel method. The model is created in memory and only later serialized and saved in the "RDF/XML-ABBREV" format, which produces human readable output. As a general convention, the naming of all elements generated for this model follow the rule that classes need to begin with a capital letter, and properties with a small letter.

The next step sees the creation of all classes from the relations of the chosen database. Properties are created afterwards, since relations might refer to other classes which may not have already been created otherwise.

In Section 3.1 we discussed the problems arising from allowing the use of multiple domain and range constraints for a single property in RDFS. Besides the wrong entailments which might result from this, another problem would occur if multiple ranges were allowed for attributes using simple datatypes. As an example if two different database relations have a column with the same name, and the fields for this column are constrained to integers in one case and to boolean in another, the automated lifting with multiple ranges allowed would therefore constrain the range of the resulting RDFS attribute to both datatypes. As noted in the design section, this is prevented by a unique naming convention: For each property created, the table name is concatenated with the column name to derive a unique identifier in the RDFS ontology.

For each table column , the kind of property to be created is selected. In a similar manner to D2RQ Property Bridge creation detailed in the next section, the use of a particular column in a foreign key is representing the main decision criterium of whether to create a literal property (an attribute) or a relation to another class. Furthermore, for the literal properties either an untyped RDFS literal is chosen as the range constraint, or, if applicable, a suitable XSD datatype can be used for this role.

Table 4.1 shows the correlation between database field types from both MySQL and PostgreSQL databases and XSD datatypes (both primitive and derived types, as defined by [7]). Any database field type not listed is interpreted as an RDFS literal. Other RDBMS can be mapped to XSD datatypes in a similar manner.

Finally the model is closed and written back to the assigned file, to be used in later steps of the lifting process.

Creating D2RQ Elements

The following steps are conducted to create the necessary D2RQ elements, which map the database schema to the RDFS ontology. The ontology itself was generated as described in the previous section.

As the D2RQ mappings to be created are themselves instances of the D2RQ language specification ontology, in the first step a new RDFS Ontology Model is created using the Jena API, more specifically the ModelFactory's `createOntologyModel` method. The model is created in memory and only later serialized in the "N3-PP" format[2] and saved in a file. The D2RQ

[2]a formatted version of Notation 3, described at `http://www.w3.org/2000/10/swap/Primer.html`

PostgreSQL Data Type (aliases)	MySQL Data Type	XSD Datatype
bigint (int8)	bigint	XSD:long
integer (int, int4)	int	XSD:int
smallint (int2)	smallint	XSD:short
real (float4)	float	XSD:float
double precision (float8)	double	XSD:double
character varying (varchar)	varchar	XSD:string
text	text	XSD:string
boolean (bool)	tinyint	XSD:boolean
timestamp	timestamp	XSD:dateTime

Table 4.1: Relation between Database Field Types and XSD datatypes

language specification is imported next, as another Ontology Model. The current specification is available to the lifting component as a file in N3 format. This model of the language specification will be used in the following steps to fetch the classes which are instantiated for a concrete D2RQ mapping description. Finally a third model is imported: the previously generated RDFS Ontology itself. This model will then be used to actually relate Class Maps and Property Bridges to ontology elements. The premier step in the actual generation of contents for the D2RQ model is the provision of the database settings. As explained in Chapter 3, even in the case of lifting multiple databases to one connected virtual graph, only one database is mapped at a time, so all necessary connection parameters are added to the model, as well as a series of additional values. Among those, are attributes which define the datatype for each column. As these definitions influence the quoting style to be used in the query translation, and as we don't want to constrain which attributes are actually going to be used in a query, all values for all database columns have to be added in this way. Following this, the Class Maps for each relation in the database schema are created. This has to be done first, before any Property Bridges can be created, as the Property Bridges relating to objects might reference arbitrary Class Maps. Next, the Property Bridges for each database column are created, as explained in more detail below.

Property Bridges

Two kinds of property bridges need to be created in D2RQ, as was shown in Section 2.4. Object property bridges link several database tables and result in the creation of a relation in the generated RDF instances. Datatype property bridges are responsible for the creation of literal attributes for the generated instances, which means that they steer how the name of the property is determined and which value from the database is to be used as the object of the corresponding statement. Therefore, the first decision, when creating property bridges, is to determine which kind of bridge should be generated. The foremost decision criterium is whether the column regarded is used as a part of a foreign key in the database (i.e., whether it's values are constrained by a foreign key linking to another relation). For the columns which are used in a foreign key, an Object Property Bridge is created, otherwise a Datatype Property

Bridge is suitable. Datatype Property Bridges can be built in a straightforward manner. Four different properties have to be defined for an instance of the Property Bridge (from Table 2.3): `d2rq:belongsToClassMap`, `d2rq:property`, `d2rq:column` and `d2rq:datatype`. For `d2rq:datatype` the same RDF datatype is chosen as in the creation of the RDFS property itself, that is either a suitable XSD datatype can be used (again according to the equivalencies shown in Table 4.1), or the value is an untyped RDFS literal. Object Property Bridges on the other hand need the following properties, taken from Table 2.3: `d2rq:belongsToClassMap`, `d2rq:property`, `d2rq:refersToClassMap` and `d2rq:join`. For the `d2rq:refers-ToClassMap` property the correct Class Map is determined by finding a suitable Class Map for the referenced table in the foreign key. In a similar vein, the `d2rq:join` property is initialized by picking up the referenced attributes of the foreign keys. For the case of joining multiple databases, some Datatype Property Bridges will have to be removed in a later step and replaced with corresponding Object Property Bridges. This mechanism is described in Section 3.1 and the implementation described below.

Finally the model is closed and written to a file, the name of which was specified by the user during the dialogue with the Import Wizard.

Joining Multiple Databases

As was described in Section 3.1, multiple databases are joined together by evaluating user provided foreign keys. In order to specify these foreign keys across database boundaries, the Database Import Wizard of the Modeller has been expanded with suitable wizard pages. The user first adds all the databases which should be imported, providing all connection parameters as in the case of a single database import (of course it is also possible to remove database via the Database Import Wizard). On the next page, the wizard presents all databases as a set of browseable trees on the left side, while the attributes to be joined are shown to the right, as seen in Figure 4.6. The user can drag and drop attributes from the relations to the colored boxes of a join attribute row. In the figure, two relations have already been joined, shown in the top row of the cross-join area. Finishing the wizard causes the lifting of the joined databases to commence.

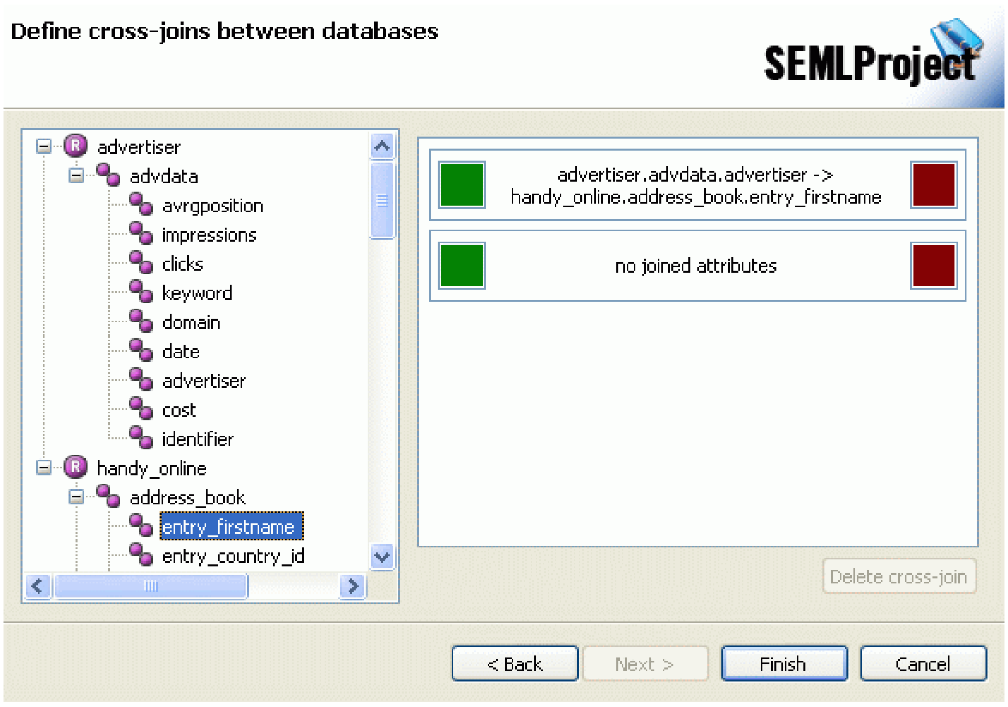
After all single artifacts, RDFS Ontologies and D2RQ documents, have been generated by the lifting component, they are first joined together by combining the separate models via Jena's `Model.add` method. This results in combined models for both the RDFS Ontology and D2RQ mappings, which need to be further refined by resetting their namespace prefixes. The last step involves the evaluation of the foreign keys. For each `crossDBFkey` defined by the Modeller user, the relevant source database is picked up. The RDFS model then needs to be adapted, as the relevant property (i.e., the property created from the foreign key attribute) is transformed from an attribute to a relation. This involves changing the range of the property, as the allowed object for the property needs to be constrained to instances of the target class. The D2RQ mappings finally need to be rewritten in a similar manner. After the creation of a "dummy" Class Bridge, the Datatype Property Bridge for the source attribute is changed into an Object Property Bridge, related to the dummy Class Bridge. This Class Bridge's URI attribute has to be equivalent to the Class Bridge of the target Ontology, as noted in the design section.

4.3 Lifting Component for XML Documents

For the integration of the Normkit with the SEML Modeller, an XML Schema Import Wizard was created, the main page of which features a file selection dialog to choose the XSD file to be lifted. After selecting the XSD file, the user has to choose an Ontology URI to uniquely identify the newly created ontology, either by supplying her own URI, or by selecting a namespace from the XML Schema and concatenating this with a provided local name.

The lifting functionalities have been directly taken from the Normkit, and are equivalent to the version described in Section 2.5. The class XSDConverter, shown in Figure 4.7, subclasses the main C-Normalization class from Normkit, which conducts the first phase of the conceptual normalization. The most important methods are explained in the following section.

The XSDConverter class itself is integrated with the SEML Modeller GUI, as it provides two central methods for the lifting of the given XML Schema: the method `getRootCandidateNames` finds the suitable candidates for the generated Ontology's root element, from which the tool user can select a single element via the file selection dialog of the XSD Import Wizard. The method `toRDFS` initializes the C-Normalization phase of the Normkit, and returns the generated RDFS document.

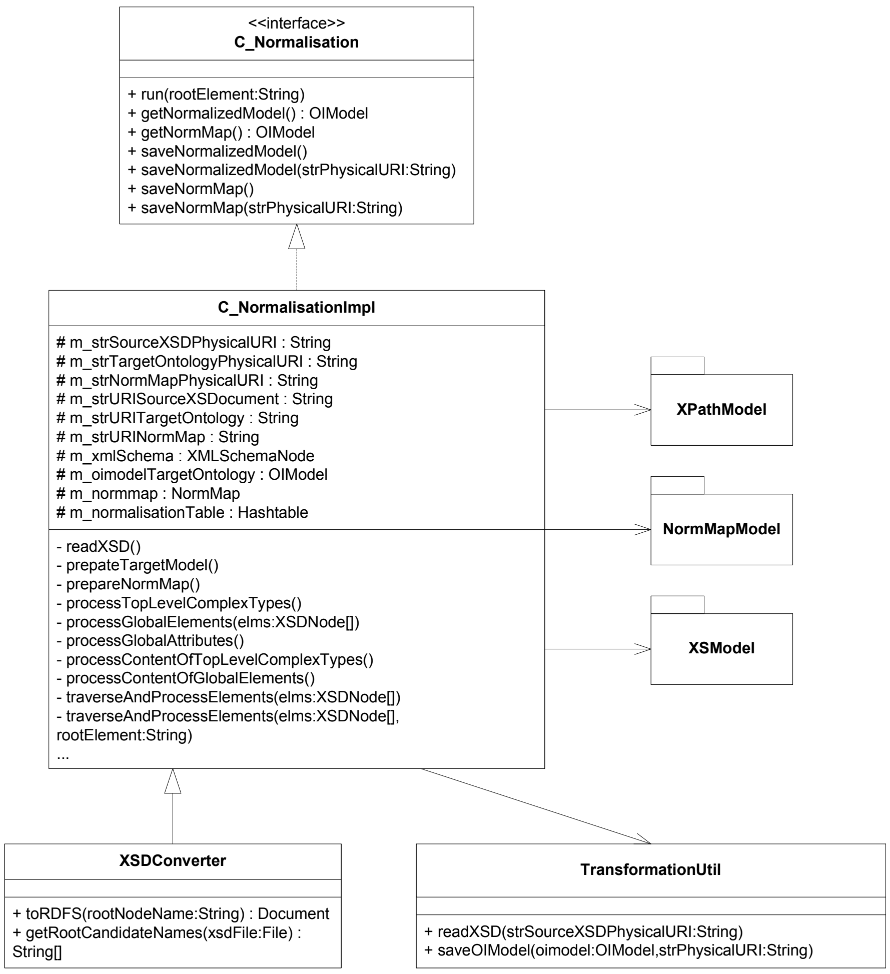

Define cross-joins between databases

advertiser
advdata
avrgposition
impressions
clicks
keyword
domain
date
advertiser
cost
identifier
handy_online
address_book
entry_firstname
entry_country_id

advertiser.advdata.advertiser -> handy_online.address_book.entry_firstname

no joined attributes

Figure 4.6: Database Import Wizard: Foreign Key Creation



<<interface>>
C_Normalisation
+ run(rootElement:String)
+ getNormalizedModel() : OIModel
+ getNormMap() : OIModel
+ saveNormalizedModel()
+ saveNormalizedModel(strPhysicalURI:String)
+ saveNormMap()
+ saveNormMap(strPhysicalURI:String)
C_NormalisationImpl
# m_strSourceXSDPhysicalURI : String
# m_strTargetOntologyPhysicalURI : String
# m_strNormMapPhysicalURI : String
# m_strURISourceXSDocument : String
# m_strURITargetOntology : String
# m_strURINormMap : String
# m_xmlSchema : XMLSchemaNode
# m_oimodelTargetOntology : OIModel
# m_normmap : NormMap
# m_normalisationTable : Hashtable
- readXSD()
- prepateTargetModel()
- prepareNormMap()
- processTopLevelComplexTypes()
- processGlobalElements(elms:XSDNode[])
- processGlobalAttributes()
- processContentOfTopLevelComplexTypes()
- processContentOfGlobalElements()
- traverseAndProcessElements(elms:XSDNode[])
- traverseAndProcessElements(elms:XSDNode[], rootElement:String)
...
XPathModel
NormMapModel
XSModel
XSDConverter
+ toRDFS(rootNodeName:String) : Document
+ getRootCandidateNames(xsdFile:File) : String[]
TransformationUtil
+ readXSD(strSourceXSDPhysicalURI:String)
+ saveOIModel(oimodel:OIModel,strPhysicalURI:String)

Figure 4.7: XML Schema Lifting - class diagram

CHAPTER 5

Evaluation

The lifting component for XML Schema and relational databases, as described in the previous chapter, has been integrated into the ontology mediation toolkit SEML. This chapter presents first results gathered from the application of the tool in a case study. Additional experiences and performance measurements collected from the described case study and other application areas are outlined as well. Therefore this chapter is organized as follows: The first section briefly describes the case study, including the setup for the prototype and the data sources and mappings used in the case study. The following section presents several results concerning performance, scalability and flexibility, as discovered during the implementation of the case study. Finally several additional application areas are mentioned, where the SEML toolkit was applied to specific integration needs. Experiences gathered from these projects are also briefly presented.

5.1 Case Study

Description

During the course of the SEML project, a marketing tool based on the technology developed in SEML was created. This tool, called the Advertising Tracking Tool (ATT), can be used to manage the profit margin of different advertising campaigns. Specifically, the tool can be used in the area of Search Engine Marketing (SEM), where the main advertising media are keywords (i.e., words or phrases related to one's business). Whenever such a keyword is entered in a search engine for which an advertisement contract exists, the results will include links to the advertised website in the sponsored listings. An example for this kind of advertising would be the Google AdWords[1] program. Using the ATT, the campaign manager can browse statistics for each of the keywords, such as the costs these keywords create, the conversion rate [2] and the revenue

[1] http://adwords.google.de

[2] The usual definition of conversion rate is the percentage of unique website visitors who take a desired action, such as placing an order.

generated. A set of distributed databases is queried by the tool to collect the necessary data, and thus an integration of these different data sources is required.

Figure 5.1 shows a use case diagram for the ATT, where “Evaluate Advertisement Statistics” is the main use case the actor (the tool user, probably an advertisement or product manager) is involved with. Additionally the actor is also aggregating different advertiser reports by using the tool’s import capabilities, which might also involve the need to fetch reports from advertiser portals, using Web services offered by the advertiser. An extension point for this use case is the possible lack of a suitable report API. In this case, the report data is usually available in CSV format, and the manually downloaded report files will be parsed by a component of the tool and imported in a separate relational database.

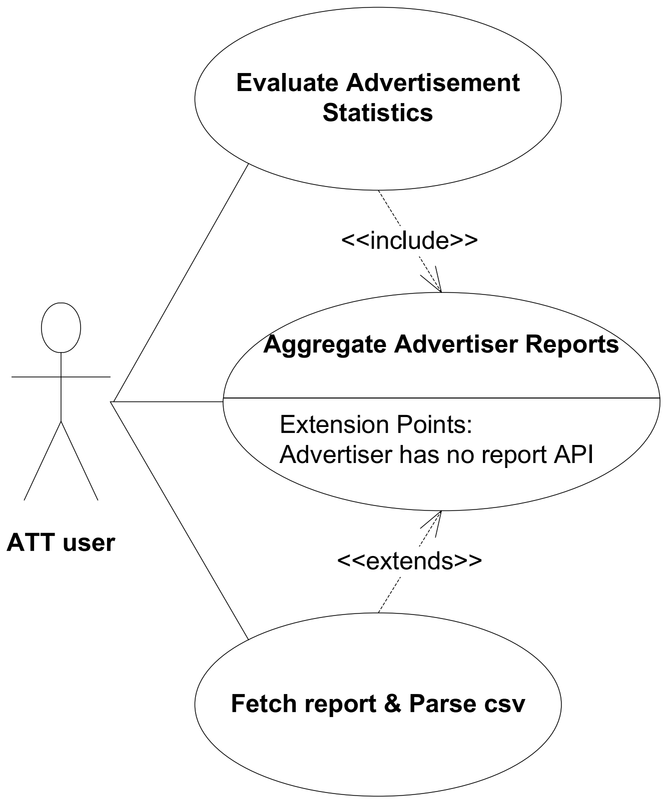

Evaluate Advertisement Statistics
<<include>>
Aggregate Advertiser Reports
Extension Points:
Advertiser has no report API
ATT user
<<extends>>
Fetch report & Parse csv

Figure 5.1: ATT Use Case Diagram

The data sources, the central ontology used by the tool and the corresponding mappings generated by the lifting component are presented in the following sections.

Data Sources

Data from three different, distributed databases was integrated for the first version of the ATT, as shown in Figure 5.2.

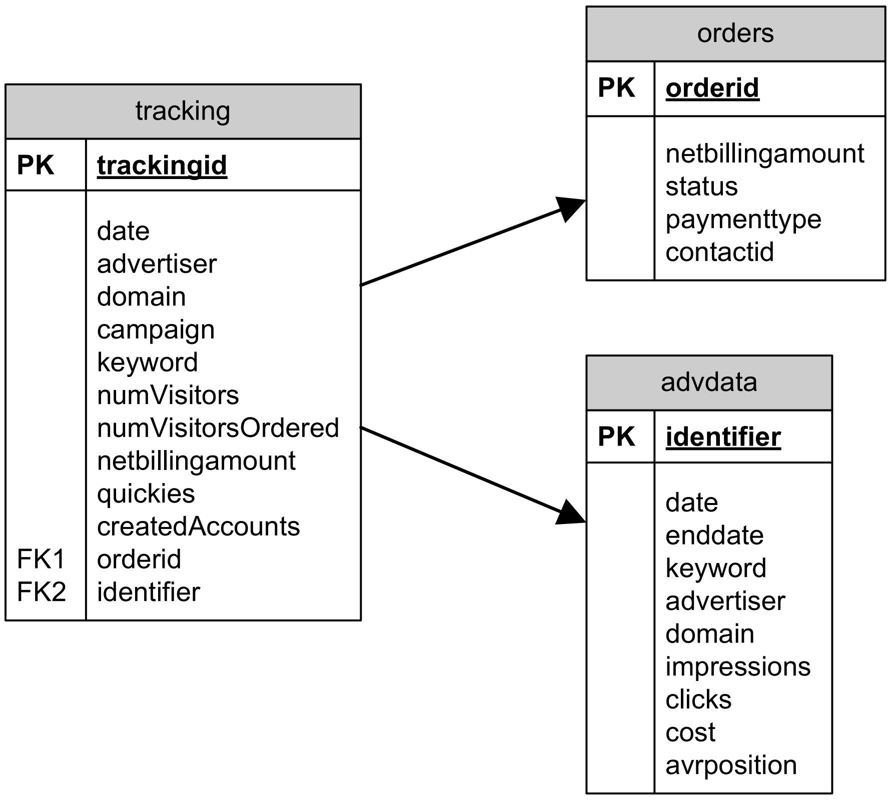


Figure 5.2: Data sources used for Advertising Tracking

The relations shown in the figure above come from different databases, used by a concrete e-Commerce solution from the case study: the “orders” relation is part of the main customer database in a dedicated customer care and billing system, the “tracking” relation represents part of the logging data collected by the webshop and the “advdata” relation correlates to the statistics gathered by the advertiser and parsed from advertisement reports. Table 5.1 further explicates the data sources as used for the prototype of the ATT, briefly explaining the meaning of the featured attributes.

Mappings

To aggregate the data from the three data sources introduced in the previous section, a virtual graph is created using the SEML Modeller. Thus, the components used for this include the database lifting, as well as the integration of multiple databases using the cross database foreign keys. As a result of this process a single (D2RQ) mapping document is produced, which contains all relevant mappings to the virtual graph.

A fragment of this mapping document is shown in Listing 5.1, which contains a single `ClassMap` for one of the databases and two corresponding attributes (for the keyword in question and the advertiser, respectively). A previous example from the mappings generated for this

Database (relation)	Attribute	Description
Advertiser (advdata)	identifier	Each keyword can be identified in an unique manner. For this, the date, the name of the advertiser and product specific data is aggregated.
	advertiser	Name of the advertiser (e.g., Google AdWords...)
	domain	The domain used for the advertisement (e.g., de, at, com)
	date	The date when the stored information was generated
	keyword	The exact keyword matched (e.g., “homepage”)
	clicks	On the specified date, how often a keyword has been clicked on
	cost	Costs for the keyword, for the specified date
	avrgposition	Average position an advertising text appears on a page, in comparison to competitors
Webshop (tracking)	identifier	Similar to the advertiser identifier
	orders	Advertisements for the keyword resulted in this number of actual orders
	orderid	Identifier for an order. Multiple orders are represented by multiple entries in this relation
Customer (orders)	orderid	Unique identifier of an order
	netbillingamount	Net amount a customer has been billed with, after ordering a product
	status	The order’s state (e.g., “paid”, “completed”)
	paymenttype	Type of payment selected for this order (e.g., credit card or bank transfer)

Table 5.1: Database Relations and Attributes for the ATT

case study was shown in 3.1 and used to highlight the concepts introduced in 3.1. The complete listing for the D2RQ document can be found in A.3.

```
adv:KeywordClassMap rdf:type d2rq:ClassMap;
            d2rq:class :Keyword;
            d2rq:uriPattern "http://www.niwa.at/statistik/concepts/Keyword@@advdata.
                identifier@@";
            d2rq:dataStorage adv:Database1;
            .

adv:Keyword rdf:type d2rq:DatatypePropertyBridge ;
            d2rq:property :keyword;
            d2rq:column "advdata.keyword" ;
            d2rq:belongsToClassMap adv:KeywordClassMap ;
            .

adv:Advertiser rdf:type d2rq:DatatypePropertyBridge ;
            d2rq:property :advertiser;
            d2rq:column "advdata.advertiser" ;
            d2rq:belongsToClassMap adv:KeywordClassMap ;
            .
```

Listing 5.1: D2RQ Fragment for the ATT

The generated virtual graph is used by the different functionalities of the ATT, in order to query the different data sources and combine the relevant results. Queries to the graph are issued by the tool in the RDF query language SPARQL [33] and forwarded to the query translation component. The translation process produces a set of SQL queries from this input, using the methods for query translation described in [26]. In order to optimize the performance, the set of SQL queries are then reordered using the query reformulation strategies described in the same work. The queries are then executed on the original databases, and the results are translated to RDF instances of the ontology described by the original graph.

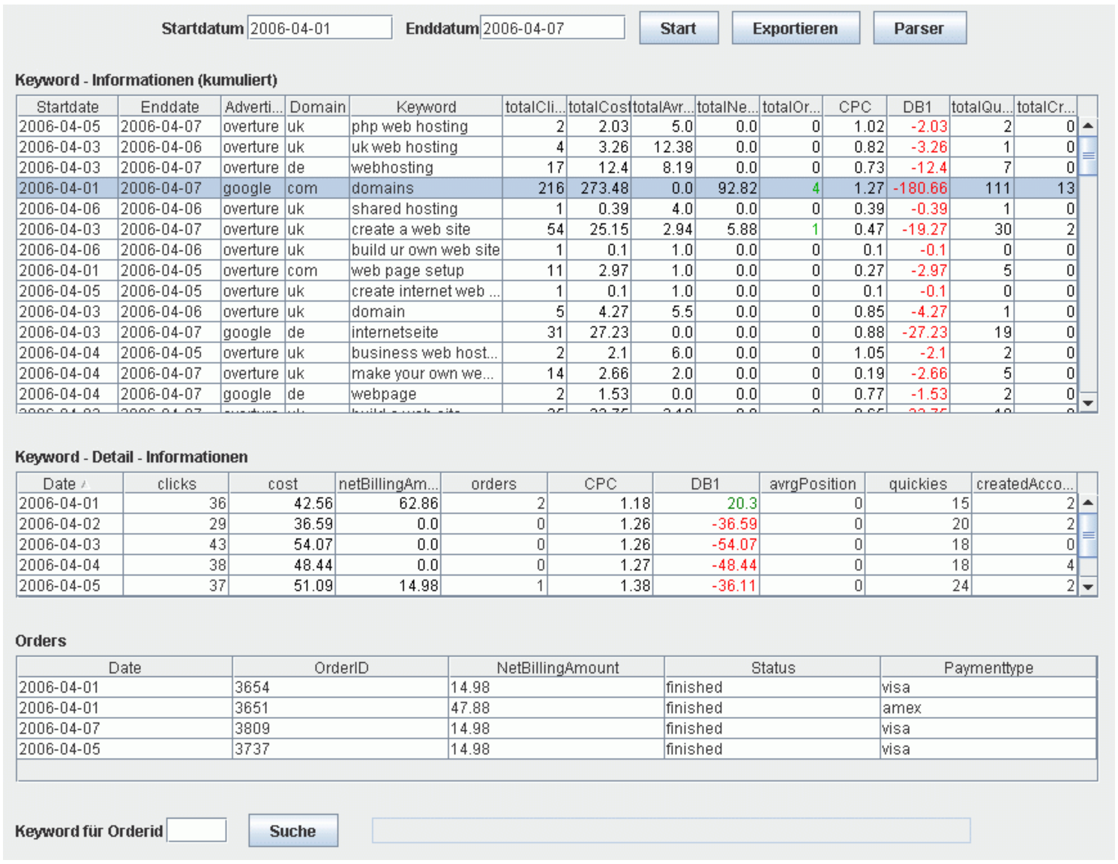
5.2 Results

The virtual graph created by the generated mappings can be queried by the tool to produce detailed statistical information. A sample screenshot is depicted in Figure 5.3 and shows the main view at the collected and aggregated data.

The tool itself has proven to be useful to give an overview of the effects of Search Engine Marketing. The main results of the prototype are:

- The process of integrating databases using the SEML Modeller is straight-forward, but the tool is still very much aligned to the view of an IT expert. Future development of the SEML toolkit will include a redesign of the Modeller, providing a Web accessible version for it. It is expected that an ASP-based version concentrating on a subset of the available functionalities, as needed for the integration of databases, will enlarge the possible target audience for the tool.

- More flexibility has been achieved, and the time to maintain the database integration has been reduced. Formerly hardcoded adapters had to be used for such a small scale project,


Startdatum 2006-04-01 Enddatum 2006-04-07 Start Exportieren Parser

Keyword - Informationen (kumuliert)

Startdate	Enddate	Adverti...	Domain	Keyword	totalCli...	totalCost	totalAvr...	totalNe...	totalOr...	CPC	DB1	totalQu...	totalCr...
2006-04-05	2006-04-07	overture	uk	php web hosting	2	2.03	5.0	0.0	0	1.02	-2.03	2	0
2006-04-03	2006-04-06	overture	uk	uk web hosting	4	3.26	12.38	0.0	0	0.82	-3.26	1	0
2006-04-03	2006-04-07	overture	de	webhosting	17	12.4	8.19	0.0	0	0.73	-12.4	7	0
2006-04-01	2006-04-07	google	com	domains	216	273.48	0.0	92.82	4	1.27	-180.66	111	13
2006-04-06	2006-04-06	overture	uk	shared hosting	1	0.39	4.0	0.0	0	0.39	-0.39	1	0
2006-04-03	2006-04-07	overture	uk	create a web site	54	25.15	2.94	5.88	1	0.47	-19.27	30	2
2006-04-06	2006-04-06	overture	uk	build ur own web site	1	0.1	1.0	0.0	0	0.1	-0.1	0	0
2006-04-01	2006-04-05	overture	com	web page setup	11	2.97	1.0	0.0	0	0.27	-2.97	5	0
2006-04-05	2006-04-05	overture	uk	create internet web ...	1	0.1	1.0	0.0	0	0.1	-0.1	0	0
2006-04-03	2006-04-06	overture	uk	domain	5	4.27	5.5	0.0	0	0.85	-4.27	1	0
2006-04-03	2006-04-07	google	de	internetseite	31	27.23	0.0	0.0	0	0.88	-27.23	19	0
2006-04-04	2006-04-05	overture	uk	business web host...	2	2.1	6.0	0.0	0	1.05	-2.1	2	0
2006-04-04	2006-04-07	overture	uk	make your own we...	14	2.66	2.0	0.0	0	0.19	-2.66	5	0
2006-04-04	2006-04-07	google	de	webpage	2	1.53	0.0	0.0	0	0.77	-1.53	2	0

Keyword - Detail - Informationen

Date	clicks	cost	netBillingAm...	orders	CPC	DB1	avrgPosition	quickies	createdAcco...
2006-04-01	36	42.56	62.86	2	1.18	20.3	0	15	2
2006-04-02	29	36.59	0.0	0	1.26	-36.59	0	20	2
2006-04-03	43	54.07	0.0	0	1.26	-54.07	0	18	0
2006-04-04	38	48.44	0.0	0	1.27	-48.44	0	18	4
2006-04-05	37	51.09	14.98	1	1.38	-36.11	0	24	2

Orders

Date	OrderID	NetBillingAmount	Status	Paymenttype
2006-04-01	3654	14.98	finished	visa
2006-04-01	3651	47.88	finished	amex
2006-04-07	3809	14.98	finished	visa
2006-04-05	3737	14.98	finished	visa

Keyword für Orderid Suche

Figure 5.3: Advertising Tracking Tool

since more powerful integration frameworks are not applicable at this scope. Attaching new databases or updating existing mappings is easily manageable with a new mapping project.

- Related to this, some form of versioning support was found to be lacking from the current release of the Modeller toolkit. Future releases will examine the idea of introducing support for versioned mapping documents. Different Ontology Mediation tools already include such functionalities and will be examined accordingly.

To provide a formal evaluation, the query translation based on the generated D2RQ mappings has been examined in a more thorough manner. The measurements conducted during this process included performance overviews of the query translation component, which is the main part determining the run-time usefulness of the toolkit and its components.

In order to perform queries to distributed databases in a scalable manner, a modified query handler was developed during the project. Figure 5.4 shows measurement results of a comparison of the performance of the standard query handler available with D2RQ with the query handler integrated with the SEML toolkit. On the left hand side a SPARQL query is shown, where the relations bound to the “person” and “address” concepts reside in different databases. The right hand side shows the amount of table entries in the databases in the first column. This is followed by the query execution time in milliseconds, first for the new distributed D2RQ query handler (dD2RQ), then for the standard query handler (SQH). As can be seen the new implementation improves upon the performance of the SQH, usually being up to ten to twenty times faster.

The thesis presented in [26] contains a lengthy examination of the performance of the different query handlers developed for the SEML project.

```
SELECT ?lastname ?prename ?info
       ?city ?postcode
WHERE {
  ?person  :lastname ?lastname .
  ?person  :prename  ?prename  .
  ?person  :info     ?info     .
  ?person  :adress  ?adress    .
  ?adress  :city      ?city    .
  ?adress  :postcode ’12345’   .
}
```

Database entries	dD2RQ performance in ms	SQH performance in ms
100	25	354
500	83	1402
1000	227	2723
2000	497	5311
5000	2233	13671
10000	13730	26568

Figure 5.4: SPARQL Query performance comparison

As a result of the prototypical implementation of the ATT, the tool itself is going to be part of a more fully-fledged e-Commerce suite currently developed at Hanival Internet Services. This e-Commerce suite will utilize SEML technology in several of its components, in different ways. Some of these other application areas are explained in the following section.

5.3 Other Application Areas

Two other projects which include SEML technology, specifically the legacy data source lifting designed and implemented as described in this thesis, are described in the following sections.

SemNetMan

In the nationally funded project “SemNetMan” (Semantically based Network Management) [3] we have used methods and concepts of information science and organizational studies, such as Social Network Analysis (SNA), as well as Semantic Web technologies to support network management, focussing on the team building phase. To facilitate the discovery of “suitable” partners for a team or project, different proximity measurement algorithms have been used. A detailed survey of the different technologies combined to achieve the project’s objectives can be found in [12].

The developed methods and tools have been implemented as a prototype, which is going to be applied and evaluated in three different case studies - the Semantic Web School [4], the

[3]http://www.semnetman.at

[4]http://www.semantic-web.at/main.php

Plattform Wissensmanagement [5] and the Österreichisches Biogasnetzwerk [6]. The following diagrams have been taken from the technical documentation for the project, which is not available to public at the time of this writing.

Figure 5.5 shows the complete architecture of the SemNetMan prototype, highlighting all components of the system. The diagram features the creation of a suitable instance base using the data source lifting technology developed in this work as the main enabling building block of the architecture. The SemNetMan methods then transform these models, resulting in a final graph which shows the proximity measurement results.

The central component integrating the lifting component is the so called Instance Base Generator. This component is accessed by the SemNetMan Web application via a Web service interface. The Instance Base Generator creates suitable instance data from a supplied database. These instances conform to the domain ontology created for each case study, like the example shown in Figure 5.6. Using the D2RQ mappings and the additional ontology mapping created manually with the SEML Modeller, either the complete database or selected parts are migrated to ontology instances.

This instance graph is then forwarded to other parts of the architecture - first the SemNetMan aggregator, which applies several modifiable transformation rules on the instance base and finally to the Weighting Service, which applies weights to the graphs's relations and supplies the final result to the Web application user. An example for such a result graph is shown in Figure 5.7.

E-Commerce Tool Suite

Semantic Web applications like the Smart Assistant by Smart Information Systems GmbH, analyze all semantically described products and offers to provide a user with the best matching results for his or her requirements[7]. The Smart Assistant is a so-called recommender system and can be integrated into the online shops of dealers. Currently, the RDF data describing these online offers has to be created semi-automatically by a stand-alone instantiation manager with the data itself hosted on a dedicated server. Further information pertaining to this issue can be found in “The realization of Semantic Web based E-Commerce and its impact on Business, Consumers and the Economy” [19].

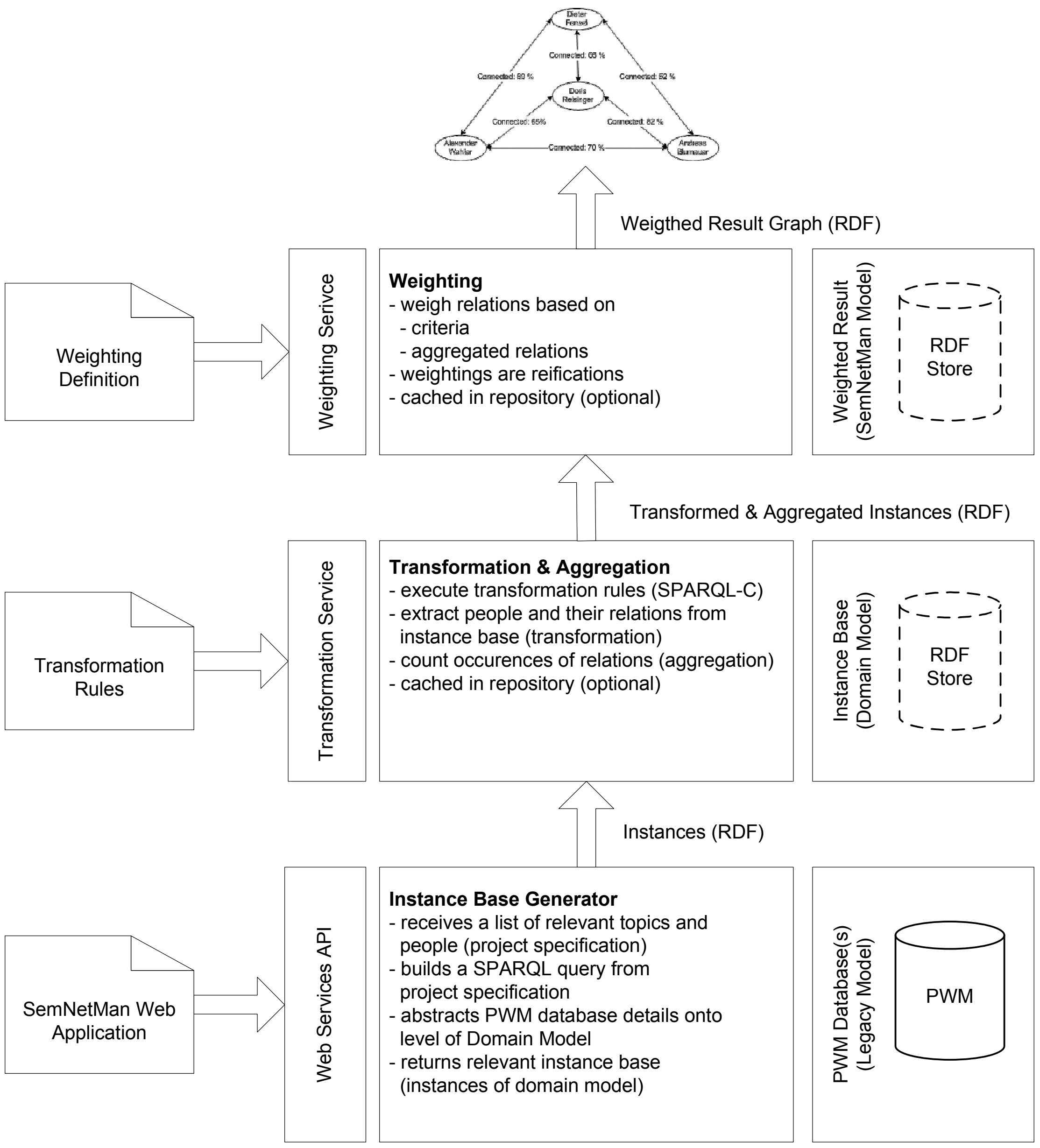
The shop owner will provide the RDF data compliant to ontologized versions of well-established product classification systems on his own website, parallel to the normal HTML content. One of the classification systems which has been used in the scope of this project was eCl@ass, a hierarchic system to establish groupings of materials, products and services. The system uses a logic schema to detail product specific attributes which lends itself well to ontologization, as conducted by Martin Hepp [23], with an OWL Lite version freely available online[8]. In addition, to prevent duplication of product data (i.e., technical details for the products), the offers are likely to be linked to more detailed descriptions available at the producer's or supplier's website. Advantages of using the eCl@ss approach to classify one's products include that shop

[5] `http://www.pwm.at`

[6] `http://www.oebn.at`

[7] A demo for the application is available at `http://www.smart-infosys.com/`

[8] available at `http://www.heppnetz.de/projects/eclassowl/`



Dieter Fensel
Connected: 89 %
Connected: 65 %
Connected: 52 %
Doris Reisinger
Connected: 65%
Connected: 82 %
Alexander Wahler
Connected: 70 %
Andreas Blumauer
Weigthed Result Graph (RDF)
Weighting Definition
Weighting Serivce
Weighting
- weigh relations based on
- criteria
- aggregated relations
- weightings are reifications
- cached in repository (optional)
Weighted Result (SemNetMan Model)
RDF Store
Transformed & Aggregated Instances (RDF)
Transformation Rules
Transformation Service
Transformation & Aggregation
- execute transformation rules (SPARQL-C)
- extract people and their relations from instance base (transformation)
- count occurences of relations (aggregation)
- cached in repository (optional)
Instance Base (Domain Model)
RDF Store
Instances (RDF)
SemNetMan Web Application
Web Services API
Instance Base Generator
- receives a list of relevant topics and people (project specification)
- builds a SPARQL query from project specification
- abstracts PWM database details onto level of Domain Model
- returns relevant instance base (instances of domain model)
PWM Database(s) (Legacy Model)
PWM

Figure 5.5: SemNetMan architecture

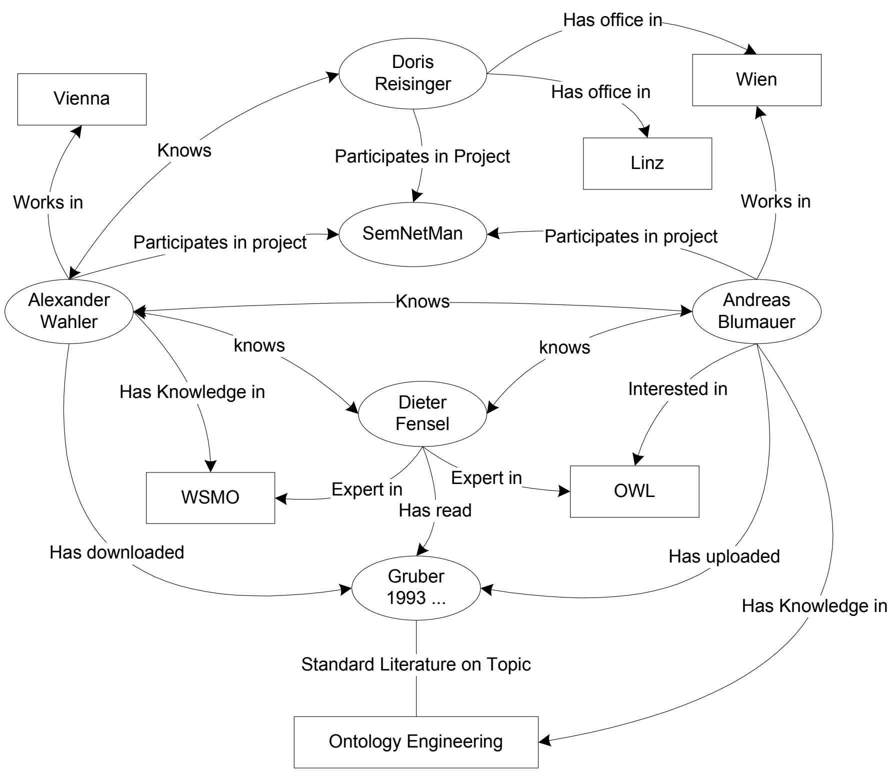


Figure 5.6: SemNetMan instance graph

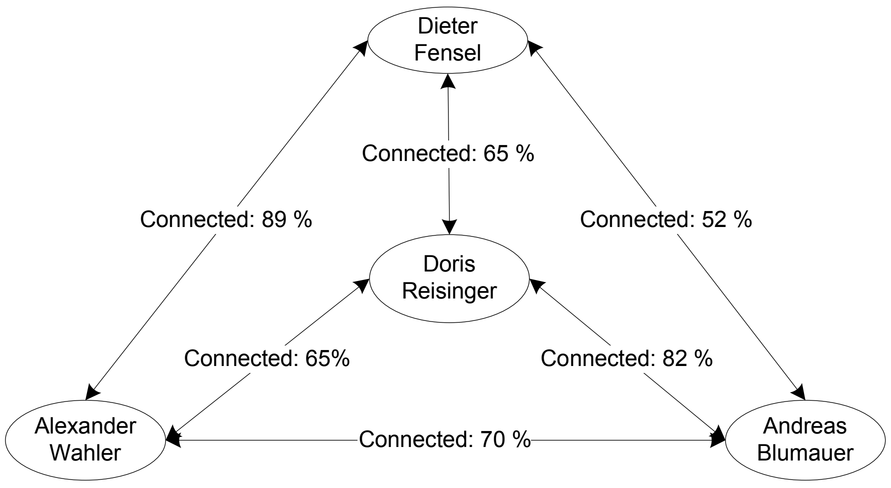


Figure 5.7: SemNetMan result graph

owners profit from standardization efforts, enabling better search results and facilitating interoperability with other suppliers and vendors.

Our paper on “Enabling Semantic Web-ready E-Commerce Solutions” [35] examines an e-Commerce platform, consisting of a well-known open source e-Commerce solution extended by the following functionalities:

- The Web shop owner can map his product catalogue to one of several applicable product or offer ontologies (currently an ontology based on eCl@ss is supported). The product data is imported from the relational database of the Web shop and instances of the product ontology are generated by executing the defined mapping rules.
- Run-time RDF generation is made available through a SPARQL endpoint, accessible on the Web shop owner’s Web space, which can be queried by different Semantic Web applications (e.g., the Smart Assistant, a Semantic Web browser or a dedicated SPARQL client application). This endpoint is realized by a D2R Server deployed in the shop owner’s Web space. Queries are translated by a service to SQL queries executed on the original RDBMS. Results are transformed to RDF instances (of the supported product ontology) and returned to the requester.
- Generated RDF instances are identified by suitable URIs based on patterns created by the mapping tool and compliant with the design issues described in [6]. Additionally, shop owners can link their offer descriptions to available product catalogues of suppliers (this is realized by either importing the additional information directly into the product descriptions or by adding an `rdfs:seeAlso` property to the generated instances).

Combining the approaches of ontologizing a product catalogue, according to a well specified product categorization system, with the capabilities of recommender systems leads to a flexible e-Commerce solution which directly aids smaller enterprizes by making their offers more visible. Also, providing such an RDF-based e-Commerce solution also adds to the objective of disseminating Semantic Web technology. Parts of the technology developed in this work will be directly applicable to the described scenario, by providing the means to create RDF-based product catalogues and mappings to existing product categorization ontologies.

CHAPTER 6

Related Work

This chapter surveys some of the related work in the field of lifting relational databases and XML Schema to ontologies. A commercial mediation solution is presented as well, and compared with the approach discussed in this work.

6.1 Direct Mapping and R2RML

Direct Mapping is a set of algorithms defining a transformation process, which takes the schema and data of a relational database as its input and automatically composes an RDF graph from this. This W3C Recommendation[1] of September 2012 provides a starting point for further, more complex transformations. RDF graphs can either be actually materialized, e.g., in order to be persisted in a dedicated triplestore, or defined as virtual.

Transformation Process

The following example for a transformation from a relational database to RDF highlights some of the most important aspects of the automatic transformation via Direct Mapping. Listing 6.1 shows the DDL for a sample database with two tables, each with a single-column primary key and one foreign key reference between them:

```
CREATE TABLE "Address" (
        "ID" INT, PRIMARY KEY("ID"),
        "city" CHAR(10),
        "street" CHAR(20),
        "pobox" CHAR(10)
)

CREATE TABLE "Person" (
        "ID" INT, PRIMARY KEY("ID"),
        "firstname" CHAR(10),
```

[1]available at http://www.w3.org/TR/rdb-direct-mapping/

```
        "address" INT,
        FOREIGN KEY("address") REFERENCES "Address"("ID")
)

INSERT INTO "Address" ("ID", "city", "street", "pobox") VALUES (2, 'Vienna', '
    Karlsplatz', '1040')
INSERT INTO "Person" ("ID", "firstname", "address") VALUES (3, 'Peter', 2)
INSERT INTO "Person" ("ID", "firstname", "address") VALUES (4, 'Maria', NULL)
```

Listing 6.1: Direct Mapping Example: SQL (DDL) Input

By providing a base IRI (Internationalized Resource Identifier), e.g. 'http://lift.example/DB/', Direct Mapping generates the RDF showin in Listing 6.2.

```
@base <http://lift.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<Person/ID=7> rdf:type <Person> .
<Person/ID=7> <Person#ID> 3 .
<Person/ID=7> <Person#firstname> "Peter" .
<Person/ID=7> <Person#address> 2 .
<Person/ID=7> <Person#ref-address> <Address/ID=2> .
<Person/ID=8> rdf:type <Person> .
<Person/ID=8> <Person#ID> 4 .
<Person/ID=8> <Person#firstname> "Maria" .

<Address/ID=2> rdf:type <Address> .
<Address/ID=2> <Address#ID> 2 .
<Address/ID=2> <Address#city> "Vienna" .
<Address/ID=2> <Address#street> "Karlsplatz" .
<Address/ID=2> <Address#pobox> "1030" .
```

Listing 6.2: Direct Mapping Example: RDF Result

For each row in the original database, a set of triples has been created, with each triple having the same subject. These subjects are formed by the concatenation of table name, primary key column name and the actual primary key value. Similarly, predicate IRIs have been created by concatenating table and column names. In addition, each foreign key reference is added as an additional triple, the object of which is the generated IRI for the referenced row in the referenced table. The transformation process can also handle composite keys, which are mapped as triples where the IRIs are constructed from all involved column names of the composite key. If no primary keys are defined for a table, the generated RDF introduces new blank nodes as the subjects of the triple set for each row.

As can be seen from the example above, the Direct Mapping process does not generate triples for the NULL values in the relational database. This is due to the problem of the use and semantics of NULL values in SQL as previously discussed in 2.4.

R2RML

R2RML[2] is a mapping language created as a companion to (and relaxed variant of) Direct Mapping. While the Direct Mapping transformation is meant to be an automatic process, users of R2RML can define custom mappings between relational databases and RDF. Mappings defined in R2RML are themselves valid RDF graphs. Every mapping in R2RML is meant for a specific database schema and an arbitrary target RDF vocabulary. While the target RDF vocabulary in Direct Mapping directly relates to the names of the elements in the database schema (as it is generated from these names), the R2RML allows users to create mappings utilizing existing target vocabularies, enabling broader use cases for the mapped data.

An R2RML mapping refers to a so called logical table, a term which can relate to either relational database tables, views or valid SQL queries. Furthermore, the mapping consists of one or more triples maps. A triples map is a rule that maps each row in a logical table to corresponding RDF triples. This rule consists of two main parts, a subject map and one or more predicate-object maps. A subject map is used to generate subjects, e.g. IRIs that are generated from the primary key column's values using a template specified in the mapping. Predicate-object maps consist of both predicate maps and object maps that are combined with a subject map to generate the output set of triples. Finally, triples maps can include graph maps, which assign certain generated triples to named output graphs. Without such graph maps, created triples are instead part of the same default output graph.

Listing 6.3 shows a partial R2RML mapping, which generates triples from the database table `PERSON`, which contains an `ID` and `LASTNAME` column. The RDF vocabulary, which is used in the R2RML mapping, contains an `Employee` entity, with a `name` attribute.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "PERSON" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{ID}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "LASTNAME" ];
    ].
```

Listing 6.3: Sample R2RML mapping

There are already first R2RML processors available, i.e., systems that, given an R2RML mapping and an input database, produce an output dataset in RDF. Examples for such processors include free tools such as Spyder[3] and commercial tools such as Ultrawrap[4]. These tools differ in their support for various database managements systems, the version of SPARQL that is supported and additional features for their use in enterprise scale applications.

[2]available at `http://www.w3.org/TR/r2rml/`

[3]available at `http://www.revelytix.com/content/spyder`

[4]available at `http://www.capsenta.com/product.html`

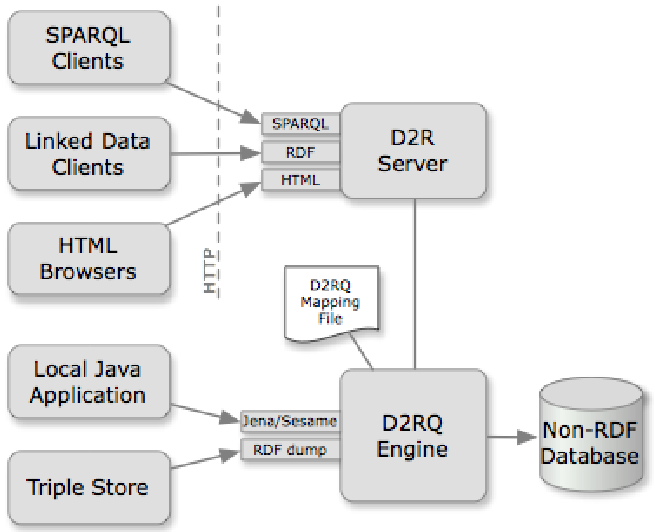


Figure 6.1: D2R Server Architecture [8]

6.2 D2R Server: Publishing Relational Databases on the Semantic Web

D2R Server [10] provides SPARQL access to relational databases and thus supports the goal to publish relational data on the Semantic Web. It automatically captures the relation between a legacy database schema and an RDF vocabulary by generating suitable D2RQ mappings. The RDF vocabulary used for this purpose is generated as well.

D2R Server publishes relational databases to the Semantic Web in both RDF and XHTML form. Thus the data can be browsed by either a specialized Semantic Web data browser or a common HTML browser. In addition D2R Server features SPAQRL endpoints for the relational database, in order for Web applications to directly query the database. This feature is especially useful for large scale databases which would not perform if every single object was made directly accessible on the Web. Figure 6.1 shows the general usage scenario for D2R Server.

Two different interfaces support these functionalities, the “dereferencing interface” and the “SPARQL interface”. When publishing data on the Web, D2R Server links resources by providing support for hyperlink navigation both on the RDF and XHTML level, according to the expectations addressed in the design note on linked data [6] by Tim Berners-Lee. In addition, the server allows the generated URIs denoting resources to be dereferenced. RDF representations of resources are sent to any requesting party sending a HTTP GET request to the server.

The SPARQL interface on the other hand takes SPARQL queries from the Web and rewrites them via a D2RQ mapping into SQL queries against a relational database. This means that RDF based applications can query legacy data sources at run-time, without the need to migrate large datasets to RDF. D2R Server can be used to integrate existing databases into RDF systems, and to add SPARQL interfaces to database-backed software products.

The current prototype[5] works with Oracle, MySQL, PostgreSQL, SQL Server, HSQLDB, Interbase/Firebird and any SQL-92 compatible database. D2R Server combines the D2RQ API and Joseki SPARQL Server to map a database's contents into a virtual RDF graph that can be accessed over the SPARQL protocol. Since version 0.8.1, D2R Server provides support for W3C's Direct Mapping described previously.

D2R Server includes some functionalities similar to the integrated mediation toolkit described in this thesis. Specifically the query translation component (from SPARQL to SQL), allowing on-the-fly querying of legacy databases, has also been included in the toolkit described in Chapter 3, but a comparison of these components is not within the scope of this thesis. In addition the generation of D2RQ mapping rules is possible with a tool available with D2R Server, providing the same functionality as the mapping generation tool designed in Section 3.1 of this work.

D2R Server, while offering a similar set of functionalities as the integrated mediation toolkit, is designed for the use case of query translation, specifically for querying relational databases. In addition to these functionalities, the integrated mediation toolkit is envisioned as also providing support for the instance migration use case and access to other data sources, such as XML schema.

6.3 R_2O

R_2O [3] is an extensible and declarative language with which it is possible to describe complex mappings between relational database schemas and ontologies implemented in RDF(S) or OWL. The main application area of the language is the extraction and transformation of database content into instances of an ontology. The DB and ontology models can be different in the scope of their covered domains and both models have to pre-exist and are not created specifically for the purpose of instance migration or query translation. The language allows for the definition of mappings, stored in a single document. This set of mappings then has to be interpreted by a mapping processor, in a manner similar to the D2RQ approach.

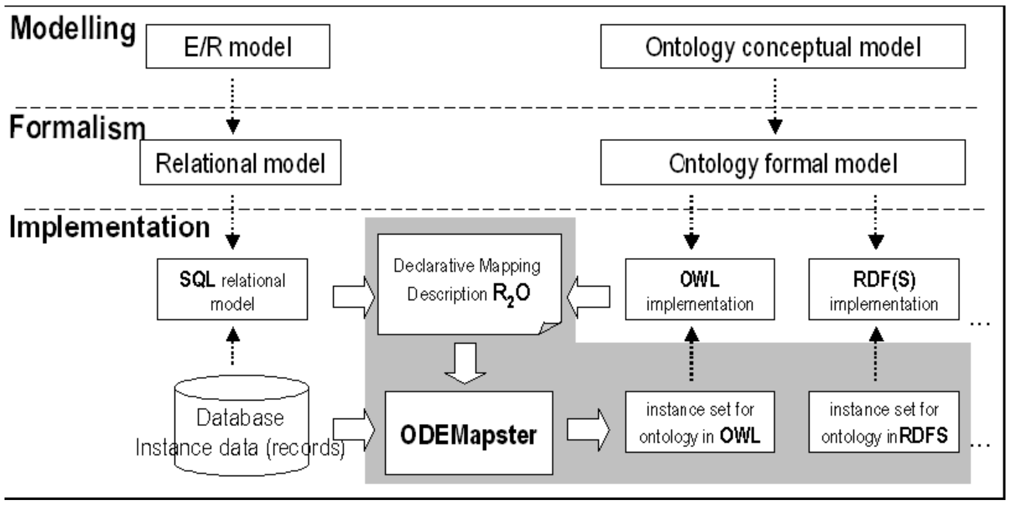
Some main features of the language R_2O are:

- An R_2O mapping document enables the automatic population of an ontology with instances extracted from a database. Both instance migration and instance creation on demand - by run-time query translation - is supported by the mappings.

- R_2O mappings allow for a characterization of data sources to enable a dynamic query distribution in intelligent information integration approaches.

R_2O provides an extensible set of primitives with well defined semantics. This language has been conceived expressive enough to cope with complex mapping cases arising from situations of low similarity between the ontology and the DB model(s). Figure 6.2 shows the overall R_2O mapping architecture, which has been taken from [3].

[5]available at http://d2rq.org/d2r-server



Modelling
E/R model
Ontology conceptual model
Formalism
Relational model
Cntology formal model
Implementation
SQL relational model
Declarative Mapping Description R_2O
OWL implementation
RDF(S) implementation
...
Database Instance data (records)
ODEMapster
instance set for ontology in OWL
instance set for ontology in RDFS
...

Figure 6.2: R_2O mapping architecture

In general, mappings expressed by R_2O are to be generated and processed by mediation middleware and tools, such as the ODEMapster mapping processor, as featured at a poster presentation at the WWW2006 conference [34].

Some of the differences with the previously examined D2RQ mapping language include:

- The ontology, as well as the database schema, has to pre-exist.
- Mappings are "one-way", which means they only support the mediation of data from the database to the ontology.
- R_2O mappings are to be generated manually, in order to exploit the high expressivity of the language, with which it is possible to describe many different, complex mapping cases. D2RQ language constructs - with their focus on more straight forward mappings - are better suited to an automated generation mechanism.

Therefore R_2O provides a solution for the complete mediation scenario between an arbitrary ontology and a database schema, in the same way as the integrated mediation framework envisioned in Section 2.3 does. As it only concerns itself with relational databases (and only provides a one-way mapping to ontologies) the language is deemed unsuitable in the context of a mediation tool for different, heterogeneous data sources. Still, the complex mapping situations covered by the language are of interest to a further development of the relational lifting, where user-defined preferences could steer the default lifting mechanisms. This idea is further examined in Section 7.2.

6.4 Other Related Work

Other approaches have appeared in literature, which - in one way or another - concern themselves with the problem of reusing existing data in an ontology-centric environment. Most are interesting from an academic viewpoint, but never evolved from a prototyping stage. Still, several of these approaches are mentioned in this section, and are compared to the work conducted in this thesis.

Lifting Metamodels to Ontologies

In this work [25], the authors tackle the problem of mediating between different modeling languages used in software development, by lifting metamodels to an ontological layer. The approach described herein is similar to the ontology learning process - more exhaustively described in [28]: After extracting a first version of an ontology from a give metamodel, the resulting ontologies are refined to better suit their purpose - using ontology mappings for deriving implementation specific transformation code between modeling languages. The conducted refinements are called ontology refactoring and ontology enrichment in the context of this paper. While the use case described for this lifting process is entirely different to the use cases envisioned for the lifting component in SEML, the actual lifting and refinement patterns could be applied to future work in order to enhance the quality of lifted ontologies.

A Method for Transforming Relational Schemas Into Conceptual Schemas

While not part of the current developments of lifting legacy schemata to an ontological level and actually preceding the inception of the Semantic Web, this work [24] still presents valid concepts applicable to current lifting strategies. The authors describe a method to translate a schema in the relational model into a schema in a conceptual model, thus enabling the conceptual understanding of the data.

The conceptual model used for this method is a formalization of an extended ER model, which has been expanded by the inclusion of a subtype concept. The authors introduce four basic schema transformation methods, based on which a comprehensive method for the translation of schemas is explicated. A prototypical implementation of this method in Prolog is mentioned, simplifying an expert system approach to the subject.

The authors observe that methods to produce conceptual schemas without any user interaction can only cover a small part of the semantic information implicitly hidden in a relational schema. An extensive interaction with the user is deemed mandatory for this kind of schema translation. While this is a sensible observation for the scope of schema translation, in the work conducted in this thesis, the user interaction occurs on the level of ontology to ontology mediation, enabling the automation of the database lifting phase. Furthermore the authors state that an advantage of their research is the use of simple and well established concepts of relational database theory in their work, including a strict formalization of both the relational and the conceptual schema. This approach has been taken up and certain parts of a formalized foundation for the lifting of legacy data sources have been introduced in the previous chapters.

Relational.OWL

In a comparable approach to other database lifting techniques utilizing a declarative language, the authors of this paper [32] motivate their use of the OWL Full dialect of the Web Ontology

Language to describe both schema and data information of relational databases.

The major difference to the approach of D2RQ is the focus on gaining complete application independency by describing relational schemas in an abstract way. The “Relational.OWL” Ontology suggested by the authors introduces elements which allow the representation of arbitrary relational database elements. Thus the language is deemed appropriate to serve as a universal exchange format for both the database schema itself and the data represented in this schema.

This kind of approach to data and scheme presentation is seen to be especially suitable to solve the challenges inherent to Peer-to-Peer (P2P) databases. Among other reasons, the possible short availability of volatile peers in such environments prevents the use of dedicated transformation components. Explicit exchange formats in (semi-)structured data models, such as XML Schema, are equally inappropriate due to the large number of possible peers interoperating.

The ideas introduced by the authors are suitable for further consideration, but would need to be extended for the use cases envisioned in the scope of this thesis, specifically for the query rewriting use case.

CHAPTER 7

Conclusions

7.1 Summary

This thesis has examined the problem of data mediation, utilizing semantic technology to solve the problem of data heterogeneity existing between different legacy data sources. A detailed state of the art survey of different mediation frameworks and data source lifting methods was conducted. Among the frameworks examined, the Mapping Framework (MAFRA) developed by the FZI Karlsruhe was chosen as one of the building blocks of the SEML Modeller, a tool to integrate diverses heterogeneous data sources via ontology mediation. The lifting component therefore had to support this framework by producing suitable ontologies and instance bases from the underlying data sources. For relational databases, the D2RQ approach by the Free University of Berlin was chosen, a declarative language to describe mappings between relational database schemas and OWL/RDFS ontologies. For the lifting of XML data, the Normalization Toolkit, part of the comprehensive MAFRA implementation hMAFRA, was selected.

Following this examination, the design for a lifting component was presented, which focuses on relational databases and XML Schema documents as the supported legacy data sources. Among the problems which came up during this design, the main issue was the support for the joining of multiple databases to a single virtual RDF graph, which can then be used as the source of further ontology mediation steps made possible by the SEML Modeller.

An implementation of this lifting component was conducted in the scope of this work, and the resulting component was integrated with the SEML Modeller. This toolkit is the result of a project concerned with ontology mediation, and has been used in several prototypes and case studies. One such case study, the Advertising Tracking Tool was detailed in this work, with several other application scenarios described as well.

An evaluation of the lifting component was conducted in conjunction with the other parts which make up the SEML Modeller toolkit. As mentioned, the tool was employed in several projects and case studies, where it could be shown that the methods to integrate a new data source with others – via ontology mediation – is a flexible and lightweight solution. Mappings can be easily created and refined to reflect changes to the original databases. The design-time instance base generation is straightforward and allows applications based on Semantic Web technology to utilize the transformed data. The performance of the run-time query translation solution was examined as well. A new distributed query handler was exchanged for the one supplied with D2RQ, to optimize query performance. Still, for large scale databases, querying the data at run-time needs further optimization work.

7.2 Future Work

Future work related to the lifting component, and the SEML project in general will be conducted, including at least the following main issues:

- Besides RDFS, other ontology representation languages, such as OWL DL, are going to be supported by the SEML Modeller (and therefore also have to be supported by the lifting component). This development step is necessary to use Ontologies such as eCl@ss OWL as source or target of mediation projects. This also has the benefit that new mapping rules become suitable during the lifting process. As an example OWL's cardinality constraints can be mapped directly to related XML Schema elements.

- As explained in Section 6.3, it is planned to extend the mapping generation with flexible rules which can be configured by the tool user. Depending on the actual application scenario, this will enable the tool to produce better suited ontologies from the legacy data sources. In this way domain specific knowledge hidden in the original data models could be expressed in the ontologies in a direct way.

- Additional data sources are going to be supported in the future. Besides other semi-structured documents, such as log files, we will also look at Web Services as potential sources for data to be lifted to the ontological level. WSDL descriptions contain sections which determine the structure of messages, and consist of XML Schema fragments to constrain data types and message formats. This is would enable an application developer to utilize the SEML Modeller to connect to third parties' data sources, whenever such parties would not provide access to their original data, but instead offer web services.

- Another future development will be the further integration of SEML technology with the comprehensive e-Commerce tool suite to be developed by Hanival Internet Services. The e-Commerce tool was briefly mentioned in Chapter 5, and will form the basis of the first product, with SEML technology, which will be shipped to customers.

APPENDIX A

Codesamples

A.1 Sample XSD

The following Listing shows an XML Schema defining regional events for the Finnish Tourist Board (MEK). This simple schema describes one main element (EVENT) which consists of a number of child elements in a sequence.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <xsd:element name = "EVENT">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name = "ProdUniqueID" type = "xsd:string"/>
        <xsd:element name = "ProdLastModified_D" type = "xsd:string"/>
        <xsd:element name = "ProdLastModified_M" type = "xsd:string"/>
        <xsd:element name = "ProdLastModified_Y" type = "xsd:string"/>
        <xsd:element name = "BookingWWW" type = "xsd:string"/>
        <xsd:element name = "BookingFaxNumber" type = "xsd:string"/>
        <xsd:element name = "BookingFaxPrefix" type = "xsd:string"/>
        <xsd:element name = "BookingPostOffice" type = "xsd:string"/>
        <xsd:element name = "BookingEMail" type = "xsd:string"/>
        <xsd:element name = "BookingPhoneNumber" type = "xsd:string"/>
        <xsd:element name = "BookingPhonePrefix" type = "xsd:string"/>
        <xsd:element name = "BookingPostNumber" type = "xsd:string"/>
        <xsd:element name = "BookingPostCodePrefix" type = "xsd:string"/>
        <xsd:element name = "BookingStreetAddress" type = "xsd:string"/>
        <xsd:element name = "BookingName_FIN" type = "xsd:string"/>
        <xsd:element name = "BookingName" type = "xsd:string"/>
        <xsd:element name = "SupplierWWW" type = "xsd:string"/>
        <xsd:element name = "SupplierFaxNumber" type = "xsd:string"/>
        <xsd:element name = "SupplierFaxPrefix" type = "xsd:string"/>
        <xsd:element name = "SupplierPostOffice" type = "xsd:string"/>
        <xsd:element name = "SupplierEMail" type = "xsd:string"/>
        <xsd:element name = "SupplierPhoneNumber" type = "xsd:string"/>
        <xsd:element name = "SupplierPhonePrefix" type = "xsd:string"/>
        <xsd:element name = "SupplierPostNumber" type = "xsd:string"/>
        <xsd:element name = "SupplierPostCodePrefix" type = "xsd:string"/>
        <xsd:element name = "SupplierStreetAddress" type = "xsd:string"/>
        <xsd:element name = "SupplierName_FIN" type = "xsd:string"/>
        <xsd:element name = "SupplierName" type = "xsd:string"/>
        <xsd:element name = "LocMajorRegion" type = "xsd:string"/>
        <xsd:element name = "LocRegion" type = "xsd:string"/>
        <xsd:element name = "LocCity" type = "xsd:string"/>
        <xsd:element name = "Price" type = "xsd:string"/>
        <xsd:element name = "LocAPTcode" type = "xsd:string"/>
        <xsd:element name = "EventPlace" type = "xsd:string"/>
        <xsd:element name = "EndingDate_D" type = "xsd:string"/>
        <xsd:element name = "EndingDate_M" type = "xsd:string"/>
        <xsd:element name = "EndingDate_Y" type = "xsd:string"/>
        <xsd:element name = "StartingDate_D" type = "xsd:string"/>
        <xsd:element name = "StartingDate_M" type = "xsd:string"/>
        <xsd:element name = "StartingDate_Y" type = "xsd:string"/>
        <xsd:element name = "EventExtent" type = "xsd:string"/>
        <xsd:element name = "FinFestival" type = "xsd:string"/>
        <xsd:element name = "EventType" type = "xsd:string" maxOccurs = "unbounded"/>
        <xsd:element name = "EventURL" type = "xsd:string"/>
        <xsd:element name = "ProdDescription" type = "xsd:string"/>
        <xsd:element name = "ProdName_FIN" type = "xsd:string"/>
        <xsd:element name = "ProdName" type = "xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing A.1: XML Schema for MEK Events

A.2 Sample Ontology

Two ontologies generated by the lifting component are listed in this section. The first of these ontologies has been lifted from the XML Schema shown in the previous section, and represents the ontologized version of the MEK events.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>

<rdf:RDF xmlns="http://www.hanival.net/MEK#"
    xml:base="http://www.hanival.net/MEK"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Property rdf:ID="BookingEMail">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingFaxNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingFaxPrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingName">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingName_FIN">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingPhoneNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingPhonePrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingPostCodePrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingPostNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingPostOffice">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingStreetAddress">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="BookingWWW">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EVENT">
    <rdfs:range rdf:resource="#EVENT_class"/>
</rdf:Property>
<rdfs:Class rdf:ID="EVENT_class">
    <rdfs:subClassOf
        rdf:resource="http://kaon.semanticweb.org/2001/11/kaon-lexical#Root"/>
</rdfs:Class>
<rdf:Property rdf:ID="EndingDate_D">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EndingDate_M">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EndingDate_Y">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EventExtent">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EventPlace">
```

```
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EventType">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="EventURL">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="FinFestival">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="LocAPTcode">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="LocCity">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="LocMajorRegion">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="LocRegion">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="Price">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdDescription">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdLastModified_D">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdLastModified_M">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdLastModified_Y">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdName">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdName_FIN">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="ProdUniqueID">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
```

```
</rdf:Property>
<rdf:Property rdf:ID="StartingDate_D">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="StartingDate_M">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="StartingDate_Y">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierEMail">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierFaxNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierFaxPrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierName">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierName_FIN">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierPhoneNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierPhonePrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierPostCodePrefix">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierPostNumber">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierPostOffice">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierStreetAddress">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="SupplierWWW">
    <rdfs:domain rdf:resource="#EVENT_class"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
```

```
</rdf:RDF>
```

Listing A.2: RDFS Ontology for MEK Events

The second ontology is the one used for the case study described in Chapter 5. The Advertising Tracking Tool ontology has been generated from three distinct databases, representing data collected from the advertiser, tracking information from the webshop and actual order information. More details on the underlying data sources can be found in the description of the ATT. The accompanying D2RQ Map for this ontology is shown in the next section.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.hanival.net/advertising#"
     xml:base="http://www.hanival.net/advertising"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Property rdf:ID="advertiser">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;string"/>
        <rdfs:comment rdf:datatype="&xsd;string">The advertiser hosting the ad for
            this keyword (e.g. Google AdWords, Overture)</rdfs:comment>
    </rdf:Property>
    <rdf:Property rdf:ID="avrgposition">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;int"/>
        <rdfs:comment rdf:datatype="&xsd;string">the average position of one&apos;s
            advertisement found when searching for the given keyword</rdfs:comment>
    </rdf:Property>
    <rdf:Property rdf:ID="clicks">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </rdf:Property>
    <rdf:Property rdf:ID="cost">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </rdf:Property>
    <rdf:Property rdf:ID="date">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;date"/>
    </rdf:Property>
    <rdf:Property rdf:ID="domain">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </rdf:Property>
    <rdf:Property rdf:ID="hasKeyword">
        <rdfs:domain rdf:resource="#Webshop_Tracking"/>
        <rdfs:range rdf:resource="#TrackingToKeywordDummy"/>
    </rdf:Property>
    <rdf:Property rdf:ID="hasOrder">
        <rdfs:domain rdf:resource="#Webshop_Tracking"/>
        <rdfs:range rdf:resource="#TrackingOrderDummy"/>
    </rdf:Property>
    <rdf:Property rdf:ID="identifier">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </rdf:Property>
    <rdf:Property rdf:ID="impressions">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </rdf:Property>
    <rdf:Property rdf:ID="keyword">
        <rdfs:domain rdf:resource="#Keyword"/>
        <rdfs:range rdf:resource="&xsd;string"/>
        <rdfs:comment rdf:datatype="&xsd;string">The actual string representation of
            the keyword itself</rdfs:comment>
    </rdf:Property>
    <rdf:Property rdf:ID="nrOfOrders">
        <rdfs:domain rdf:resource="#Webshop_Tracking"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </rdf:Property>
    <rdf:Property rdf:ID="orderNetBillingAmount">
        <rdfs:domain rdf:resource="#Order"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </rdf:Property>
    <rdf:Property rdf:ID="orderPaymenttype">
        <rdfs:domain rdf:resource="#Order"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </rdf:Property>
    <rdf:Property rdf:ID="orderStatus">
        <rdfs:domain rdf:resource="#Order"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </rdf:Property>
    <rdfs:Class rdf:ID="Keyword"/>
    <rdfs:Class rdf:ID="Webshop_Tracking"/>
    <rdfs:Class rdf:ID="Order"/>
    <rdfs:Class rdf:ID="TrackingOrderDummy"/>
    <rdfs:Class rdf:ID="TrackingToKeywordDummy"/>
</rdf:RDF>
```

Listing A.3: RDFS Ontology for the ATT

A.3 Sample D2RQ Map

The listing below shows the D2RQ map which was generated for the Advertising Tracking Tool, as detailed in Chapter 5. In order to present an uniform view on the different artifacts created by the lifting component, the listing is shown in RDF/XML notation. The transformation from the generated version in Notation 3 (N3) was performed with a freely available RDF converter[1].

```
<rdf:RDF xmlns="http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#"
    xmlns:adv="http://www.niwa.at/statistik/advertiser#"
    xmlns:customer="http://www.niwa.at/statistik/customer#"
    xmlns:d2rq="http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dctype="http://purl.org/dc/dcmitype/"
    xmlns:log="http://www.w3.org/2000/10/swap/log#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
```

[1]available at http://www.mindswap.org/2002/rdfconvert/

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:webshop="http://www.niwa.at/statistik/webshop#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<ProcessingInstructions
    rdf:about="http://www.niwa.at/statistik#ProcessingInstructions1">
    <queryHandler>de.fuberlin.wiwiss.d2rq.rdql.D2RQQueryHandler </queryHandler>
</ProcessingInstructions>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/advertiser#Advertiser">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.advertiser</column>
    <property rdf:resource="http://www.niwa.at/statistik#advertiser"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/advertiser#AvrgPosition">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.avrgposition</column>
    <property rdf:resource="http://www.niwa.at/statistik#avrgPosition"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/advertiser#Clicks">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.clicks</column>
    <property rdf:resource="http://www.niwa.at/statistik#clicks"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/advertiser#Cost">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.cost</column>
    <property rdf:resource="http://www.niwa.at/statistik#cost"/>
</DatatypePropertyBridge>

<Database rdf:about="http://www.niwa.at/statistik/advertiser#Database1">
    <allowDistinct>false</allowDistinct>
    <dateColumn>advdata.enddate</dateColumn>
    <jdbcDSN>jdbc:postgresql://andastra.niwa.at/advertiser?user=jboss </jdbcDSN>
    <jdbcDriver>org.postgresql.Driver</jdbcDriver>
    <numericColumn>advdata.avrgposition</numericColumn>
    <numericColumn>advdata.clicks</numericColumn>
    <numericColumn>advdata.cost</numericColumn>
    <numericColumn>advdata.identifier</numericColumn>
    <numericColumn>advdata.impressions</numericColumn>
    <textColumn>advdata.advertiser</textColumn>
    <textColumn>advdata.date</textColumn>
    <textColumn>advdata.domain</textColumn>
    <textColumn>advdata.keyword</textColumn>
</Database>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/advertiser#Domain">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.domain</column>
    <property rdf:resource="http://www.niwa.at/statistik#domain"/>
</DatatypePropertyBridge>
```

```
<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/advertiser#Impressions">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.impressions</column>
    <property rdf:resource="http://www.niwa.at/statistik#impressions"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/advertiser#Keyword">
    <belongsToClassMap rdf:resource=
        "http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.keyword</column>
    <property rdf:resource="http://www.niwa.at/statistik#keyword"/>
</DatatypePropertyBridge>

<ClassMap rdf:about="http://www.niwa.at/statistik/advertiser#KeywordClassMap">
    <class rdf:resource="http://www.niwa.at/statistik#Keyword"/>
    <dataStorage rdf:resource="http://www.niwa.at/statistik/advertiser#Database1"/>
    <uriPattern>http://www.niwa.at/statistik/concepts/
    Keyword@@advdata.identifier@@</uriPattern>
</ClassMap>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/advertiser#Startdate">
    <belongsToClassMap
        rdf:resource="http://www.niwa.at/statistik/advertiser#KeywordClassMap"/>
    <column>advdata.date</column>
    <property rdf:resource="http://www.niwa.at/statistik#date"/>
</DatatypePropertyBridge>

<Database rdf:about="http://www.niwa.at/statistik/customer#Database3">
    <jdbcDSN>jdbc:postgresql://andastra.niwa.at/orderData?user=jboss</jdbcDSN>
    <jdbcDriver>org.postgresql.Driver</jdbcDriver>
    <numericColumn>t_order.netbillingamount</numericColumn>
    <numericColumn>t_order.orderid</numericColumn>
    <textColumn>t_order.contactid</textColumn>
    <textColumn>t_order.status</textColumn>
</Database>

<ClassMap rdf:about="http://www.niwa.at/statistik/customer#OrderClassMap">
    <class rdf:resource="http://www.niwa.at/statistik#Order"/>
    <dataStorage rdf:resource="http://www.niwa.at/statistik/customer#Database3"/>
    <uriPattern>http://www.niwa.at/statistik/concepts/
    Order@@t_order.orderid@@</uriPattern>
</ClassMap>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/customer#OrderId">
    <belongsToClassMap
        rdf:resource="http://www.niwa.at/statistik/customer#OrderClassMap"/>
    <column>t_order.orderid</column>
    <property rdf:resource="http://www.niwa.at/statistik#orderId"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about=
    "http://www.niwa.at/statistik/customer#OrderNetBillingAmount">
    <belongsToClassMap
        rdf:resource="http://www.niwa.at/statistik/customer#OrderClassMap"/>
    <column>t_order.netbillingamount</column>
    <property rdf:resource="http://www.niwa.at/statistik#orderNetBillingAmount"/>
</DatatypePropertyBridge>
```

```
    <DatatypePropertyBridge
        rdf:about="http://www.niwa.at/statistik/customer#OrderStatus">
        <belongsToClassMap
            rdf:resource="http://www.niwa.at/statistik/customer#OrderClassMap"/>
        <column>t_order.status</column>
        <property rdf:resource="http://www.niwa.at/statistik#orderStatus"/>
    </DatatypePropertyBridge>

    <DatatypePropertyBridge
        rdf:about="http://www.niwa.at/statistik/webshop#Advertiser">
        <belongsToClassMap
            rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
        <column>tracking2.advertiser</column>
        <property rdf:resource="http://www.niwa.at/statistik#trackingAdvertiser"/>
    </DatatypePropertyBridge>

    <DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Campaign">
        <belongsToClassMap
            rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
        <column>tracking2.campaign</column>
        <property rdf:resource="http://www.niwa.at/statistik#trackingCampaign"/>
    </DatatypePropertyBridge>

    <DatatypePropertyBridge
        rdf:about="http://www.niwa.at/statistik/webshop#CreatedAccounts">
        <belongsToClassMap
            rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
        <column>tracking2.createdAccounts</column>
        <property rdf:resource="http://www.niwa.at/statistik#createdAccounts"/>
    </DatatypePropertyBridge>

    <Database rdf:about="http://www.niwa.at/statistik/webshop#Database2">
        <allowDistinct>false</allowDistinct>
        <jdbcDSN>jdbc:postgresql://andastra.niwa.at/webshopData?user=jboss
        </jdbcDSN>
        <jdbcDriver>org.postgresql.Driver</jdbcDriver>
        <numericColumn>tracking2.createdAccounts</numericColumn>
        <numericColumn>tracking2.netbillingamount</numericColumn>
        <numericColumn>tracking2.numVisitors</numericColumn>
        <numericColumn>tracking2.numVisitorsOrdered</numericColumn>
        <numericColumn>tracking2.orderid</numericColumn>
        <numericColumn>tracking2.quickies</numericColumn>
        <numericColumn>tracking2.trackingid</numericColumn>
        <textColumn>tracking2.advertiser</textColumn>
        <textColumn>tracking2.campaign</textColumn>
        <textColumn>tracking2.date</textColumn>
        <textColumn>tracking2.domain</textColumn>
        <textColumn>tracking2.keyword</textColumn>
        <textColumn>tracking2.keywordidentifier</textColumn>
        <textColumn>tracking2.paymenttype</textColumn>
    </Database>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Date">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.date</column>
  <property rdf:resource="http://www.niwa.at/statistik#trackingDate"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Domain">
```

```
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.domain</column>
  <property rdf:resource="http://www.niwa.at/statistik#trackingDomain"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Keyword">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.keyword</column>
  <property rdf:resource="http://www.niwa.at/statistik#trackingKeyword"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/webshop#NetBillingAmount">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.netbillingamount</column>
  <property rdf:resource="http://www.niwa.at/statistik#netBillingAmount"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Paymenttype">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.paymenttype</column>
  <condition>tracking2.orderid&#62;0</condition>
  <property rdf:resource="http://www.niwa.at/statistik#paymenttype"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#Quickies">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.quickies</column>
  <property rdf:resource="http://www.niwa.at/statistik#quickies"/>
</DatatypePropertyBridge>

<ClassMap rdf:about="http://www.niwa.at/statistik/webshop#TrackingClassMap">
  <class rdf:resource="http://www.niwa.at/statistik#TrackingKeyword"/>
  <dataStorage rdf:resource="http://www.niwa.at/statistik/webshop#Database2"/>
  <uriPattern>http://www.niwa.at/statistik/concepts/
  Tracking@@tracking2.trackingid@@</uriPattern>
</ClassMap>

<ClassMap rdf:about="http://www.niwa.at/statistik/webshop#TrackingOrderDummyClassMap">
  <class rdf:resource="http://www.niwa.at/statistik#TrackingOrderDummy"/>
  <dataStorage rdf:resource="http://www.niwa.at/statistik/webshop#Database2"/>
  <uriPattern>http://www.niwa.at/statistik/concepts/
  Order@@tracking2.orderid@@</uriPattern>
</ClassMap>

<ObjectPropertyBridge
    rdf:about="http://www.niwa.at/statistik/webshop#TrackingOrderToOrder">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <condition>tracking2.orderid&#62;0</condition>
  <property rdf:resource="http://www.niwa.at/statistik#hasOrder"/>
  <refersToClassMap rdf:resource=
      "http://www.niwa.at/statistik/webshop#TrackingOrderDummyClassMap"/>
</ObjectPropertyBridge>

<ObjectPropertyBridge
    rdf:about="http://www.niwa.at/statistik/webshop#TrackingToKeyword">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <property rdf:resource="http://www.niwa.at/statistik#hasKeyword"/>
  <refersToClassMap rdf:resource=
      "http://www.niwa.at/statistik/webshop#TrackingToKeywordDummyClassMap"/>
</ObjectPropertyBridge>

<ClassMap
    rdf:about="http://www.niwa.at/statistik/webshop#TrackingToKeywordDummyClassMap">
  <class rdf:resource="http://www.niwa.at/statistik#TrackingToKeywordDummy"/>
  <dataStorage rdf:resource="http://www.niwa.at/statistik/webshop#Database2"/>
  <uriPattern>http://www.niwa.at/statistik/concepts/
  Keyword@@tracking2.keywordidentifier@@</uriPattern>
</ClassMap>

<DatatypePropertyBridge rdf:about="http://www.niwa.at/statistik/webshop#numVisitors">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.numVisitors</column>
  <property rdf:resource="http://www.niwa.at/statistik#numVisitors"/>
</DatatypePropertyBridge>

<DatatypePropertyBridge
    rdf:about="http://www.niwa.at/statistik/webshop#numVisitorsOrdered">
  <belongsToClassMap
      rdf:resource="http://www.niwa.at/statistik/webshop#TrackingClassMap"/>
  <column>tracking2.numVisitorsOrdered</column>
  <property rdf:resource="http://www.niwa.at/statistik#numVisitorsOrdered"/>
</DatatypePropertyBridge>
</rdf:RDF>
```

Listing A.4: D2RQ Map for the ATT

Bibliography

[1] Irina Astrova. Reverse engineering of relational databases to ontologies. In *The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004, May 10-12, 2004. Proceedings*, volume 3053, pages 327–341, Heraklion, Crete, Greece, 2004.

[2] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908. ACM, 2005.

[3] J. Barrasa, Ó. Corcho, and A. Gómez-Pérez. R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language. In *Second Workshop on Semantic Web and Databases (SWDB2004)*, Toronto, Canada, August 2004. Springer-Verlag.

[4] D. Becket and J. Grant. SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes. `http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/`, 2005.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284:34–43, 2001.

[6] Tim Berners-Lee. Linked Data. `http://www.w3.org/DesignIssues/LinkedData.html`, 2006.

[7] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004. `http://www.w3.org/TR/xmlschema-2/`, 2004.

[8] C. Bizer, R. Cyganiak, J. Garbers, and O. Maresch. D2RQ V0.8 - Accessing Relational Databases as Virtual RDF Graphs. `http://d2rq.org/`, accessed in June, 2013.

[9] Christian Bizer. D2R MAP - A Database to RDF Mapping Language. In *Poster Proceedings of the 12th World Wide Web Conference*, Budapest, Hungary, May 2003.

[10] Christian Bizer and Richard Cyganiak. D2R Server - Publishing Relational Databases on the Semantic Web. In *Poster Proceedings of the 5th International Semantic Web (ISWC2006)*, Athens, GA, USA, November, 2006.

[11] Christian Bizer and Andy Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *Poster Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November, 2004.

[12] Andreas Blumauer. Semnetman: Semantisch basiertes management sozialer netzwerke. `http://www.semantic-web.at/file_upload/1_tmpphp9dIbab.pdf`, 2005.

[13] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`, 2004.

[14] Christoph Bussler, Dieter Fensel, and Alexander Maedche. A conceptual architecture for semantic web enabled web services. *ACM Sigmod Record*, 31(4):24–29, 2002.

[15] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[16] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

[17] Jos de Bruijn. SEKT Deliverable D4.2.1: State-of-the-art survey on Ontology Merging and Aligning. `www.sekt-project.org/rd/deliverables/wp04/sekt-d-4-2-1-Mediation-survey-final.pdf`, 2004.

[18] Hong-Hai Do. *Schema Matching and Mapping-based Data Integration*. Dissertation, Department of Computer Science, Universität Leipzig, Germany, 2006.

[19] Schahram Dustdar, Dieter Fensel, Markus Linder, Heinrich Otruba, Tassilo Pellegrini, and Martin Schliefnig. The realization of semantic web based e-commerce and its impact on business, consumers and the economy. `http://www.austriapro.at/arbeitskreise/pva/37uebersichtsartikel.pdf`, 2006.

[20] O. Fodor and H. Werthner. Harversting lightweight ontologies out of legacy xml sources. In *Proceedings of the First International Conference on Knowledge Engineering and Decision Support (ICKEDS'04)*, 2004.

[21] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5:199–220, 1993.

[22] Patrick Hayes. RDF Semantics, W3C Recommendation 10 February 2004. `http://www.w3.org/TR/rdf-mt/`, 2004.

[23] M. Hepp. Representing the Hierarchy of Industrial Taxonomies in OWL: The gen/tax Approach. In *Proceedings of the ISWC 2005 Workshop on Semantic Web Case Studies and Best Practices for eBusiness*, 2005.

[24] Paul Johannesson. A Method for Transforming Relational Schemas Into Conceptual Schemas. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 190–201, Washington, DC, USA, 1994. IEEE Computer Society.

[25] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *Model Driven Engineering Languages and Systems*, pages 528 – 542. Springer LNCS, 2006.

[26] Peer Küppers. Datenintegration zur Laufzeit mittels Syntaktischer Übersetzung und Semantischer Mediation Relationaler Abfragen. Diplomarbeit, Fakultüt IV - Elektrotechnik und Informatik, Technische Universität Berlin, Germany, 2006.

[27] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra - a mapping framework for distributed ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, EKAW '02, pages 235–250, London, UK, UK, 2002. Springer-Verlag.

[28] Alexander Maedche and Steffen Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[29] Frank Manola and Eric Miller. RDF Primer, W3C Recommendation 10 February 2004. `http://www.w3.org/TR/rdf-primer/`, 2004.

[30] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards recovering the broken soa triangle: a software engineering perspective. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 22–28, New York, NY, USA, 2007. ACM.

[31] Borys Omelayenko and Dieter Fensel. A two-layered integration approach for product information in b2b e-commerce. In Kurt Bauknecht, SanjayKumar Madria, and Günther Pernul, editors, *Electronic Commerce and Web Technologies*, volume 2115 of *Lecture Notes in Computer Science*, pages 226–239. Springer Berlin Heidelberg, 2001.

[32] Cristian Pérez de Laborda and Stefan Conrad. Relational.OWL - A Data and Schema Representation Format Based on OWL. In Sven Hartmann and Markus Stumptner, editors, *Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, volume 43 of *CRPIT*, pages 89–96, Newcastle, Australia, 2005. ACS.

[33] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, W3C Proposed Recommendation 12 November 2007. `http://www.w3.org/TR/rdf-sparql-query/`, 2007.

[34] Jesús Barrasa Rodriguez and Asunción Gómez-Pérez. Upgrading relational legacy data to the semantic web. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 1069–1070, New York, NY, USA, 2006. ACM.

[35] Bernhard Schreder, Alexander Wahler, Markus Linder, Martin Schliefnig, and Svetlana Hollerer. Enabling Semantic Web-ready E-Commerce Solutions. In *Proceedings of the 1st European Semantic Technology Conference 2007*, Vienna, Austria, May 2007.

[36] Nuno Silva and Joao Rocha. Service-oriented ontology mapping system. In *Proceedings of the Workshop on Semantic Integration of the International Semantic Web Conference*, Sanibel Island (FL), USA, 2003.

[37] Nuno Silva and Joao Rocha. Service-oriented semi-automatic ontology bridging. In *Proceedings of the First International Conference on Knowledge Engineering and Decision Support (ICKEDS'04)*, Porto, Portugal, 2004.

[38] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004. `http://www.w3.org/TR/xmlschema-1/`, 2004.