Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Transforming Relational Data into Ontology Based RDF Data

by

Martin Švihla

A thesis submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

PhD programme: Electrical Engineering and Information Technology
Specialization: Computer Science and Engineering

June 2007

Thesis Supervisor:
Ivan Jelínek
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13
121 35 Praha 2
Czech Republic

Abstract and contributions

The semantic web is the initiative, which strives to solve the current web's problems by adding machine-understandable metadata to web resources. These metadata, when handled by intelligent agents or applications, enable automatic information processing and improve information retrieval, sharing, aggregation, or management.

It is a widely-accepted fact that the growth of the semantic web is dependent on the mass creation of metadata that will cover current web resources. In this thesis we aim to address the problem of the semantic web metadata production. Since most of the web content is backed by relational databases (RDB), this thesis is focused on transformation of relational data into RDF metadata based on mapping between a relational database schema and existing RDFS ontology. This transformation adds explicit semantics to relational data by means of semantic web technologies.

In this thesis we propose a new data transformation model based on two layers. This model is designed with regard to its performance and usability. Our work stands on the theoretical foundations of semantic web technologies but we also take into account practical issues while developing the formal model. In addition, we implement a data transformation prototype and deploy it in various case studies. The experiments with the implemented system prove that algorithms derived from our data transformation model are more efficient that those published before in related works.

The main contributions of the thesis are the following:

1. A formal description of a new model for data transformation based on schema mapping.
2. Design of two declarative XML languages based on the proposed model.
3. Proposal of a high performance data transformation algorithm based on the model.
4. Implementation of the data transformation tool and its deployment in several experimental case studies.
5. Experimental verification of our approach that gives a new view to current approaches of RDB to RDF data transformation and RDF storage.

Moreover, it appears that our data transformation model and algorithm can be generalised to be used with other data formats and formalisms. This indicates room for future research in the field.

Keywords:

semantic web, data transformation, RDF, relational database, relational model, RDFS, ontology

Acknowledgements

First of all, I would like to express my gratitude to Professor Pavel Tvrdík and my supervisor Ivan Jelínek, who accepted me for graduate study and gave me an opportunity to spend all that interesting and enjoyable time at the university. Indeed, my biggest acknowledgement goes to my thesis supervisor who supported me during my study in the formal and expert ways. I most appreciate the human approach that he takes towards his students.

Many other people influenced me and my work. I cannot name every colleague from the department so I thank them all for creating an inspiring and friendly environment. I have learned a lot working with them.

I must name some other people who showed me the purpose and means of scientific work and gave a direction to my thesis and personal development. These people are Stefan Stenudd, Robert Shipley, Vojtěch Svátek, and Darja Havelková. I would never have grown to the point of finishing this thesis without their guidance.

My research work and completed thesis would not be possible without the support of our department. Again I thank Mr. Tvrdík and also Mr. Kolář for taking care of my financial aid. Moreover, my work has been partially supported by the FRVŠ grant agency under grant no. 1804/2005, an internal grant from the Czech Technical University (IGS) under the external number CTU0507513, and from the Czech Grant Agency under grant GAČR no. 201/06/0648.

I could endlessly continue with my acknowledgements but I will stop now. The dedication of this work is for all those to whom I wish to show my respect and gratitude.

Dedication

To my parents and all others who open my spirit for possibilities.

Motto

Everything should be made as simple as possible, but no simpler.
(Albert Einstein)

Brief contents

Contents

List of Figures

List of Tables

1 Introduction

The World Wide Web (WWW) is the largest knowledge database that mankind has ever created. Its simple and open principles enable people to publish or access information in a very easy way. On the other hand, as the web continuously grows, we face problems with the amount of information and its organisation. It is getting more and more difficult to find relevant information, manage heterogeneous knowledge sources, and transparently aggregate several information providers. This problem is complicated by the fact that the most current web content is presented only for humans to read and understand. Machines – software agents, web crawlers, and search engines – can hardly understand semantic information on web pages. This means that they can only minimally help with searching for a proper piece of information, for example. In other words, for computers the web content is only data, not information.

There are several initiatives to improve the situation. One of them is the semantic web, which would give more structure and computer-understandable meaning to the data on the WWW. The semantic web is not a separate web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [14].

The idea is to enrich web resources by metadata, which are capable of describing the meaning of information in a computer-understandable way. These metadata can be used for example for semantic information retrieval, knowledge sharing, automatic data interchange, intelligent agent processing, and hypermedia adaptation.

However, these possibilities are not yet used. Though the semantic web standards are already deployed, the web is still not annotated by metadata. Metadata generation and processing are still topics of a research. Our work is focused on the generation of semantic web content, as detailed in the following section.

1.1 Motivation

Realization of the semantic web is still in its very early stage and widespread use is yet to be achieved. Its success also depends on mass creation of semantic metadata that is expected to cover the existing web by machine-understandable meaning. To reach this goal, several approaches for manual and/or automatic annotation of existing web resources

were designed.

Since a large number of web resources are dynamic websites with relational database (RDB) at the back-end, we focus our work into this area. In our work we investigate the issue of metadata generation directly from a data source, so that production is fast and cheap and metadata are always up-to-date.

Another aspect we observe is usability of the proposed solution. It is suggested [54] that the creation of metadata should be as simple as the creation of HTML. Unfortunately, this is not the case yet. The success of the web is based on its simplicity, but semantic web technologies are much more complex. One mission in semantic web research today is to hide this complexity from the web community [69].

As an answer to these challenges, in this work we propose a system for generating metadata directly from relational data. The data transformation concept is based on the mapping of a relational database schema to an ontology. This transformation adds explicit semantics to relational data by means of semantic web technologies.

The following application scenario will explain the ideas in a practical example, the formal specification of goals of our work can be found in Chapter 3.

1.1.1 Application scenario

Let us imagine a university web portal in which there are HTML web pages about teachers, students, educational activities, and research projects. The content of this portal is quite well-structured, and let us say there is one PHP script for each group of pages. In fact, this script is a template for a group of similar pages. For instance, pages about various persons are structurally identical – there is a name, position, contact, list of publications, and etcetera.

However, these pages are not machine-understandable, which means that information cannot be processed automatically. The task is to make semantic extension of this web presentation, so that every person, project, and subject has its own metadata representation in semantic web data formats.

Data for a web presentation are stored in an RDB, where they are even better structured than in HTML documents. Since in our hypothetical case a portal maintainer is responsible for deployment of a semantic extension, metadata can be produced directly from a database in the same way as HTML. Another requirement is that the semantic part of the web portal

should be independent from the classical one so that existing parts of the portal do not have to be changed.

Since the semantic web is quite a new idea and underlying technologies are much more complex than old web technologies, it can be a problem for web application developers to implement and deploy the semantic web extension of their portal. This situation can be greatly improved by providing a tool that shades web developers from the complexities of semantic technology and makes a metadata production simple and easy. In this work we propose a data transformation system as such a tool.

Using this tool the semantic metadata can be easily produced directly from a database simply by writing a set of XML files, because a programming interface of the framework is based on two proposed XML languages. Moreover, the system architecture is designed to make the system easily adaptable in the current web development process.

A process for developing the semantic web portal with our mapping framework can be briefly described as follows. First, an ontology for a university knowledge domain is developed. Then a mapping from database schema to an ontology structure is designed. According to this mapping metadata can be produced and published on the web in order to extend classical web documents. Since the presented information is well structured and there are templates for HTML pages about persons, projects, subjects and so on, it is obvious that mapping can also consist of templates, which form the base for groups of RDF documents.

Such a semantic extension would transform a university information system into a part of the semantic web, which means that its information can be transparently shared with other subjects. It also enables development of sophisticated applications (semantic information retrieval, for example; see section 9.2) on top of the semantic portal.

1.1.2 Possible uses

Academic web portals are not the only application domain of our proposal. Other possible uses cases for a database-to-metadata transformation are discussed in details in [95]. In that paper are mentioned scientific databases, syndication, and community web portals. We can add to this list e-commerce catalogues, digital libraries, and other types of web applications, but the idea is clear: the relevant target group for such transformation is a large number of dynamic web sites built on top of RDBs.

1.2 Contributions of the thesis

The detailed research field state-of-the-art and formal goal specification are described in the following chapters. In this section we only provide a short list of thesis contributions:

1. A novel model for data transformation was designed and formally described. The model differs from other approaches (Section 2.3) – it is simpler than others while supporting all features of RDF and RDFS. The direct consequences of this simplicity are higher usability and performance. To reach such parameters, we:
   - identified correspondences between relational model and RDFS and also between the structure of relational data and RDF and
   - divided the model architecture into two layers: one for schema mapping and the another for instance transformation based on schema mapping.
2. We proposed two XML languages based on the formal model. One language is capable of describing schema mapping between relational schema and RDFS ontology and the other one is for building RDF documents from relational database.
3. The part of the formal data transformation model is a proposal of high performance algorithm for data transformation.
4. A data transformation tool was implemented according to proposed concepts and languages. This tool was also deployed in several test case studies and proved usability and high performance of our theoretical ideas.
5. We performed a complex set of experiments with our system as well as with similar solutions. The test results proved the high performance of our approach. Moreover, they brought a completely new view to the field of RDB to RDF data transformation and RDF storage (as discussed in Section 8.3).

Although this thesis is focused on the particular schema formalisms (relational model, RDFS) and data formats (RDF), it appears that proposed concepts and algorithms can be generalised and used for other formalisms or formats. This issue is discussed in the future work proposal (Section 10.3).

1.3 Thesis structure

This chapter gives only a very brief introduction to the problem and our work. Chapter 2 describe the research field and its state-of-the-art in a detail way and Chapter 3 gives an overview to our approach – thesis goals and methodology.

The formal core of the thesis is contained in chapters 4, 5 and 6. Chapter 4 proposes the abstract data transformation architecture and formal description of the data transformation model. Two XML languages built on this formal foundations are described in Chapter 5 together with their normative syntax. All these issues are summarised in Chapter 6, where we formally consider the completeness of our approach according to the used schema formalisms and data formats.

Chapter 7 provides an overview to the data transformation system implementation – its architecture, algorithms, performance, usability issues and supporting tools. A detailed performance analysis is given in Chapter 8, where we compare our tool with other approaches. Several cases of the system deployment are briefly described in Chapter 9.

Chapter 10 contains the conclusion of the thesis, discussion about goals and achieved results, as well as proposals for future work.

2 Background and state of the art

The semantic web is an expanding initiative that grows from various research areas. In this chapter we give a brief introduction to its underlying concepts and technologies.

Since our work is focused on the semantic web content generation, we also discuss related works from this field. Though this issue is only a small part of semantic web activity, many different approaches to this problem were developed. We introduce various concepts of metadata production with a special focus on the problem of relational model to RDF mapping.

2.1 Semantic web

2.1.1 Overview

As discussed in the introduction, the World Wide Web is currently reaching its limits. Many services on the web, e.g. full-text search or catalogue services, are becoming inappropriate with the growing volume of information on the web. According to this, the web has been naturally changing in recent years. There are various streams in the field based on various concepts. A big part of knowledge management has moved to the web community and we can witness phenomena like Wikipedia [101] and social bookmarking [100]. The need for interoperability and cooperation is followed by the emergence of new communication protocols as web services [27] or REST [37] and new standardised data formats – XML [43] and many its sublanguages (SVG, MathML, XForms) or ODF (Open Document Format [74]) develop.

However, many problems on the web occur just because computer programs that process web documents do not understand the meaning of the web content; the web was designed mainly for human users. However, classical techniques of artificial intelligence, like natural language processing, are not able to solve this problem today due to the huge heterogeneity, lack of central control, and openness of the web.

The semantic web [14] is an initiative that focuses on solving the mentioned problems in the global web environment. It is not based on classical artificial intelligence. Its idea is quite simple: it provides a simple method to express information in computer-understandable way. Since it was proposed as an extension of the current web, its standards are designed

for a distributed data environment. Since both information and its structure are described in a standardised fashion, the computer application can "understand" this information and process it automatically.

2.1.2 History and current status

The process of adding semantic metadata to the web content in order to help computers with processing of web content started very early – almost right after the birth of the web.

HTML language, used on the World Wide Web, had no mechanisms to semantically markup the content in its very first versions. According to this some metadata tags were added to HTML 1.2 early on (in 1993), but due to their simplicity they caused almost no improvement.

Another attempt, trying to add ontology into web pages, was the introduction of Simple HTML Ontology Extensions (SHOE) [52] in 1995, but this language was never widely used.

It can be said that semantic web activity began in 1997, when Resource Description Framework (RDF) was proposed. However, the complete concept of the semantic web was presented in 2000 [11]. This presentation proposed the philosophy, architecture and practical deployment of the semantic web.

Since 2000, we have seen very intensive research activity, focused mainly on the deployment of semantic web standards. As a result of this effort all fundamental standards (RDF, RDFS and OWL) were released as W3C recommendations in the first half of 2004.

Consequently many experimental projects were deployed in the year 2004 to demonstrate semantic web capabilities in areas as web search [32], knowledge sharing [84], and information management [29].

However, the semantic web vision of a universal medium of data exchange is not yet realized. This is not only because of lingering practical issues, but mainly because the semantic web idea is not yet accepted by the wider web community. From a practical point of view there is no profit for a common web portal to enter the semantic web.

This has two explanations – there are almost no semantic web metadata on the web (compared to the amount of web documents) and there are no applications that would make metadata useful. These two reasons creates a deadlock – web page maintainers would not provide their content in metadata formats because of a lack of applications and software

companies would not deploy applications that have no metadata to process.

Unfortunately, this status is ignored almost by all semantic web researchers today. The activity in the field in the last two years has been focused on improving expressiveness of the technology (named graphs [21], Fuzzy OWL [90] etc.). This work would have made sense if there were already metadata and processing applications that could be improved, which is not yet the case. Considering practical applications being developed in the community, they are very simple (e.g. FOAF [18], RSS1.0 [82], DublinCore [30]) or the technology is closed in a particular application domain instead of being open to the whole web (e.g. K-Space activity[1], KP-Lab[2] or many others), or both.

In this thesis we would like to stress the importance of easy and effective RDF production based on currently available standards. We are not alone in this opinion – very respected authorities have recently pointed in this direction.

Rudi Studer challenged the semantic web community to create applications that are bigger than "Tweety"[3] (i.e. bigger than very small) at the International Semantic Web Conference 2006 [93].

Tim Berners-Lee also called at researchers on semantic web conferences to focus on practical application of the technology rather than on extending logical foundations of existing standards[4].

According to this, the current task for the semantic web community is to *(i)* create a critical mass of metadata that will activate a chain reaction of metadata production and application deployment, and *(ii)* bring complex and complicated semantic web technologies closer to the web community.

This thesis is to address both of these challenges.

[1] K-Space project website is available at http://kspace.qmul.net/.

[2] KP-Lab project website is available at http://www.kp-lab.org/.

[3] Tweety is a small bird from Warner Brothers cartoons, always chased by Sylvester the Tom Tom Tomcat

[4] This paragraph is only an informal statement. The issue is very sensitive and we cannot find any formal document on this. It happen in 2005 during a Lee's talk at ISWC or ESWC (according to an informal information from Prof. RNDr. Peter Vojtáš, DrSc.). However, some points on this issue can be found in Lee's presentation at ISWC 2005 Industry Day ([12], page 26)

2.1.3 Technologies

In this section we discuss semantic web technologies. Since the semantic web is built on top of World Wide Web, first we introduce the concepts lying under the semantic web. Thus we will briefly look at the Internet and World Wide Web and we will describe how they relate to the semantic web. Then XML will be discussed as the base for the semantic web and the description of RDF and ontologies will follow.

The technologies of the semantic web create a pyramid composed of layers (Figure 2.1). This schema is quite old (proposed in 1999 by Tim Berners-Lee) but it can still serve as a simple illustration of the semantic web architecture. The basic layer of data representation is RDF model and schema, which is based on XML, URI and Unicode. On the top of RDF there is an ontology layer to define a vocabulary and rules for data expressed by RDF. Logical rules from an ontology can be used for logical reasoning, where a new piece of information can be inferred from an asserted one.

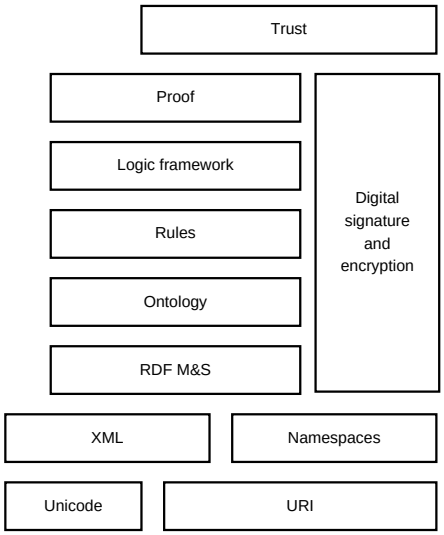


Figure 2.1: Semantic web technologies

In the following text we describe the semantic web technologies from Figure 2.1 up to the ontology layer. Layers above ontologies are beyond the scope of our work.

2.1.3.1 Internet

The current Internet grew from a small project, ARPANET, that was founded in 1969 by the Advanced Research Projects Agency (ARPA) [67]. Very soon after, two new protocols were introduced[5] – Transmission Control Protocol (TCP) for data transfer and Internet Protocol (IP) for member computer addressing. These two protocols are known together as TCP/IP, low level protocol, which is a base for the current Internet.

High level protocols for various Internet services rely on TCP/IP. These protocols are for example the File Transport Protocol (FTP), Simple Mail Transfer Protocol (SMTP) or Hypertext Transfer Protocol (HTTP) [36].

HTTP is a base for one of the most common services on the Internet – the World Wide Web.

2.1.3.2 World Wide Web

The fundamental concepts of the World Wide Web were proposed by Tim Berners-Lee [9] in 1989. In this document, Lee described effective information management by interconnected computers as an answer to an information overload in CERN (Figure 2.2). The solution was based on a system of distributed documents joined by hyperlinks[6]. As a tool for its implementation, the very first version of HyperText Markup Language (HTML) was designed.

Basic principles of the web are really simple. It uses a client-server model. Data are stored in so-called web servers and clients (web browsers) can access and render these data for users. Web resources (mainly HTML documents) in servers are addressed by Unified Resource Locator (URL). The communication is defined by the Hypertext Transfer Protocol (HTTP) [36] and consists of requests and responses. The client requests a document with a particular URL and the server sends this document in a response. The result of such simple architecture is that anyone can very easily publish information on the web and also access this information pool.

An HTML[7] document is a simple text file that includes some special marks. These marks,

[5] Interesting coincidence: Vinton Gray Cerf, one of authors of the protocols, visited Prague in April 2007 during the time this section was written.

[6] It is not without interest that a very similar concept was proposed by Vannevar Bush in 1945 [20]

[7] We refer only to HTML as to a language of hypertext documents here. XHTML is discussed in the following section.

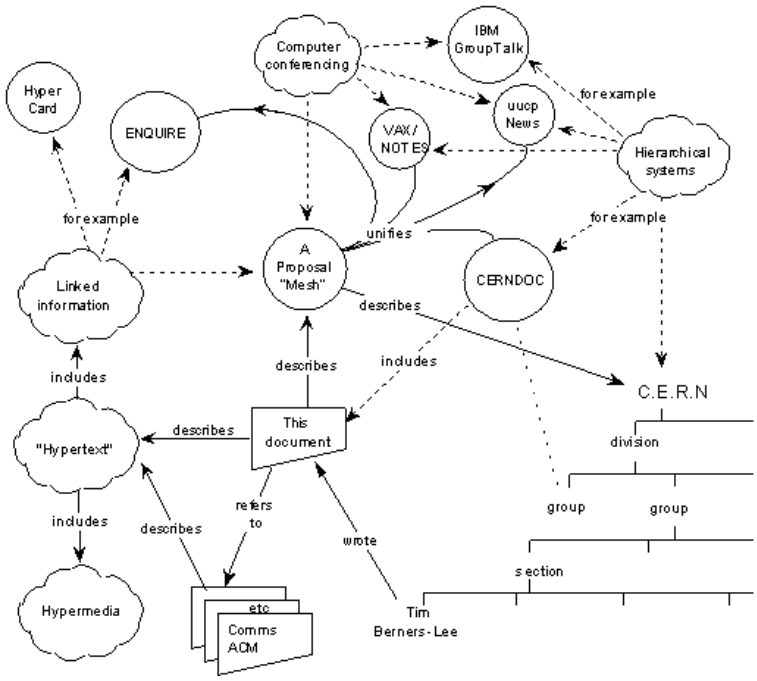


Figure 2.2: Tim Berners-Lee's proposal of distributed hypertext system (from [9])

called tags, mainly define how the text is organised and presented. They can identify headers, tables, and lists, and can also format the text itself by specification of a font, colour and so on. The most important tag in HTML is the anchor tag (`<A HREF="`*url*`">`*text*`</A>`) that indicates that the text within it is a link to a document with an address defined by URL.

HTML also provides weak support for expressing semantic information. The tag `<META>` specifies general information about the whole document including the name of the author, the description, or keywords. Then there are attributes `<REL>` and `<REV>` for tags `<A>` and `<LINK>`. The former specifies the relationship of the parent document to the hyperlinked document and the latter does the same in the reverse direction.

Even though HTML itself can not express the semantics of a content, it can be enriched by various metadata languages. Two main approaches to enriching HTML documents by semantics are embedding metadata to HTML and linking metadata from HTML. Both issues are discussed in [75], but currently there is no official recommendation to solve this problem.

However, HTML language is designed to make information accessible for a human user and

is not suitable even for a simple data interchange between computers. XML was developed for this task.

2.1.3.3 XML

XML stands for Extensible Markup Language [102] and it is (similarly to HTML) an application of meta-language called Standard Generalized Markup Language (SGML) [58].

XML adds tags to a text stream to provide some structure and additional information in the same way that HTML does, however, XML tags do not provide any explicit meaning.

The flexibility of XML allows anyone to describe any content easily, but this freedom can cause lack of understanding between a document's author and its consumer. The semantics of tags in XML is only implicit; machine processing of XML documents usually relies on standardised tag sets, e.g. DocBook [99], Scalable Vector Graphics (SVG) [98] or MathML [97].

XHTML (Extensible HyperText Markup Language) [96] was delivered in January 2000 by W3C to bring advantages of XML directly to HTML. This language, built on XML, was expected to improve accessibility and functionality on the web. There are not many new or deprecated tags and attributes in XHTML compared to HTML, there is mainly just a new set of coding rules. XHTML was also easily extensible by any other XML documents, which means, for instance, that it could embed fragments of RDF[8].

Such a standardised grammar for XML documents can be formally defined by Document Type Definition (DTD) [102], XML Schema [33] or RELAX NG [73] schema. The older way is DTD, which specifies valid elements and their attributes. XML Schema is a more recent language for restricting the structure of XML documents and also extends XML with datatypes. Moreover, it uses XML as a serialisation syntax. A RELAX NG schema specifies a pattern for the structure and content of an XML document. A RELAX NG schema can be serialised as an XML document but the specification also offers a compact, non-XML syntax. Compared to other schema languages, RELAX NG is relatively simple.

An XML document that has an associated grammar and conforms to the rules defined by this grammar is said to be valid.

[8]However, these features were never properly supported by major web browsers and thus it seems the web is not yet prepared for the technology. Due to this fact and in response to pressure from the web community, the new W3C HTML Working Group was recently established (March 7th, 2007) to continue HTML development.

However, XML and its grammar languages provide only syntax for structured document and have no capability to express its exact meaning. This drawback makes it difficult to integrate or interchange XML documents automatically. To express a semantic of information the Resource Description Framework (RDF) was developed.

2.1.3.4 RDF

To address XML semantic limitation the Resource Description Framework (RDF) [70] was developed as a language capable of expressing meaning of information.

Compared to XML, which is document-oriented, RDF takes into consideration a knowledge-oriented approach that is designed specifically for the web. One of the advantages of RDF over XML is that an RDF graph depicts in a unique form the information to be conveyed while there are several possibilities to represent the same semantic graph in XML[40].

RDF, based on the concept of semantic networks, provides a simple yet powerful data model for semantic assertions. This model is based on so called triples. Using these triples a semantic of information is formulated like an elementary sentence consisting of a subject, verb and object. In this way we can make assertions that a subject, e.g. *scientific paper*, has a property, e.g. *is a friend of*, with a particular value, such as *person*. Resources (subjects) and predicates (verbs) are identified by URI, and a value of a property can either be literal or be another URI identified resource. RDF data model is formally detailed in Section 4.2.2.

RDF provides some more features to this datamodel such as collections handling, typed literals, and reification, which is the possibility to make assertions about a whole statement (triple).

In the semantic web, RDF is used to describe resources in a machine-understandable way. RDF metadata can contain assertions about anything that can be identified by URI. While the most common URI schema is web URL for web resources identification, assertions can also be made about real world entities such as books or even people.

There are various serialisation syntaxes for RDF, but the most common one for a RDF document exchange is RDF/XML (based on XML), since RDF was designed to be complementary to XML in order to specify semantics for data based on XML in a standardised and interoperable manner.

However, RDF is only a tool for resource description; it is not able to define concepts

in a knowledge domain. Ontologies are used to specify a vocabulary of terms and their relationships for RDF documents in the semantic web.

2.1.3.5 Ontologies

To enable an integration of different sources, there is a need for a shared understanding of the relevant domain. According to this ontologies were added to the concept of the semantic web. Ontologies are able to provide a common vocabulary to support the sharing and reuse of knowledge.

The term ontology is described by various incompatible definitions. It originally came from philosophy, where it denotes a study of the nature and organisation of reality. In computer science and especially in artificial intelligence the most common definition is the one from Gruber [45]: "An ontology is an explicit specification of a conceptualisation," where a conceptualisation is an abstract view of the world.

Ontologies formally describe terms used in metadata since an ontology can be used to explicitly represent the meaning and relationships of terms in vocabularies when the information contained in documents needs to be processed by applications.

There are several ontology formalisms – e.g. SHOE [52], DAML [53], OIL [35], DAML+OIL [56], RDFS [17], and OWL [88]. RDFS and OWL are mainly used in the semantic web based on RDF.

RDFS is the next level above RDF as a language to create controlled, sharable, and extensible vocabularies. It allows one to define a hierarchies of classes and properties for RDF resources. However, RDFS is simple and provides only basic capabilities for semantic description in RDF. For this reason another ontology language was constructed on top – Web Ontology Language (OWL).

OWL adds more vocabulary for describing properties and classes, including: relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

The more formal description of RDFS can be found in Section 4.2.3 and OWL is described in 4.2.4.

2.2 Metadata generation

One of the core challenges of the semantic web is the annotation of existing web resources by semantic metadata. There are various approaches to this issue, which we divided into several categories by the following criteria:

- Who is annotating web resources?
- What is the source for the metadata creation?
- What is the level of the annotation automation?

The author of the metadata can be *(i)* a maintainer of the annotated web resource or *(ii)* a third party. In the latter case annotation can be provided by one subject or by mass collaboration.

A data source for the metadata creation can be *(i)* an annotated web resource itself (typically an HTML web page), *(ii)* a database from which the web resource was generated (e.g. a relational database), or *(iii)* some data created specially for the purpose of annotation (e.g. an XML file).

The annotation itself can be *(i)* manual, when the author of a web resource or the third-party annotator uses some annotation tool to create metadata, *(ii)* semi-automatic, when a human user effort is necessary to create a semantic template upon which the machine-processed annotation is based, or *(iii)* automatic, when the artificial intelligence techniques are used to recognise web resource semantics.

In the following subsections manual and semi-automatic annotation is discussed while other criteria are evaluated within this division. In this work we do not consider the fully automatic semantic annotation, which is based on natural language processing, or automatic schema matching.

2.2.1 Manual annotation

The simplest use of the web resource annotation involves a single user editor. This application shows an annotated resource (e.g. HTML web page or PDF file) and enables the user to create metadata instances and their relationships by some GUI. An application generates metadata in the appropriate format. In case the annotator is also an owner

of the annotated document, metadata can be embedded in the document or linked from it. If a metadata author cannot modify an original document, he can publish metadata separately and link an annotated document from them.

However, this kind of annotation is slow and expensive and produced metadata are always static.

A more interesting method of annotation is the creation of metadata by mass collaboration, i.e. by combining the efforts of a large number of people. For this purpose special applications were developed to provide means for users to communicate about web documents by attaching external annotation metadata to the documents. Users can easily add their metadata to the web resource by using such an application and can also find annotations related to a particular document. Several projects address this issue – e.g. Annotea [61], CREAM [47], MnM [94], or Mindswap [42].

These approaches usually assume that web resources are static. However, a main part of web resources is generated dynamically, very often from relational databases. If these resources were annotated by manual approaches, created metadata would become outdated very quickly.

2.2.2 Semi-automatic annotation

Semi-automatic generation means that metadata are produced automatically, but rules on how to extract semantics from a data source are designed by a human user.

There are various approaches that differ mainly in data sources for the metadata production. One common data source is a relational database, which can be used when metadata are provided directly by a web presentation maintainer who has access to this database. This approach is based on mapping of relational model into an RDF data model and is discussed in Section 2.3.

When a web resource maintainer does not want to provide database access to third parties, but wants them to query a data source, it is possible to add more semantic to the web resource. This approach can be illustrated, for example, by the so-called *On Deep Annotation* [48]. The goal of this project is to allow interested parties to gain access to the source data by providing a semantic mapping between a relational database and the resulting HTML document.

The third possibility is when the annotation author only has access to the resulting web

resource, typically the HTML web page. The semantic extraction from the web resource can be done by using wrappers [83]. These computer applications use explicit definition of HTML or XML queries to extract information from an HTML page. This way metadata can be created from a set of structurally similar web pages. The semantics of annotated web resources is usually recognised by a human user who creates HTML queries, although there are also approaches using AI techniques [24]. However, the shortcoming of this approach is that the metadata production depends on a document layout rather than on the underlying data structures.

2.3 Existing relational model to RDF mapping approaches

Since a great amount of data is stored in relational databases and one idea of the semantic web is to enable transparent interoperability between these data sources, there are several studies of relational data to RDF migration. These approaches differs in purpose, architecture, usability, algorithm efficiency, and so on. In this section, we examine major contributions to this field and their main attributes.

As discussed in the previous chapter, the question is how to expose the content of a relational database as RDF. This problem can be solved by two different approaches:

- **A relational database is queried by an RDF query language.** Mapping is created between relational database schema and predefined ontology and based on this mapping, relational data can be queried by semantic queries. A result of the query is a set of RDF statements.
- **A relational database content is transformed into RDF.** A subset of relational database content is transformed into RDF. Resulting RDF can be stored in static RDF documents or in a native RDF database.

Both approaches are discussed in the following subsections, but first we mention the related research areas and early proposals in our field.

It is necessary to stress that our approach does not fit into this simple classification since it stands between these two main groups. We do not use any standardised semantic query language to fetch data from relational databases nor do we transform whole database content into RDF. These issues are detailed in chapters 3 and 4.

2.3.1 Mapping and data transformation in other domains

A long time before the semantic web appeared there was a need for schema mapping and data transformation in order to support integration across organisational and application boundaries.

A lot of work was done in database schema integration in the 1980s (summed in the survey [3]) and this effort was extended with automatic schema matching approaches in the 1990s [81]. These works refer to integration of relational databases.

Another very inspiring area is a migration of relational schemas and data to object-oriented (OO) databases [15]. RDFS formalism is very similar in structure to OO with respect to classes, properties, and inheritance rules. It is very interesting that evolution of RDB to OO DB transformation is quite similar to RDB to RDF approaches. A spectrum of relational schema approaches to OO mapping range from fully-automated transformers that omit many OO features [1] through interactive mapping tools [59] to extensive methodologies for a reverse engineering with a semantic enrichment [46]. These approaches mostly consider schema mapping but not data transformation. An interesting exception is [8], which considers a complete migration process, i.e. semantic enrichment, schema transformation, and data migration.

XML was introduced in 1996 and a need for data migration between XML and RDBs appeared immediately. Many approaches considered mappings between relational schemata and DTD [102], XML Schema [33] or RELAX NG [73] schemata. Production of XML from relational datasources was proposed, for example in [87], and the opposite direction is discussed in [39].

As different ontology formalisms evolved during the late 1990s (see Section 2.1.3.5) there were also attempts to map relational schemas to these formalisms. One example is proposed in [62], an approach for designing an ontology for information retrieval based on: (a) the schemas of the databases, and (b) a collection of queries that are of interest to the users.

Works on RDB to RDF migration described in the following text stand more or less on these foundations.

2.3.2 Early proposals

The semantic web is sometimes proposed as a huge distributed database. Thus very early, in 1998 (one year after the first RDF proposal), Tim Berners-Lee proposed the first work on exposing a relational database on the semantic web [10]. This document compares the relational model with the RDF data model and also discusses mapping issues. However, this mapping of relational data to RDF is rather basic, omitting completely, e.g., a concrete schema representation.

A more complex proposal is in [7], where the authors describe a naïve approach for mapping RDBMS data onto RDF data. The proposal covers concept naming, relationships, datatypes and other problems. However, the work is still mainly on the level of the RDF data model and not on the level of schema mapping.

2.3.3 RDF gateways to relational databases

SquirrelRDF [89] is a tool which allows relational databases to be queried using SPARQL [79]. It is just an implementation of *the naïve RDB to RDF mapping*, described in [7], thus an ontology is not considered. When mapping is created, RDB can be queried by SPARQL [79]. A result of the SPARQL query over RDB can be RDF (in case a *CONSTRUCT* query is used). The SquirrelRDF uses Jena [22] API to perform SPARQL queries.

The work proposed in [64] extends the naïve approach, focusing on linking relational and RDFS schemata. The system called FDR2 is designed for integration of relational-like information resources with RDFS-aware systems by linking a relational database schema with a predefined domain ontology. This approach describes a relational schema (all of its concepts and relationships) by RDFS and uses this description to join relational schema and given ontology. The system requires an RDFS reasoner to process the mapping, which poses a serious performance problem. Moreover, the work does not solve the problem of data transformation – it is designed to allow querying a relational database by concepts from a predefined domain ontology.

An approach very similar to the previous one is DartGrid, proposed in [23], which should provide a web-scale integration infrastructure for distributed relational database resources. The integration platform uses RDF/OWL to define mediated schemas, thus there is a need to transform relational data into RDF according to existing OWL ontology. In this process, first the database semantic is described by RDF/OWL using a set of predefined

rules. This source data semantic is later mapped into a shared ontology in order to allow semantic querying of the database. For this querying, the authors developed their own query language called Q3. The mapping approach has the same structure as the previous one, although it provides more sophisticated mapping according to use of OWL, and also suffers from the same drawbacks.

Another contribution in this group discussed in this document is Federate [78], which is designed to provide a consistent RDF interface to relational databases. The approach uses RDF agents with the ability to query a relational database with an application-specific schema. First the very simple mapping is created between relational database schema and ontology concepts and then a database is queried by an RDF query language. The query is translated to SQL in accordance with the mapping. However, returned data are in the form of a relation.

Very recent approach to DB to RDF migration is R2DQ [85]. This work introduces D2R, a declarative language to describe mappings between relational database schemas and OWL ontologies. The mapping process consists of 4 steps: *(i)* selection of a record set from the database using SQL, *(ii)* grouping of the record sets, *(iii)* creation of class instances and identifier construction and *(iv)* mapping of the grouped record set data to instance properties. The mapping is used by a processor that transforms relational data into RDF, emitting RDF/XML [5], N3 [13] or N-Triples [4]. The processor is based on JENA API [22]. In the last version (0.5) a database can be queried by SPARQL.

2.3.4 Relational database to RDF transformers

The work described in [91] introduces the system called KAON REVERSE, which is intended to be a plug-in to the large semantic web framework KAON [16]. Using this system, data from a relational database are transformed into a RDF knowledge base based on mapping of a logical database schema into an existing ontology. The approach starts with transforming the relational database model into a corresponding ontological structure based on F-Logic, which is then used for mapping the content of the database into an RDF knowledge base. This knowledge base can be queried by RDF query languages and resulting RDF statements can be published on the web. When mapping is created from this approach, the final layout of produced RDF is also defined.

Another work on relational database to RDF transformation is a part of the project MuseumFinnland [57]. This approach is focused mainly on interoperability between different

database maintainers. In [80] a two-phase data transformation is described. A database content is transformed into XML repository conforming to an XML Schema. These XML data are then transformed into RDF metadata that are semantically validated against a set of predefined RDFS ontologies. The whole relational database is migrated into RDF without any possibility of transformation control. This approach uses a conceptual database schema for mapping.

D2RQ [85], mentioned in the previous subsection, also belongs in this category. In its early version (in 2004), it was just a system exporting data from a relational database into RDF.

We can summarise that all approaches discussed above use some RDF API to handle RDF data and ontologies and that the layout of produced RDF is rather static.

3 Overview of our approach

3.1 Definitions and terminology

This section introduces some basic terms used in this document. It should be noted that some concepts from our research area are addressed by different terms in various works. In such case we choose one term and define it here. However, this chapter gives only a brief overview to the terms. Their formal definitions are in the following sections.

Since the work is focused on data transformation, we should explain general terms such as *schema formalism*, *schema*, *data* and *database*.

The term *data* will denote a digital representation of atomic facts. However, the word is often used as plural (e.g. relational data). In that case the term simply denotes digital representations of facts. A *database* is a structured set of data. It can be a relational database, an XML document or a simple file. A *schema* defines the structure of a database. Thus data can be expressed by a set of concepts and their relationships provided by schema. An example for a schema is a relational database schema, where relations and attributes define a structure for data. The way a schema describes a data structure is defined by a *schema formalism*. Every formalism defines a limited set of terms and relationships that can be used to describe a schema. For instance, a relational database schema is a representation of the formalism called the relational model. A schema is based on a particular schema formalism, which can be described using some *schema formalism language*.

The meaning of data will be addressed as *semantics*. If semantics is expressed directly by data and their structure, it will be called *explicit semantics*.

In the semantic web the term *metadata* is very common. In this context metadata means data represented in machine-understandable fashion [34]. This means that metadata has explicit semantics. Terms, their relationships and other rules for metadata are defined by an *ontology*. A particular ontology is a schema, used ontology language is a schema formalism (language). More specifically, the current semantic web technologies are RDF, RDFS and OWL. RDF provides a simple data model to express metadata while RDFS and OWL are ontology languages. Although RDF can be seen as a sort of schema formalism, in this work we understand RDF as a data serialisation format and RDFS as a schema formalism (and also as a schema formalism language). Then a particular RDFS ontology is a schema and an RDF document is a database.

A *relational model* is one schema formalism in the relational database world. In this model, data are stored in so-called *relations* (or tables), which are defined by a *relation schema*. A set of relation schemas creates a *relational database schema*, which is generally a *schema* for data. The term *relational database* (RDB) corresponds with the more general *database*. Another common term, *RDBMS* (relational database management system), denotes a software system that manages relational databases. Neither relational data nor relational model do not express their semantics explicitly.

A note should be made on the terms *relation* and *relationship*. Since *relation* is a term for the particular data structure (a table) in the relational database world, we decided to use the word *relationship* to express two things that relate to each other.

3.2 Topic overview

As described in the introduction, the area of our interest is the semantic web, and more specifically, a dynamic generation of the semantic web content. When investigating our research area and its problems, we identified the following points:

- A large part of the web content is generated from relational data stored in RDBMSes. It is not possible to give an exact ratio of dynamic to static web pages, but some sources (e.g. [48]) claim dynamic web pages outnumber static ones 100 to 1.
- The semantic web standards (RDF – Resource Description Framework [5], RDF-Schema (RDFS) [17] and OWL – Web Ontology Language [88]) are already deployed.
- The semantic web technology is considered to be complex and complicated by people outside the semantic web community.

Considering these facts, we specified our goals as follows. We propose a system that generates semantic metadata directly from a relational database. This production is based on the schema mapping between a relational database schema and ontology.

Since our research is focused on the semantic web, our proposal addresses particular semantic web technologies. The format of produced metadata is RDF (Resource Description Framework [5]) and the format of ontology language is RDFS (RDF-Schema [17]). We also aim to support OWL [88].

Regarding the problem of complexity of semantic web technologies, we execute a strong effort to design a data transformation model that is usable even for web developers without a deep knowledge of RDF and RDFS.

3.3 Partial goals

There are several additional aspects that can be seen as partial goals of our research. These aspects, influencing the design of both theoretical models and their implementation, are briefly detailed in the following subsections.

3.3.1 Data transformation

The main purpose of our work is to enable transformation of relational data into an RDF document. The transformation is based on a mapping between a relational database schema and ontology. We can say that input for our transformation are relational data in a database, an output is an RDF document, and the transformation is restricted by constraints in a particular database schema and ontology.

However, database data and RDF documents differ not only in their serialisation formats or the methods of access. The most significant difference is in the underlying data models the relational data model on one side and the RDF-directed labeled graph on the other.

Our goal is to create a complete formal mapping between these two data models while also considering other restrictions, e.g. the RDFS language capacity. Moreover, the data transformation model design is influenced by its purpose and also by other goals, primarily, performance and usability.

3.3.2 Performance

The speed of the RDF production is another limiting factor for us. The architecture and algorithms of our model must be efficient from this point of view – our goal is to make an RDF production time-efficient.

When processing an RDF document with a computer, it is usually transformed into a document object model (DOM) using some RDF API (e.g. Jena2 [22]). However, building large RDF files with a particular ontology with an RDF API can be very time- and memory-

consuming processes.

We argue that using full-scale RDF API is a bottle-neck of the data transformation. We believe that finding an algorithm that does not use it will improve the performance of this process.

3.3.3 Usability

The abstract model is designed to be a base for an implementation of the data transformation system. This system is intended to be a tool to simplify a semantic metadata production. To achieve this goal, a complexity of semantic web technologies must be hidden from web developers who use the data transformation application.

We have kept this idea in mind since the very beginning of our work. Not only a system implementation, but also the model architecture and developed XML languages must be designed in order to achieve this goal. We can see an aspect of usability in the following criteria:

- The data transformation model is designed to hide the complexity of the semantic web technology from web developers.
- The data transformation languages are as simple as possible, and are easy to understand and use.
- The framework architecture and implementation are designed to make the framework easily adaptable in the current web development process.

3.4 Formal goal specification

The main goal of our research – *a transformation of relational data into RDF documents* – is formally specified in this subsection. In this work we deal with two particular schema formalisms (a relational model and RDFS ontology language) but the problem is specified generally here. The following formal definitions are inspired by [2]

Definition 3.4.1 (Schema) *The schema S is a structure consisting of concepts, their relationships, and restrictions:*

$$S = (CON,\ REL,\ RES)$$

where

- *CON is a set of concepts,*
- *REL is a set of relationships between concepts,*
- *RES is a set of restrictions.*

There are various ways to practically represent a schema – e.g. a relational model, ER model, object-oriented model, directed graph and so on. We will address them as *schema formalisms*.

Every formalism defines a limited set of terms, relationships, and restrictions that can be used to describe a schema. For example, there can be terms such as a `class` or `entity` and relationships like `subclassOf` or `is-a`.

Definition 3.4.2 (Schema formalism) *Schema formalism ϕ is a triple*

$$\phi = (C_\phi,\ R_\phi,\ S_\phi)$$

where

- *C_ϕ is a set of formalism concepts,*
- *R_ϕ is a set of formalism relationships,*
- *S_ϕ is a set of structural restrictions on relationships between concepts.*

If we apply this definition, for instance, to the relational model, the formalism concepts are relation schemas and their attributes; relationships between relations can be expressed by primary and foreign keys and restrictions are defined by integrity constraints. It is very similar for other formalisms. This issue is discussed more in sections 4.2.1 (for the relational model) and 4.2.3 (for the RDFS). Table 3.1 shows an example of two different schemas, S1 and S2, that describes the same domain. S1 is an instance of relational schema and S2 is a hierarchy of classes and properties in an (RDFS) ontology.

Data representing schema concepts can be referred to as *instances of schema concepts* and a set of such data structured in accordance with a particular schema is a *database*.

S1 concepts	S2 concepts
Person	Thing
#id	Being::Thing
username	Person::Being
first_name	firstName
last_name	lastName

Table 3.1: Schema sample

Definition 3.4.3 (Instance) *The instance I is data representing a concept from the schema S and conforming to the schema.*

Definition 3.4.4 (Database) *The database D is a set of instances that conform to the schema S and satisfy all constraints in S.*

A *database* can be a relational database stored in RDBMS (as a set of instances conforming a particular relational database schema) or an RDF document as a set of statements.

Since we defined the main terms in our research area, we can now formally define the goal of our work. Our goal is to design a mapping model that performs a *data transformation*, defined as follows:

Definition 3.4.5 (Data transformation) *Data transformation is an operation $t_{s,t}$, which is able to create a target database D_t conforming to a schema S_t from data stored in a source database D_s conforming a schema S_s.*

$$t_{s,t} \ : \ (D_s \ \rightarrow D_t) \mid D_s \ conforms \ S_s, \ D_t \ conforms \ S_t$$

3.5 Goal summary

To summarise the main goal of our research, we will design a novel model for transforming relational data into RDF documents. The data migration is based on a mapping between a relational database schema and RDFS ontology. The models architecture is intended to be the base for an implementation of a high-performance data transformation system that is easy to use by web application developers.

Proposed data transformation would allow the publication of relational data without explicit semantics as an RDF data with explicit semantics defined by an ontology.

3.6 Methodology of the work

It can be said that RDF is a very simple data format based on an expressive and scalable data model. An advantage of RDF is that metadata can be enriched by rules expressed by higher ontology languages (RDFS, OWL, and so on). We reason that data transformation from RDB to RDF can be also very simple even when mapping between relational schema and RDFS ontology is considered.

Other approaches, described in Section 2.3 are generally *(i)* too simple (e.g. based on naïve approach, which doesn't consider any ontology [89]) or *(ii)* too complicated (e.g. based on some kind of logic, which is not necessary for this task [16]). One approach (R2DQ [85]) has a similar principle to our work but differs in the data transformation model and query interface and thus is designed for a slightly different purpose. Moreover, due to its architecture, it features significantly lower performance than our system.

When considering related work and our goals, we decided to keep our work *as simple as possible, but no simpler*. The methodology of this thesis follows this principle.

3.6.1 Formal model and derived languages

All formal issues considering RDBs are based on relational model and algebra. RDF and RDFS formalisation in this thesis is based on *an RDF algebra* introduced in [40].

The work on the formal model was as follows:

1. The process of data transformation was divided into two parts: *(i)* schema mapping (creating a mapping between a particular relational database schema and an RDFS ontology) and *(ii)* instance transformation (transforming relational data into RDF documents according to the mapping). Thus the data transformation model has two parts: *schema mapping layer* and *template layer* (see the design rationale in 4.4).
2. The work on the schema mapping layer required:

    - Identification of corresponding parts of relational schema and RDFS.

- Design of a model capable of mapping corresponding concepts from both formalisms.

3. The template layer is for querying RDB by means of RDF. We used a common approach called *query translation* to translate our queries to SQL but we designed a unique query interface in the template layer. Our query has the form of an RDF graph and this graph defines not only the content of a resulting RDF document but also its structure. We can say that our template layer is *an RDF view to a relational database*.

4. We proposed two declarative XML languages based on the formal model. One language is capable of describing schema mapping between relational schema and RDFS and the other is for building RDF documents from relational database. Reasons for choosing XML serialisation syntax are discussed in Section 5.2.

5. We evaluated the completeness of our formal proposal in the following way:

   - Relational completeness of our model was verified using Elsmasri's and Navathe's [72] approach – we proved that the minimal set $\{\sigma, \pi, \times, \cup, -\}$ of relational operators is complete.
   - RDFS completeness was evaluated by enumerating of all supported RDFS features.
   - When considering RDF completeness first the support for the RDF data model was formally proven and then all other supported RDF features were enumerated.

3.6.2 Data transformation system implementation

We defined two partial goals regarding the implementation: *usability* and *performance*. Both are discussed in detail in Chapter 7.

1. *Usability* is guaranteed by the two-layer design which conforms to roles in web development and by a simple user interface (template layer) that hides complexity of semantic web technologies from a user.

2. Our data transformation system is designed as a stream processor - it neither creates a whole resulting RDF model in a memory nor uses any extensive programming library

for RDF manipulation. This novel approach in the field brings high performance as showed in the experimental evaluation.

3.6.3 Experimental evaluation

To compare our approach with others and to prove our concepts, we executed an extensive performance analysis, for which the methodology is described in Chapter 8.

3.6.4 Experimental deployment

We deployed the implemented system in several case studies to test it in a practical environment. The case studies (discussed in Section) informally proved the usability of our approach. We also used the studies to create an experimental RDF knowledge domain for our continued research.

4 Data transformation model

In this chapter we detail and discuss the abstract data transformation architecture and the formal description of the data transformation model.

4.1 Problem specification

As discussed in Chapter 3, the goal of this work is to enable *a transformation of relational data into an RDF document*. The transformation is based on *a mapping that maps a relational database schema into an RDFS ontology*.

The transformation of data from a relational database to an RDF document based on schema mapping is depicted in Figure 4.1.

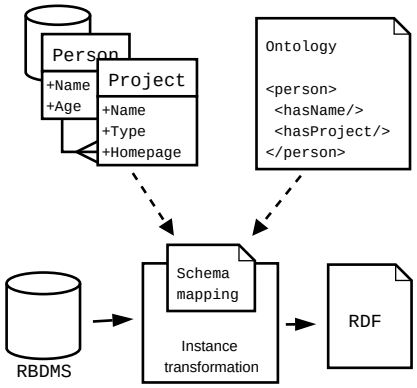


Figure 4.1: Data transformation schema

To be more specific about schema formalisms and data formats in Figure 4.1, we propose following points:

- a schema formalism for a source schema is the relational model,
- source data are stored in an RDBMS,
- a schema formalism for a target schema (i.e. for an ontology) is RDFS, and
- target data are stored in an RDF document.

4.2 Formal definitions

The transformation process discussed in the previous section has two steps:

1. Creating a mapping between a particular relational database schema and an RDFS ontology.
2. Transforming relational data into RDF documents according to the mapping from the first step.

Generally we can identify the two different problems that we will address as: *(i) schema mapping* and *(ii) instance transformation.*

Schema mapping takes two schemas from the same knowledge domain as input and produces a mapping between elements of the schemas that correspond "semantically" to each other.

Instance transformation describes queries that transform values from a data source into a different fixed target data in accordance with the previously defined mapping between schemas.

The following definitions describe these terms more formally.

Definition 4.2.1 (Schema mapping) *Given a source schema* $S_s = (CON_s,\ REL_s,\ RES_s)$ *and a target schema* $S_t = (CON_t,\ REL_t,\ RES_t)$ *describing one knowledge domain. The schema mapping M is a set of mapping elements each of which indicates that certain concepts or relationships of schema* S_s *relate to certain concepts or relationships in* S_t*.*

$$M = (E_c(c_s \rightarrow c_t),\ E_r(r_s \rightarrow r_t) \mid c_s \in CON_s,\ r_s \in REL_s,\ c_t \in CON_t,\ r_t \in REL_t)$$

where

- E_c *is a set of mapping elements between concepts,*
- E_r *is a set of mapping elements between relationships*

and mapping elements are created in accordance with restrictions from RES_s *and* RES_t*.*

For example, a mapping between S1 and S2 (in the example from Section 3.4) could contain a mapping element joining the concept `Person.first_name` with the concept `firstName`, which is a property of the class `Person`. To define the term *mapping element* generally, we can say:

Definition 4.2.2 (Mapping element) *Given a source schema S_s and a target schema S_t. The mapping element* **e** *is a construct that refers to a particular concept or relationship from S_s and to a corresponding concept or relationship from S_t.*

$$e = (ref_s,\ ref_t)$$

where

- *ref_s is a reference to a concept or relationship from a schema S_s,*
- *ref_t is a reference to a concept or relationship from a schema S_t.*

Definition 4.2.3 (Instance transformation) *Given a source schema S_s, target schema S_t and source database D_s that conforms the schema S_s. A mapping M is established between S_s and S_t.*

The instance transformation T is a set of queries Q based on M over the D_s so that they create a target database D_t that conforms to a schema S_t.

Until this point all formal definitions were written in a general fashion using terms such as a schema formalism, schema or database. Since our work uses particular schemas and data formats (see the list in Section 4.1) the following formal descriptions of our data transformation model will reflect these formalisms, which are defined in the following sections.

4.2.1 Relational model

In the relational model [25], data are organised in relations (sometimes called tables).

Definition 4.2.4 (Relation) *A relation r over a collection of sets (domain values) $D_1,\ D_2, \ldots D_n$ is a subset of the Cartesian Product $D_1 \times D_2 \times \ldots \times D_n$.*
Thus, a relation is a set of n–tuples $(d_1,\ d_2, \ldots,\ d_n)$, where $di \in D_i$.

A name and structure of a relation is specified by a relation schema.

Definition 4.2.5 (Relation schema) *Let* $A_1,\ A_2, \dots A_n$ *be attributes with domains* $D_1,\ D_2, \dots D_n$ *then* $R(A_1 : D_1,\ A_2 : D_2, \dots A_n : D_n)$ *is a relation schema.*

And finally we can define a relational database schema.

Definition 4.2.6 (Relational database schema) *A collection of relation schemas is called a relational database schema.*

When comparing described relational model to our schema formalism definition (definition 3.4.2), we can see some common points: *(i)* relation schemas and their attributes are concepts; *(ii)* from relation schema definition we can identify relationships between relation schema and its attributes, moreover, relationships between relations can be expressed by primary and foreign keys; *(iii)* restrictions are defined by integrity constraints.

Relational algebra [26] is a procedural language developed to perform operations on the relational model. Queries in relational algebra are applied to relations and the result of a query is a relation. The minimal set of relational algebra operators is: selection, projection, cartesian product, set-union, and set-difference ($\{\sigma, \pi, \times, \cup, -\}$). The operators take one or two relations as input and give a new relation as a result – the relational algebra is "closed".

4.2.2 RDF data model

The RDF data model [70] is based on the directed labelled graph (DLG), even it differs slightly from DLG because multiple edges between two nodes are allowed and node labels must be unique. The graph consists of nodes that represent resources or literals (strings) and edges that represents properties. Nodes representing resources can be identified by URI or can be blank nodes.

We identify the following sets in RDF data model (from *RAL*, RDF algebra [40]): R (set of resources), U (set of URI references), B (set of blank nodes), L (set of literals), and P (set of properties). At RDF level the following holds for these sets: $R = U \cup B$, $P \subset R$ and U, B, and L are pair-wise disjoint.

Using RDF we can make assertions (or *statements* in RDF terminology) about resources. The statement consists of: *subject*, *predicate*, and *object*. Consisting of three parts, the

statement is sometimes called a *triple*. The subject and object are nodes of the graph, and the predicate is an arc (see Figure 4.2). The subject is a resource being described by the statement, the predicate is a specific property of the subject and the object is a value of the property.

To be more formal (according to [63]):

Definition 4.2.7 (RDF triple) *An RDF triple contains three components:*

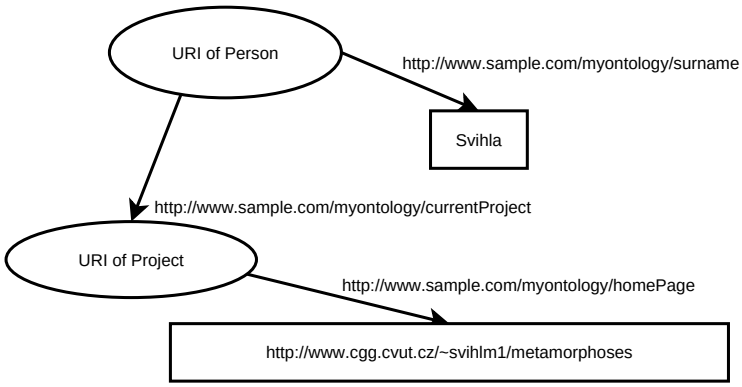
- *the subject, which is a resource (an RDF URI reference or a blank node),*
- *the predicate, which is an RDF URI reference (a property label)*
- *the object, which is a resource (an RDF URI reference or blank node) or a literal.*

An RDF triple is conventionally written in the order of subject, predicate, then object.

When considering RDFS ontology (see the next section) as a schema for an RDF document, resources are instances of ontology classes and predicates are links to ontology properties.

Simple RDF assertions are depicted in Figure 4.2. RDF statements from the picture are based on ontology from Listing 5.1. Ellipses represent URI-identified resources, rectangles are literals and arcs are URI-identified predicates.

Figure 4.2: RDF graph example



Definition 4.2.8 (RDF model (document)) *An RDF model M is a finite set of RDF triples.*

$$M \subset R \times U \times (R \cup L)$$

Graph representation of RDF models can be formalised as follos [40]:

Definition 4.2.9 (RDF graph model) *A graph representing RDF model M is:*

$$G = (N, E, l_N, l_E)$$

where

- $l_N = N \rightarrow R \cup L$,
- $l_E = E \rightarrow P$,
- *N and E denotes nodes and edges, l_N and l_E their labels.*

Following this graph representation, we can also specify basic properties for the nodes and edges [40].

As described in Table 4.1 each node has three basic properties. The *id* of a node represents the (identification) label associated with it. The nodes from the subset of resources that represent the blank nodes do not have an id associated with them. There are two types of nodes: *rdfs:Resource* and *rdfs:Literal*. The *nodeId* provides the unique internal identifier of each node in the graph. *nodeId* has the same value as id for the nodes that have a label, but in addition it gives a unique identifier to the blank nodes. The internal identifier *nodeId* is not available for external use, i.e. it is not disclosed for querying.

Basic property	**Resource** $u \in U$	**Resource** $u \in B$	**Literal** $l \in L$
id	$l_N(u)$	-	$l_N(l)$
type	*rdfs:Resource*	*rdfs:Resource*	*rdfs:Literal*
nodeId	internal id	internal id	internal id

Table 4.1: Node basic properties

Each edge has three basic properties as described in Table 4.2. Compared with nodes, which have unique identifiers, edges have a *name* (label), which may be not unique. There can be several edges sharing the same *name* but connecting different pairs of vertices. The *name* of an edge is (lexically) identified with the *id* of the resource corresponding to the

Basic property	**Edge** e **from** $r \in R$ **to** $o \in R \cup L$
name	$l_E(e)$
subject	r
object	o

Table 4.2: Edge basic properties

property associated with the edge. The *subject* of an edge provides the resource node from which the edge is starting. The *object* returns the resource or literal node to where the edge ends, i.e. the value of the property.

The previous text is a short introduction to RDF data model necessary for this work. RDF specification defines some additional features for RDF as containers, collections or reification, but all of them are based on the data model described above. Complete specification of the RDF model is in [65] or [63]. Moreover, an algebra for RDF data model called *RAL* is proposed in [40] – some definitions in this section come from this proposal.

An RDF document can be serialised using several formats. The most common are RDF/XML [5], N3 [13], NTRIPLES [4], TRIX [76], and Turtle [6]. The RDF/XML is serialisation syntax based on XML. Since XML is very common syntax for sharing data on the web, we use RDF/XML for resulting RDF documents. The sample RDF/XML document is available in Listing 5.12.

4.2.3 RDF Schema (RDFS)

RDF Schema (RDFS) [17] provides a modelling language on top of RDF. It adds new modelling primitives as RDF resources with additional semantics. However, an RDFS model can be handled as a plain RDF model. Basic RDFS primitives are the following: `rdfs:Resource`, `rdfs:Class`, `rdfs:Literal`, `rdf:Property`, `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:subClassOf`, `rdfs:domain` and `rdfs:range`. Their relationships are depicted in Figure 4.3.

Every resource with property `rdf:type` equal to `rdfs:Class` is a class in a RDFS ontology. Every resource with property `rdf:type` equal to `rdf:Property` is a property in a RDFS ontology.

To define RDFS regarding our general definition for a schema formalism (definition 3.4.2), we can say:

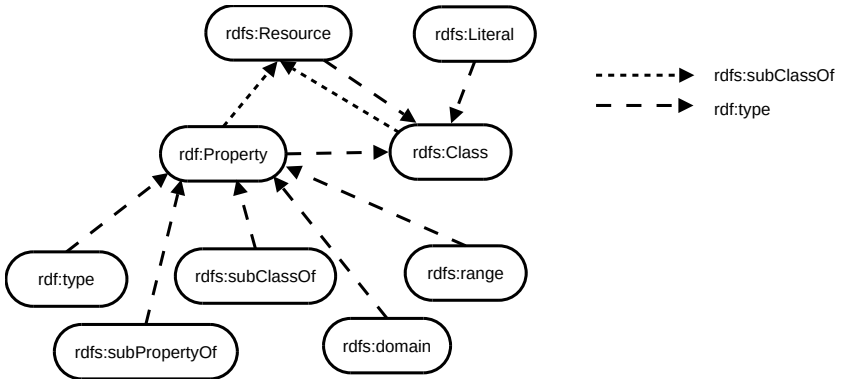


Figure 4.3: RDFS primitives

Definition 4.2.10 (RDFS formalism) *RDFS formalism ϕ_{RDFS} is a triple of sets of concepts, relationships and restrictions*

$$\phi_{RDFS} = (C_{RDFS}\ ,\ R_{RDFS},\ S_{RDFS})$$

where

- C_{RDFS} = {*rdfs:Resource, rdfs:Class, rdfs:Literal, rdf:Property*},
- R_{RDFS} = {*rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range*},
- S_{RDFS} = {}

RDFS formalism provides no primitives for setting constraints on (RDF) data instances. Support for such declarations is provided by richer ontology languages such as OWL (see Section 4.2.4).

A sample of RDFS ontology can be found in Listing 5.1 (Chapter 5).

4.2.4 OWL (Web Ontology Language)

OWL (Web Ontology Language) [88] currently holds the highest position of ontology formalisms for the semantic web. It is based on RDFS (see the previos section), which is an extension of the RDF. It introduces constructs for an ontology definition: classes, properties, literals, resources, corresponding relationships and restrictions. OWL has three

sub-languages with different expressive and inference power: OWL Full, OWL DL, and OWL Lite.

The formal definition of OWL language, based on definition 3.4.2, follows bellow. It does not describe all concepts and relationships available in OWL because the list, available in [31], is rather long. However, we can provide an overview of OWL features.

Definition 4.2.11 (OWL formalism) *OWL formalism ϕ_{OWL} is a triple of sets of concepts, relationships and restrictions*

$$\phi_{OWL} = (C_{OWL}\ ,\ R_{OWL},\ S_{OWL})$$

where

- C_{OWL} = {*owl:Class, owl:Thing, owl:ObjectProperty, owl:DatatypeProperty, owl:TransitiveProperty, owl:SymmetricProperty...*}
- R_{OWL} = {*owl:equivalentClass, owl:equivalentProperty, owl:inverseOf, owl:disjointWith, owl:intersectionOf,...*}
- S_{OWL} *is set of restrictions that can be expressed by using built-in OWL primitives as owl:Restriction, owl:FunctionalProperty, owl:InverseFunctionalProperty, owl:maxCardinality, owl:minCardinality, owl:cardinality, owl:allValuesFrom, owl:someValuesFrom or owl:hasValue*

The OWL is not the main schema formalism in our work – we rather focus on RDFS as a basic ontology formalism above RDF. However, some OWL features are naturally supported by our data transformation model, which is designed to be extensible to OWL in the future (see the future work Section 10.3).

The OWL support is discussed in Section 6.4 and summarised in the appendix B.

4.3 Discussion on used formalisms and data formats

This work focuses on the transformation of relational data into RDF (as detailed in 4.1). The formalisms for these data formats are relational model and RDFS. In this section we discuss similarities between these formalisms and data formats and we identify corresponding structures.

The way we map these structures together will be detailed later in Section 4.4.

4.3.1 Schema mapping

There are two basic elements in the RDFS formalism (see Section 4.2.3): `rdfs:Class` and `rdf:Property`. All ontology classes have the type (`rdf:type`) of the former and all properties have the type of the latter. These classes and properties form concepts and their relationships in RDFS ontologies. Properties naturally belongs to classes; we can see the class as a abstract concept that holds a set of properties.

In the same manner we can identify two similar concepts in the relational model: relation schema and attribute. By its definition (4.2.5) the relation schema is a set of attributes.

Using this similarity, we can say that

Definition 4.3.1 (Schema correspondence) *having relational database schema and RDFS ontology, it holds*

- *a relation schema corresponds to a class in an RDFS ontology and*
- *an attribute in a relation schema corresponds to a property in an RDFS ontology.*

Thus, when building schema mapping (definition 4.2.1) we can use mapping elements (definition 4.2.2) to map ontology classes to corresponding relation schemata and to map ontology properties to corresponding attributes.

Of course, one particular class in a particular ontology does not have to correspond with one particular relation schema in a particular relational database. For example, the attributes corresponding to properties of one particular class could be distributed over many relation schemata. However, using relational algebra operators we can identify subset of relational database schema and this subset will be a relation schema (see the following theorem and its proof). This way we can map an ontology class to any corresponding part of the relational database schema.

Theorem 4.3.1 *Using relational algebra operators it is possible to identify any part of a relational database schema as a relation schema.*

Proof 4.3.1 (of theorem 4.3.1) *Using definitions of relational model and algebra (Section 4.2.1):*

Relational model	**RDFS**
Relation schema	Ontology class
Attribute	Ontology property
Relationship between relation schema and its attributes	(Implicit) relationship between class and property
Relationship between two relation schemata	(Implicit) relationship between two classes

Table 4.3: Schema correspondences

1. *The relational algebra operators take one or two relations as input and produce a new relation as a result (relational algebra is closed).*

2. *Each relation has its relation schema.*

3. *The relational algebra operations can be combined.*

Thus it holds that the relational algebra operators take one or two relation schemata as input and produce a new relation schema as a result. By combining operations it is possible to identify any part of a relational database schema as a relation schema, QED.

Consequences of this reasoning are detailed in Section 4.4.

Moreover, there are relationships in both schemas that correspond to each other. A relationship between a relation schema and its attributes is obvious in the relational model and a very similar relationship between ontology class and ontology properties can be found in RDFS. The former relationship is explicit: an attribute belongs to a relation schema. In the latter case, classes and properties are not connected explicitly by RDFS specification. However, this relationship can be modelled by the property `rdfs:domain`.

Another relationship is between two relation schemata in the relational model or two classes in RDFS. The former case is often modelled by foreign keys, and the latter case can be implicit or suggested by properties `rdfs:domain` and `rdfs:range` on a property connecting the two classes.

The identified correspondences are summarised in Table 4.3. The issue of schema constraints is discussed in Section 4.5.3.

Relational model	**RDF**
N-tuple (row)	Resource
Attribute name	Property
Attribute value	Literal

Table 4.4: Data format correspondences

4.3.2 Instance transformation

Using the same logic we can examine relational data and RDF. A relational database consists of relations, which consist of n-tuples or rows. Each row consists of a set of values. The RDF model is a set of triples – each triple consists of a subject node (which is a resource), a property edge, and an object node (which is a property value–a literal or another resource). We can say that the property assigns its value to the resource in the RDF triple. It is very similar in the relational model – the attribute assigns its value to the row. The corresponding elements are described in Table 4.4.

This natural similarity is also described in the next theorem:

Definition 4.3.2 (Data model correspondences) *Having relational model and RDF data model it holds that:*

- *a row from a relation corresponds with an RDF resource,*
- *an attribute (column) name corresponds with an RDF property,*
- *an attribute value in the row corresponds with an RDF property value.*

This correspondence is not new, it was presented very early in [10] and is also a base for the so-called *naïve approach* in [7]. However, in both cases the schema for the RDF was not considered and thus the naïve approach was merely an export of relational data to RDF graph. We use this correspondence just as a starting point for further reasoning.

4.4 Design rationale

Following the two steps of a data transformation process proposed in Section 4.2, we divided the data transformation model architecture into two separate parts: the *schema mapping layer* and the *template layer*, as depicted in Figure 4.4.

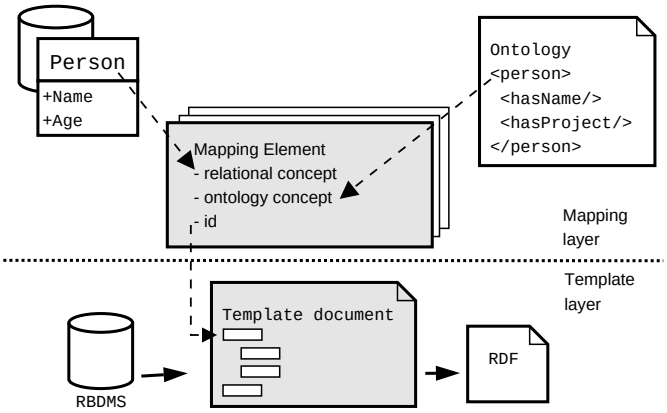


Figure 4.4: Data transformation architecture

In the schema mapping layer, which corresponds to the schema mapping process (definition 4.2.1), a relational schema is mapped into an ontology. This is done by *mapping elements* that join relating parts of a database schema and ontology. A set of such elements creates a *schema mapping document*, which can be described by *schema mapping language*. Each mapping element has a unique id within a mapping document in order to be referenced from the template layer.

The template layer corresponds with the instance transformation process (definition 4.2.3). On this level references to mapping elements from the mapping layer are combined in order to create a *template document*, which is a query for a transformation of relational data into an RDF document.

This process is illustrated in Figure 4.4. Formal issues of the architecture are detailed in the following subsections and the design and features of the languages are discussed in Section 5.

4.5 Schema mapping layer

The schema mapping layer is a level of the schema mapping in our data transformation model. Since we divided schema elements into three categories (concepts, relationships, and constraints – see definition 3.4.1), in the following text we address these three areas of mapping in turn.

4.5.1 Concept mapping

We identify two basic schema concepts: *classes* (relations in a relational schema or classes in RDFS ontology) and *properties* (attributes in a relational schema or properties in RDFS ontology). Thus we have two basic elements for concept mapping: *(i)* a *mapping class* and *(ii)* a *mapping property*.

Mapping class

The mapping class connects an ontology class (an ontology concept) with a corresponding subset of a relational database schema (a relational concept). An ontology class is referenced by a full class name with namespace. A relational database schema subset is referenced by the relational algebra operators.

Definition 4.5.1 (Mapping class) *A mapping class C_m is a triple*

$$C_m = (r_s,\ c_o,\ id \mid r_s\ \subset S_s,\ c_o\ \in\ O_t)$$

where

- *r_s is a relation schema – a subset of relational database schema – referenced by the relational algebra operators,*
- *c_o is a referenced ontology class,*
- *S_s is a source relational database schema,*
- *O_t is a target ontology and*
- *id is a unique identifier of the mapping class C_m.*

We argue that relational algebra operators are not only useful for fetching data from relational database but can also serve as identifiers of relational concepts in a relational database schema (see theorem 4.3.1). A relational algebra query returns a relation and relation schema of this relation is a relational concept referenced by a mapping class.

This approach is illustrated in Figure 4.5. The `(Equi) Join` operation is used to interconnect two relation schemas from a relational database creating one concept `RelPerson` (a relationship between these relation schemas is 1:1 or 1:N; relationships M:N are discussed elsewhere).

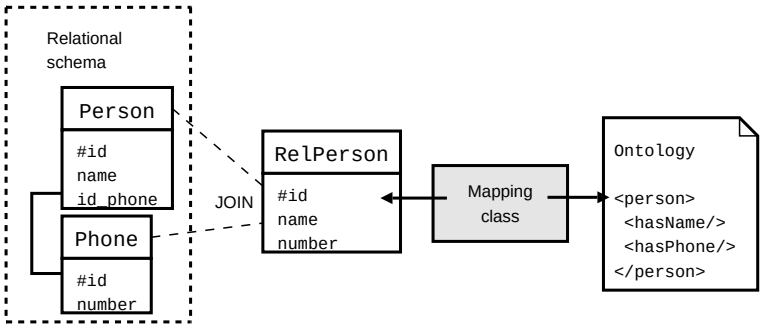


Figure 4.5: Mapping class

Mapping property

A mapping property connects an ontology property with a corresponding relation schema attribute.

A mapping property always belongs to a mapping class. Thus a relation schema attribute belongs to a relation schema referenced by a mapping class and an ontology property may belong to an ontology class referenced by a mapping class. This model follows the relational model pattern, where attribute always belongs to the relation schema. This relationship and its mapping is detailed in the next Section (4.5.2).

Definition 4.5.2 (Mapping property) *There is a mapping class C_m connecting concepts from a relational database schema S_s and an ontology O_t. A mapping property P_m that belongs to C_m can be defined as:*

$$P_m = (A_s,\ p_o,\ id \mid A_s\ \in r_s,\ p_o\ is\ property\ of\ c_o)$$

where

- *A_s is a referenced relation schema attribute,*
- *p_o is a referenced ontology property,*
- *r_s is a relation schema from S_s referenced by C_m,*
- *c_o is a class from O_t referenced by C_m and*
- *id is a unique identifier of the mapping property P_m.*

The reference to the relation schema attribute is optional in the mapping property. This reflects the fact that property in RDF can have values both literal (the value of the relation schema attribute) or resource (another mapping class instance). This is not distinguished in RDFS but in OWL the properties are referred to as `owl:datatypeProperty` in the former case and `owl:objectProperty` in the latter.

Thus we can say that:

Definition 4.5.3 (Datatype mapping property) *Datatype mapping property is a mapping property with reference to the corresponding relation schema attribute.*

and

Definition 4.5.4 (Object mapping property) *Object mapping property is a mapping property without reference to the corresponding relation schema attribute.*

Mapping attribute

In RDF/XML syntax, RDF resources and properties can have attributes that describes them. For example, the RDF attribute `rdf:about` specifies a resource URI and the attribute `rdf:datatype` is in place to specify a datatype for a literal value of a property.

The mapping attribute element is to add RDF attributes to produced RDF resources or properties on the template layer. The mapping attribute always belongs to a mapping class or property. It consists of an RDF attribute name, relation schema attribute, and additional text information. The combination of the latter two elements represent an RDF attribute value.

Definition 4.5.5 (Mapping attribute) *There is a mapping class C_m that refers to a relation schema r_s. A mapping attribute A_m that belongs to C_m or to any of its mapping properties can be defined as:*

$$A_m = (name, A_s,\ text \mid A_s\ \in r_s)$$

where

- *name is an RDF attribute name describing the mapping attribute,*
- *A_s is a referenced relation schema attribute,*

- *r_s is a relation schema from S_s referenced by C_m,*

- *text is a source of an additional textual information.*

4.5.2 Relationship mapping

A relationship between a relation schema and its attributes is obvious in the relational model. A very similar relationship between ontology class and ontology properties can be modelled by the property `rdfs:domain` (see Section 4.3.1 for details). To map this relationship we simply link mapping properties to their corresponding mapping class; every mapping property is assigned to some mapping class, as described in the definition 4.5.2.

The relationship between two relation schemata modelled by foreign keys in the relational model can be mapped into the relationship between two classes in RDFS. In RDFS any two classes can be in this relationship, however, it can be explicitly modelled by properties `rdfs:domain` and `rdfs:range` on a property connecting the two classes. To map these corresponding relationships there is a mapping element called *mapping condition* that connects two mapping classes. Each mapping condition belongs to some mapping class.

Definition 4.5.6 (Mapping condition) *The mapping condition states that its parent mapping class (C_m^1) is in N:1 relation with some other mapping class (C_m^2).*

A connection between two classes is made by a relational algebra operation *join*, which joins the two classes. Later, on the template layer, the mapping condition performs a *select* operation on a relation from the C_m^1, which results in selecting tuples that relates to the C_m^2.

N:M relationship is modelled by two 1:N relationships in the relational model and can be expressed by a pair of mapping conditions – each in one of related mapping classes.

This abstract definition is practically described in Section 5.3.1.

Regarding RDFS we identified three RDFS primitives (in the definition 4.2.10): `rdf:type`, `rdfs:subClassOf`, and `rdfs:subPropertyOf`, besides previously mentioned relationships. All of these elements are to express relationships between instances, classes, and properties. They have no counterpart in the relational model and thus we don't map them. However, they are integrated into our data transformation model. The first one, `rdf:type`, is used in the template layer to assign an ontology class to an RDF instance. The other two are

ontology issues that form a base for RDFS reasoning. They can be a part of the resulting RDF document since an ontology can be linked to the document.

4.5.3 Schema constraints

We assume that the relational database content conforms to the database schema constraints, which eliminates the need to consider these constraints in our mapping. On the other hand, we have no RDFS primitives that create ontology constraints (definition 4.2.10), thus there are no ontology restrictions for resulting RDF documents. Because of this, we do not consider schema constraints here.

The question of schema constraints is discussed in this work because we plan to extend our model with the OWL formalism, which supports constraints (see Section 4.2.4 and 10.3). Thus our schema formalism definition (3.4.2) and data transformation definition (3.4.5) contain the schema constraint concept for future extensibility.

4.5.4 Mapping document

A set of mapping elements that express a mapping between a particular relational database schema and an ontology is addressed as a *mapping document*.

Definition 4.5.7 (Mapping document) *A mapping document is a set of mapping elements that map a source schema S_s to a target schema S_t, written in a particular mapping language.*

Practically, elements can be serialised using many different data models or languages. We developed our own *schema mapping language*, based on XML. Details of this language are proposed in Section 5.3.1, and an example of a mapping document can be found in the Listing A.1.

4.6 Template layer

A template layer corresponds to the instance transformation. On this level a query is passed to a relational database through the mapping layer. Returned data instances and ontology elements from a mapping document are composed to a resulting RDF document.

The query has the form of a template document, which is a tree consisting of template elements. Template elements are references to mapping elements from a particular mapping document. Referenced mapping elements translate the query to relational database queries and data from returned relations are composed according to the template document structure. This means that the template documents form a query but also specify a structure of resulting data.

In the previous section we defined two concept mapping elements – *(i)* the *mapping class* and *(ii) mapping property*. These mapping elements are referenced by two main template elements: the former by *template instance* and the latter by *template property*. A combination of these elements forms a template document.

Moreover, there are additional template elements that join relating classes (e.g. *template condition* and *template variable*) and they will be discussed in Section 5.3.2.

Definition 4.6.1 (Template element) *A template element is a parametrised reference to a mapping element. The mandatory parameter of the template element is an identifier of the particular mapping element.*

A template document is a template for generation of an RDF document. It is a set of references to mapping elements and thus it is always dependent on some mapping document. This means that a resulting RDF document is based on a particular ontology and will be generated from a particular relational database schema. From that point of view it can be said that the template document is an abstract RDF document that can be instantioned by particular data fetched from a relational database.

Definition 4.6.2 (Template document) *A template document is a tree composed of template elements that refer to a particular mapping document.*

As mentioned above, a template document is a tree and its nodes are template instances and properties. In order to create a valid query for mapping layer, we must define a set of rules for the template document tree.

Definition 4.6.3 (Template document rules) *The rules for creating a template document tree are as follows:*

- *a document root can contain only template instance nodes,*
- *a template instance node can contain template property nodes*
- *a template property node referencing a mapping object property must contain a template instance node*
- *a template property node referencing a mapping datatype property is a leaf*

An example of a template document tree is depicted in Figure 4.6.

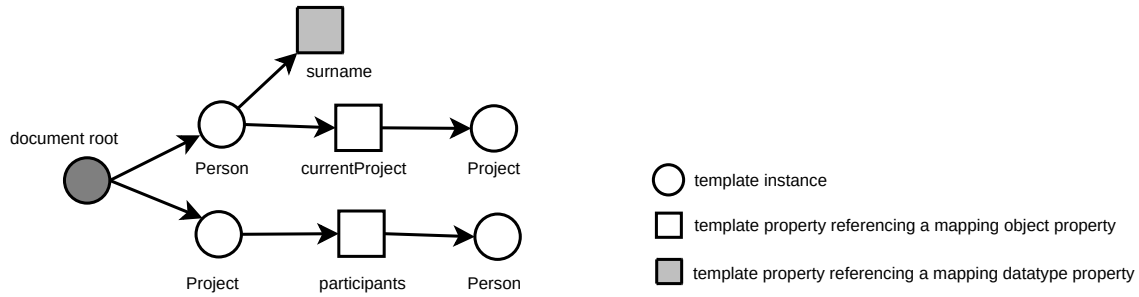


Figure 4.6: Example of a template document tree

During the data transformation, template elements are translated to RDF resources (nodes of RDF graph) and RDF properties (edges of RDF graph). The entire transformation process is described in the next section. The result of the data transformation based on the template from Figure 4.6 can be, for instance, the RDF graph depicted in Figure 4.7.

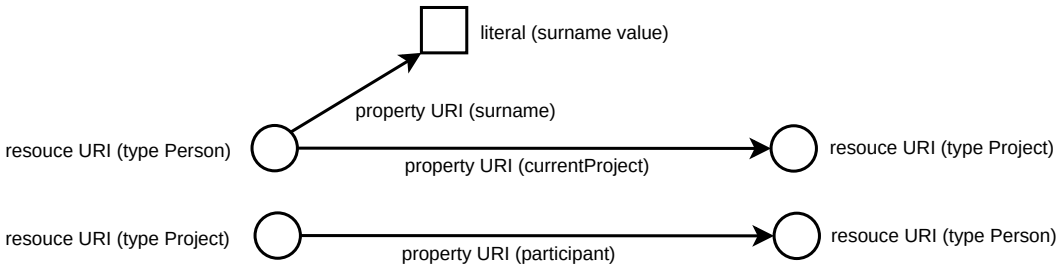


Figure 4.7: Sample of a resulting RDF graph

4.7 Putting both layers together

The schema mapping and template layer work together in order to enable the data transformation process. A mapping document joins concepts and relationships between a relational database schema and RDFS ontology and addresses data instances in a relational database. A template document refers the mapping document using template elements. Template elements form a query for the schema mapping layer, which translates it to a relational database query and fetches data from a RDB. These data and relating ontology concepts are returned to the template layer where they are composed to a resulting RDF document.

The binding between both layers is depicted in Figure 4.8. Each template element in the picture is connected to a particular ontology concept through its mapping element. On the other hand, it can be also connected to related concepts from a relational database schema and their data instances stored in a database. Since produced RDF elements are based on template elements, they are created according to RDFS ontology and relational data.

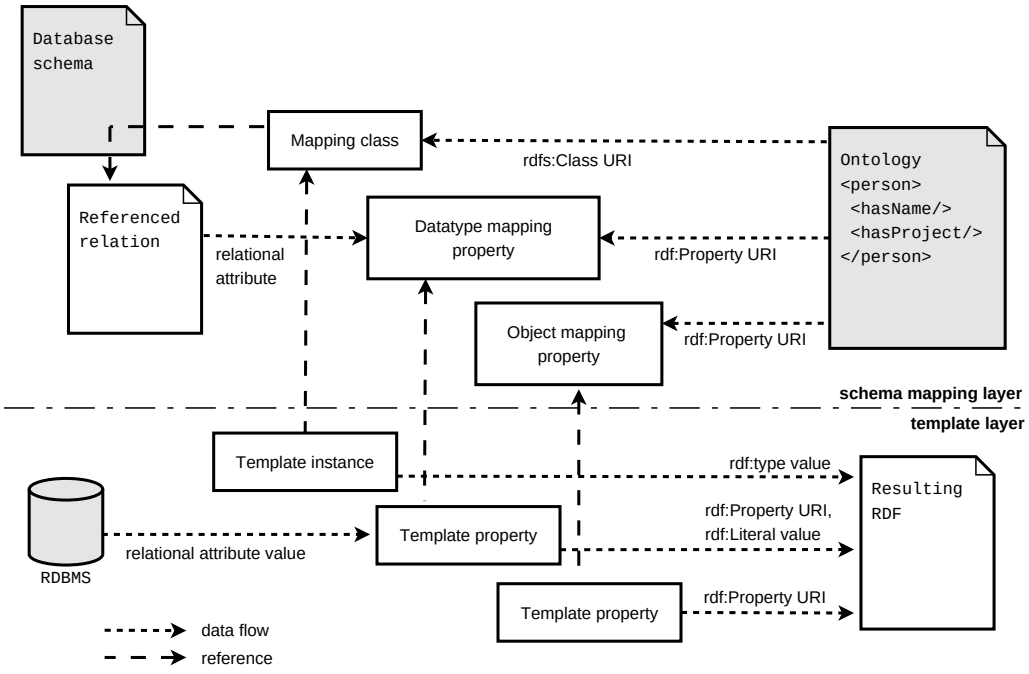


Figure 4.8: The binding between the schema mapping and template layer

A template instance passes `rdf:type` to each generated RDF resource from a referenced mapping class. A template property can refer to a datatype or object mapping property. In the former case, a property URI from ontology and the value of a corresponding literal

from a database are passed to a resulting RDF property. In the latter case only a property URI is passed.

Moreover, any mapping class or property can contain a set of its mapping attributes (attributes are not in Figure 4.8). These mapping attributes specify RDF attributes on RDF resources or properties produced on the template layer. Since a mapping attribute can contain reference to relational database schema, RDF attribute values can be generated from the relational database. A URI for RDF resource or datatype of RDF property value can be described this way.

The abstract process of the data transformation on the described model is detailed in the following algorithm:

Algorithm 4.7.1 (Abstract data transformation process)

Input: mapping document M, template document T referencing M, relational database D
Output: RDF document

Step 1: **parse** T
Step 2: **for each** template element E_t from T do
identify corresponding mapping element E_m from M
create RDF element:
– assign RDF element name from RDFS ontology stored in M
– (if necessary) assign RDF element value from RDB referenced by M
end for
Step 3: **compose** created RDF elements into RDF graph according to the structure of T

Implementation details of this process are described in Chapter 7. Features of the resulting RDF graph are discussed in the following section.

4.8 Discussion: result of the data transformation

The result of the data transformation should be an RDF document (graph), which conforms a given RDFS ontology (from definition 3.4.5). This section investigates our model from this perspective.

Theorem 4.8.1 *The result of the data transformation can be an RDF document (graph) based on a given ontology, when it holds:*

1. *an RDF triple can be composed (in the data transformation process),*
2. *a resulting RDF document can be a graph and*
3. *all produced RDF resources and properties can be typed by the corresponding RDFS ontology.*

To show that our model can produce RDF graph we will prove all partial conditions of the theorem 4.8.1.

Proof 4.8.1 (that an RDF triple can be composed in our model.) *According to definition 4.2.7, an RDF triple consists of a subject (RDF resource), a predicate (URI reference), and an object (RDF resource or literal). RDF resource can be identified with URI, otherwise it is a blank node.*

1. *The template layer can produce an RDF resource (an object of a triple) by a template instance (Section 4.6) which refers to a mapping class (definition 4.5.1).*
2. *URI can be added to a RDF resource by assigning the mapping attribute (definition 4.5.5) to the mapping class.*
3. *A template property referencing a mapping property (definition 4.5.2) can be added to the template instance – it creates RDF predicate with or without literal value (definition 4.6.3, Figure 4.6)).*
4. *If a template property contains a literal value it creates an object of a triple (definition 4.6.3).*
5. *If a template property does not contain a literal value, another template instance can be added to the template property and it will create its value – the object of a triple (definition 4.6.3).*

To sum up the enumerated points, it is possible to create a subject, a predicate, and an object (both literal and resource) and resources can contain URI. Thus it holds that an RDF triple can be composed in our data transformation model, QED.

Proof 4.8.2 (that a resulting RDF document can be a graph in our model.)

1. *A template document is a tree (definition 4.6.2, Figure 4.6).*
2. *A resulting RDF document follows its tree structure of a template document (Section 4.6).*
3. *RDF resource nodes can be identified by URI (using a mapping attribute, definition 4.5.5).*
4. *In the case that two nodes have the same URI, they describes one RDF resource (from RDF specification, [5]).*

Because of the last point, a cycle can appear in the resulting RDF document and thus **it can be a graph**, *QED.*

Figure 4.9 illustrates this possibility. The depicted RDF graph is based on the template document from Figure 4.6.

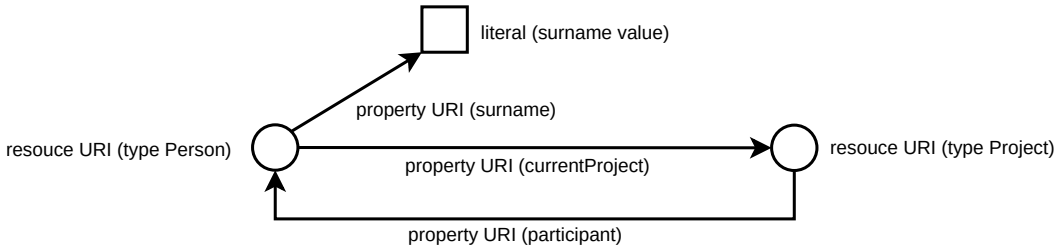


Figure 4.9: Sample of a resulting RDF graph with cycle

The following reasoning is to prove that all produced RDF elements can linked to corresponding ontology concepts in our model.

Proof 4.8.3 (Connection between resulting RDF and ontology) *Regarding the binding between the template layer and RDFS ontology via the mapping layer (Section 4.7, Figure 4.8) it holds:*

1. *an* `rdf:type` *value from the ontology can be assigned to a produced RDF resource,*

2. URI reference can be assigned to a RDF predicate from the ontology.

This means that produced RDF resources or predicates can be linked to corresponding ontology classes and properties, QED.

Lemma 4.8.1 (based on the proof 4.8.3.) *Since all elements of the resulting RDF graph can be linked to RDFS ontology classes and properties, they are also bound to other features from the ontology (e.g. subclass or subproperty hierarchy).*

Proof 4.8.4 (of theorem 4.8.1) *The proofs 4.8.1, 4.8.2 and 4.8.3 prove all partial conditions of the theorem 4.8.1 considering our model. Thus, it holds that the result of our data transformation can be an RDF document (graph) based on a given ontology, QED.*

4.9 Summary

In this chapter we introduced the architecture of our data transformation model and described all of its parts. We also showed how these parts together enable the data transformation based on the schema mapping. Moreover, we proved that a result of the data transformation is an RDF graph conforming to the RDFS ontology.

Returning to the formal goal of our work (definition 3.4.5), we intended to design a mapping model to perform the operation of **data transformation**, which is able to create a target database D_t conforming to a schema S_t from data stored in a source database D_s, which conforms to a schema S_s.

$$t_{s,t} \ : \ (D_s \ \rightarrow D_t) \mid D_s \ conforms \ S_s, \ D_t \ conforms \ S_t$$

where S_s is relational database schema, S_t is an RDFS ontology, D_s is a relational database and D_t should be a resulting RDF document.

The proposed two-layer data transformation model solves the mentioned problem. Input for the mapping layer are a relational database schema (S_s) and RDFS ontology (S_t) and its output is a mapping document. The template layer takes this mapping and relational database (D_s) and produces an RDF document (D_t). Resulting D_t conforms S_t because it is build on a mapping document referencing S_t.

The completeness of our approach according to used formalisms (relational model, RDF and RDFS) is demonstrated in Chapter 6.

5 Data transformation languages

To enable the schema mapping and data transformation described in the previous chapter, we developed two XML languages – one for the schema mapping documents and another for template documents.

This chapter describes informal semantics and syntax of both languages.

5.1 Running example

In the previous chapter, we discussed our model in such a general fashion that it was not possible to illustrate the abstract theory with real examples. In this chapter, we describe two XML languages that materialise our theoretical concepts.

Here we provide a base to the running example which will illustrate concepts described in the further text. The complete mapping document for both schemas can be found in the Appendix A.

Relational database schema

When referring to database schema in this chapter, we use such common terms as table, column and row. In the running example we suppose a small database schema, consisting of four tables (primary keys are marked by `#`):

```
PERSON(#id, id_department, username, first_name, family_name)
PROJECT(#id, web)
PERSON_PROJECT(person_id, project_id)
DEPARTMENT(#id, name)
```

The table `DEPARTMENT` is in 1:N relationship with the table `PERSON`, and the foreign key is `PERSON.id_department`. The tables `PERSON` and `PROJECT` have an M:N relationship, and the table `PERSON_PROJECT` is to join them.

RDFS ontology

The RDFS ontology that specifies a vocabulary for resulting RDF in our running example is listed in 5.1.

Listing 5.1: RDFS ontology sample

```
<rdfs:Class rdf:ID="Person">
  <rdfs:label>Person</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="surname">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="hasDepartment">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="currentProject">
  <rdfs:range rdf:resource="#Project"/>
  <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>

<rdfs:Class rdf:ID="Project">
  <rdfs:label>Project</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="participants">
  <rdfs:domain rdf:resource="#Project"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="homepage">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</rdf:Property>
```

The concepts from this ontology have no namespace prefix in the following examples.

5.2 Comment on syntax

The syntax of our mapping languages is XML. In the following list we provide some reasons to choose an XML language:

- XML was developed to represent these kind of documents and it is very common in the web community (and also in the semantic web community).
- An XML syntax can be analysed and processed using the Document Object Model (DOM), which can be parsed and manipulated by a number of free and commercial libraries, providing a good base for a mapping framework implementation.

- An XML syntax is human readable – a human user can edit XML documents manually.

- One serialisation format for RDF is XML (RDF/XML). If a template document is a template for an RDF document, it is intuitive for a human user to build it as an XML tree since RDF can also be imagined as a tree.

5.3 Language semantics overview (informative)

The mapping and template elements introduced in Chapter 4 are the base for our schema mapping and template language. The semantics of the language primitives derived from the formal model are detailed in this section.

5.3.1 Schema mapping language

Schema mapping language is used to describe the mapping between a database schema and given ontology. It can describe elements, which map concepts of a database schema to classes and properties of an ontology. These *concept elements* are later used in a template layer. In addition to the concept mapping elements, there are control structures such as conditions and variables, which define relations between concepts. These *relationship elements* are also used in a template layer and they control RDF production. Moreover, the language contains a set of *general elements* for RDF creation. Their semantic is straightforward; they are not detailed here but only in the the language syntax Section (5.4).

In short, we can say that the schema mapping language is designed to:

- describe mapping elements between a database schema and structure of a given ontology,

- enable control over these elements,

- store information for making a complete RDF document,

- establish connection with a specific relational database.

To reflect *mapping class*, *property*, *condition* and *attribute* proposed in Section 4.5 there are language constructs `Class`, `Property`, `Condition` and `Attribute`. In addition, there

is the element `Variable`. The hierarchy of these elements is illustrated in Figure 5.1 and their description follows.

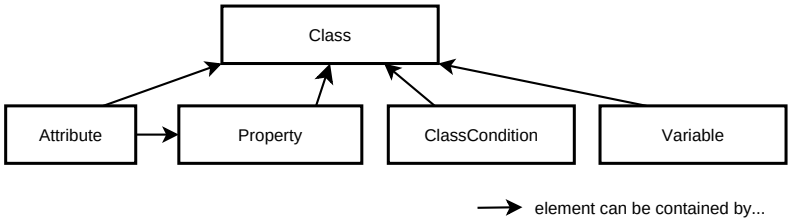


Figure 5.1: Schema mapping language elements

Element `Class` (*mapping class*)

The mapping class is a basic element of the language to express a relationship between a subset of RDB schema and corresponding ontology class. An ontology class is referenced by a full class name with a namespace, and a relational database schema concept is referenced by a SQL query. Using SQL we can address one or more database tables as one concept. The second reason for choosing a SQL query on this level is because a template layer can use it for fetching data from a particular relational database.

A mapping between the ontology class `Person` with the relevant concept from relational schema can be expressed by the following mapping class:

Listing 5.2: Mapping class example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department" />
```

Attribute `templateName` defines a unique ID for the element, `rdfLabel` refers to an RDFS ontology class, and `sql` specifies SQL query that addresses a relational concept corresponding to the ontology class. In this particular SQL query a *join* operation is used to interconnect more tables from a RDB schema creating one concept.

Element `Property` (*mapping property*)

The `Property` element always belongs to a `Class` element and joins an ontology Property with a corresponding database column, which belongs to the table referenced by the enveloping `Class`.

Listing 5.3: Datatype mapping property example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
  <Property templateName="surname" rdfLabel="surname" sqlName="
     family_name" />
</Class>
```

The Listing 5.3 describes a *datatype mapping property* (definition 4.5.3) with an ID `surname` (in the attribute `templateName`), which interconnects an RDFS property `surname` (in the attribute `rdfLabel`) with a database column `family_name` (in the attribute `sqlName`).

An *object mapping property* (definition 4.5.4) can be described in a very similar way by omitting an attribute `sqlName` (Listing 5.4), since object mapping properties do not refer relational attributes.

Listing 5.4: Object mapping property example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
  <Property templateName="currentProject" rdfLabel="currentProject" />
</Class>
```

Element `Attribute` (*mapping attribute*)

The element `Attribute` materialises an abstract mapping attribute from the definition 4.5.5. It can be contained by `Class` or `Property` and adds RDF attributes to them. An `Attribute` element must contain an attribute `rdfLabel` that denotes the name of the RDF attribute. The value of an RDF attribute can be specified by attributes `prefix` and `suffix` that are literals and by a `sqlName` that refers to a database column similarly as in the `Property` element.

Listing 5.5: Mapping attribute example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
  <Attribute rdfLabel="rdf:about"
     prefix="http://webing.felk.cvut.cz/people/" sqlName="username"/>
  <Property templateName="surname" rdfLabel="surname"
     sqlName="family_name">
    <Attribute rdfLabel="rdf:datatype"
       prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
</Class>
```

The Listing 5.5 shows two `Attribute` elements. The first, contained directly by the `Class` element, defines an RDF attribute `rdf:about` as URI reference for each RDF instance of

this mapping class. The URI (attribute value) will consist of `prefix` and value selected from the `username` database column. The other one specifies datatype of the property by adding `rdf:datatype` attribute to it.

Element `ClassCondition` (*mapping condition*)

The mapping element `ClassCondition` describes a relationship between two mapping classes. A mapping condition always belongs to some mapping class and it can indicate that its parent mapping class (further as C_m^1) is in N:1 relation with some other mapping class (further as C_m^2).

In our running example, there are two concepts – `Person` and `Project` – by which a person can undertake several projects. This is practically modelled by the relationship between tables `PERSON` and `PERSON_PROJECT` in the database schema.

Listing 5.6: Mapping condition example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
   <ClassCondition templateName="projectId" whereString="p.id = pp.
      person_id and pp.project_id =" tableString="person_project pp" />
   <ClassCondition templateName="username" whereString="p.username =" />
</Class>
```

The first element `ClassCondition` in Listing 5.6 models this relationship. Using attribute `tableString` it adds table `PERSON_PROJECT` to the parent mapping class *SELECT* query then adds a condition to the query by attribute `whereString`. Applying this condition, the resulting query is:

```
SELECT * FROM person p, department d, person_project pp
WHERE d.id = p.id_department AND p.id = pp.person_id AND pp.project_id =
```

The condition connects tables from both classes using a primary key of the C_m^1 with a foreign key from C_m^2 and creates an opened condition, which can be completed later in the template layer by employing `Variable` element from the class C_m^2. Applying a such mapping condition on the C_m^1 will result in selecting rows that relates to the C_m^2.

Another way of using `ClassCondition` is when an attribute `tableString` is omitted. In this case the query is

```
SELECT * FROM person p, department d
WHERE d.id = p.id_department AND p.username =
```

and the query condition compares a primary key of C_m^1 with a literal. This way a template layer can drive the instance production. Both samples are detailed more in Section 5.3.2.

Element `Variable`

The `Variable` element is to expose a single value from an enveloping mapping class. This is used later in the template document in order to connect instances from different mapping classes. The sample of `Variable` element is in Listing 5.7, and its further semantics and usage is discussed in Section 5.3.2.

Listing 5.7: Mapping variable example

```
<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
   <Variable templateName="personIdVariable" sqlName="p.id"/>
</Class>
```

5.3.2 Template language

To serialise a template document we developed our own *template language*, based on XML.

The template language contains a small set of elements: `PutInstance`, `PutProperty`, `Condition` and `Variable` (their hierarchy is depicted in Figure 5.2). They reflect abstract template layer concepts defined in Section 4.6.

`PutInstance` materialises the *template instance* concept, `PutProperty` is for the *template property*. These two elements can be composed to a tree-based template document in order to create a template for an RDF document.

`Condition` and `Variable` reflect the *template condition* and *template variable*. They can be added to a `PutInstance` node and they drive a production of RDF instances.

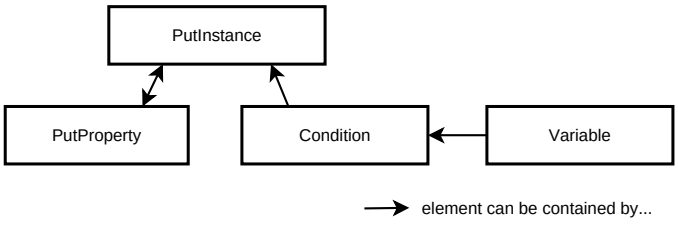


Figure 5.2: The template language elements

All of these elements reference corresponding elements from a particular schema mapping document using the attribute `name`, which connects with a `templateName` attribute in the schema mapping elements. Using SQL queries from mapping elements, a template document can be considered for an RDF view of a source RDB.

Informal semantics of these elements, including examples, are described in the following paragraphs. Their normative syntax is detailed in Section 5.5. All of the following examples are based on the mapping document from the Appendix A.

Element `PutInstance` (*template instance*)

A `PutInstance` element refers to a particular mapping class. When used in a template document, it selects data from a relational database according to a SQL query from a mapping class. Each row (a tuple) of a returned relation will be transformed into one RDF instance. An RDF instance is generated with all its attributes defined in a corresponding mapping class.

Listing 5.8: Template instance example

```
<PutInstance name="person" />
```

The code fragment from Listing 5.8 places all persons from a table `PERSON` to an RDF document. Each produced RDF resource will contain RDF attribute `rdf:about`.

Element `PutProperty` (*template property*)

A `PutProperty` element refers to a particular mapping property. It is always embedded in a `PutInstance` element. When used in a template document, it puts an RDF property with RDF attributes into produced RDF resources. If a `PutProperty` refers to a *datatype mapping property*, a value of the property is selected from a relational database (see Listing 5.9). If the element refers to an *object mapping property*, it must contain another `PutInstance` (this is further detailed in a later section) and nested RDF resource(s) will form a value(s).

Listing 5.9: Template property example

```
<putInstance name="person">
   <putProperty name="surname" />
</putInstance>
```

The example in Listing 5.9 will add RDF property `surname` with a corresponding value from a database to each produced instance of the ontology class `Person`.

Element `Condition` (*template condition*)

A `Condition` element refers to a particular mapping condition and it is always embedded in a template instance. From a template layer point of view, a template condition restricts rows of a relation returned by the template instance. It selects (and transformes into RDF instances) only those rows that conform the mapping condition.

A template condition element in a template document must have a value. his value can be either a literal or a `Variable` element, which is a reference to the relating template instance. The sample of the former case is seen in Listing 5.10, and the latter case is detailed in the `Variable` section.

Listing 5.10: Template condition example

```
<PutInstance name="person">
   <Condition name="username">svihlm1</Condition>
   <PutProperty name="surname"/>
</PutInstance>
```

The condition in Listing 5.10 contains a literal and reduces selected persons to one with the username *svihlm1*.

Element `Variable` (*template variable*)

A `Variable` element refers to a particular mapping variable and it must be embedded in a `Condition`. It puts a variable value to a condition. Since a mapping variable exposes a value from an enveloping mapping class, a referring template variable is able to make a connection between instances from two corresponding mapping classes.

Listing 5.11: Complex template document example

```
<PutInstance name="person" id="1">
   <Condition name="username">svihlm1</Condition>
   <PutProperty name="surname"/>
   <PutProperty name="currentProject" ContainerNodeId="projectBagId">
      <PutInstance name="project">
         <Condition name="personId">
            <Variable id="1" name="personIdVariable"/>
         </Condition>
         <PutProperty name="projectHomepage"/>
      </PutInstance>
   </PutProperty>
</PutInstance>
```

The example in Listing 5.11 is the most complex one and it demonstrates all features of the template language. The sample is based on the previous running example listings and also

on the schema mapping document A.1 in the Appendix A. The listing shows a relationship between one person and his or her projects. The used variable has an `id` attribute that refers to the corresponding instance: the instance of the class Person. The result of this template is seen in Listing 5.12.

Listing 5.12: Resulting RDF

```
<Person rdf:about="http://webing.felk.cvut.cz/people/svihlm1">
  <surname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Svihla
  </surname>
  <currentProject>
   <rdf:Bag rdf:nodeID="projectBagId">
    <rdf:li>
     <Project rdf:about="http://webing.felk.cvut.cz/projects/123">
       <homepage>http://webing.felk.cvut.cz/~svihlm1/metamorphoses/
       </homepage>
     </Project>
    </rdf:li>
   </rdf:Bag>
  </currentProject>
</Person>
```

5.4 Schema mapping language syntax (normative)

5.4.1 General elements

Document prolog

Since the schema mapping language is XML-based, documents must begin with the appropriate XML prolog:

```
<?xml version="1.0"?>
```

Root element

The root element of a mapping document is `Mapping`. All mapping elements described in the following text must be embedded in this root, which has the following form.

```
<Mapping xmlns="http://webing.felk.cvut.cz/metamorphoses/mapping/">
  content
</Mapping>
```

Header and footer of an RDF document

There are two mandatory elements in the mapping document language that refer directly

to produced RDF documents: header element and footer element. They have the following form, respectively:

```
<DocumentHeader><![CDATA[content]]></DocumentHeader>

<DocumentFooter><![CDATA[content]]></DocumentFooter>
```

content The content of these mandatory elements is attached to the beginning or the end, respectively, of every RDF document based on a particular mapping document. The content of these elements is `CDATA`, which stands for character data and means that XML tags can be included.

Database connection

The `DatabaseConnection` element is in place to establish a connection with a particular RDBMS. The syntax of this mandatory element does not depend on a mapping model design but rather on a particular implementation. When using a JDBC driver to connect a database, the syntax of this element can be as follows:

```
<DatabaseConnection jdbcURL = "jdbcURL"
                    jdbcDriver = "jdbcDriver"
                    username = "username"
                    password="password" />
```

The attributes of the element are self-evident.

5.4.2 Concept elements

Class

The `Class` element is a basic ability of the language to express a relation between one or more database tables and corresponding ontology class.

```
<Class templateName="templateName"
       rdfLabel="rdfLabel"
       sql="sqlQuery">
  Content
</Class>
```

templateName (mandatory) Specifies a unique element identifier. A mapping element is refered from a template document by this identifier. Identifiers are case-sensitive.

rdfLabel (mandatory) Specifies a name of a relevant ontology class that is referenced by this fragment. A full class name with namespace of a relevant ontology is required.

sqlQuery (mandatory) The attribute contains an SQL query that returns a subset of a database to be mapped by this fragment. The query can be `SELECT`, `VIEW`, etc., and its form is RDBMS-specific.

content The following mapping elements can be content of this element: `Property`, `Attribute`, `ClassCondition` or `Variable`.

Property

The `Property` element represents an ontology property. It always corresponds with an enveloping mapping class.

In the case of datatype mapping property (property for which the value is a literal) it joins the ontology property with a particular database column from a relation returned by a query from an enveloping mapping class. In the case of object mapping property, it only links the ontology property to the template layer.

```
<Property templateName="templateName"
          rdfLabel="rdfLabel"
          [sqlName="sqlName"]
          [containerType="BAG | ALT | SEQ | COL"] >
  Content
</Property>
```

templateName (mandatory) Specifies a unique identifier of a fragment within a mapping. A mapping fragment is referenced from a template document by this identifier. Identifiers are case-sensitive.

rdfLabel (mandatory) Specifies a name of a relevant ontology property that is referenced by this fragment. A full property name with namespace of a relevant ontology is required.

sqlName (optional) If a mapped ontology property is a datatype property, the value of this attribute is the name of a column from a database result set, returned by SQL query from an enveloping mapping class.
If a mapped ontology property is an object property, this attribute is ommited.

containerType (optional) If a mapped ontology property is a object property that can have more values (i.e. it is not functional property), this attribute specifies the type of RDF container that will be used for holding the property values. Legal attribute values are `BAG`, `ALT`, and `SEQ`, which stand for bag, alternative and sequence container type, or `COL`, which stands for collection from the RDF specification.

content An element `Attribute` can appear in the content of this element.

Attribute

The `Attribute` element is to specify an RDF attribute of an enveloping mapping class or property.

```
<Attribute rdfLabel="rdfLabel"
           [prefix="prefix"]
           [sqlName="sqlName"]
           [sufix="sufix"] />
```

rdfLabel (mandatory) Specifies the name of an RDF attribute.

prefix (optional) Puts the specified literal at the beginning of an RDF attribute value.

sufix (optional) Puts the specified literal at the end of an RDF attribute value.

sqlName (optional) If specified, it joins this element with a column from a database result set, returned by SQL query from an enveloping mapping class. The value returned from a database is put into an RDF attribute value between *prefix* and *sufix* values.

5.4.3 Relationship elements

Condition

The `ClassCondition` element is designed to drive the use of mapping class elements from a template document. It can serve two purposes:

- It can restrict rows in a result set returned by SQL query from an enveloping mapping class (when *tableString* is not specified).

- It can add more database tables to query (by specifying their list in *tableString* attribute). This means that more mapping classes can be linked in a template document.

This is detailed in Section 5.3. The syntax of the mapping condition element as follows.

```
<Condition templateName="templateName"
                whereString="whereString"
                [tableString="tableString"] />
```

templateName (mandatory) Specifies a unique identifier of a condition within a mapping. A condition is referenced from a template document by this identifier. Identifiers are case-sensitive.

whereString (mandatory) A value is added to a `WHERE` part of an SQL query from an enveloping mapping class.

tableString (optional) A list of database tables that should be added to a `FROM` part of an SQL query from an enveloping mapping class.

Variable

The `Variable` element is to expose a single value from an enveloping mapping class in order to form a connection with other mapping classes in a template document. This is detailed in Section 5.3.

The syntax of the mapping variable element as follows.

```
<Variable templateName="templateName"
          sqlName="sqlName" />
```

templateName (mandatory) Specifies a variable name – the unique identifier of a mapping variable within a mapping. A mapping variable is referenced from a template document by this identifier. Identifiers are case-sensitive.

sqlName (mandatory) The value of this attribute is the name of a column from a database result set, returned by SQL query from an enveloping mapping class. A literal returned from the database is a value of a variable.

5.5 Template language syntax (normative)

The template language is designed to compose templates for RDF document production.

5.5.1 General elements

Document prolog

Since the template language is XML-based, documents must begin with the appropriate XML prolog:

```
<?xml version="1.0"?>
```

Root element

The root element of a template document is `Template`. Element `Mapping` must be embedded in this root. Another possible child element in the root is only `PutInstance`.

```
<Template xmlns="http://webing.felk.cvut.cz/metamorphoses/template/">
  content
</Template>
```

Reference to a mapping document

A template document is always based on one particular mapping document. This element is to join a template document with its mapping document. The syntax as follows:

```
<Mapping url="url" />
```

url (mandatory) Specifies the location of the corresponding mapping document.

5.5.2 Production elements

PutInstance

This element puts instance(s) of a specified mapping class into a generated RDF document. An RDF instance is generated with all of its attributes defined in a corresponding mapping class.

```
<PutInstance name="name" [id="id"] [nodeId="nodeId"] >
  content
</PutInstance>
```

name (mandatory) Specifies the corresponding mapping class from a mapping document. This attribute corresponds with attribute *templateName* in a mapping document.

id (optional) If a variable of the corresponding mapping class is used somewhere in a template document, this specifies the identifier for such a variable.

nodeId (optional) In case the instance is a blank node, it is possible to specify a `rdf:nodeId` attribute for this node.

content (optional) Elements `PutProperty` and `Condition` can appear in the content of this element.

PutProperty

This element puts property of an enveloping instance into a generated RDF document. An RDF property is generated with all of its attributes defined in a corresponding mapping property.

```
<PutProperty name="name" [containerNodeId="containerNodeId"]>
  content
</PutProperty>
```

name (mandatory) Specifies the corresponding mapping property from a mapping document. This attribute corresponds with attribute *templateName* in a mapping document.

containerNodeId (optional) If a referred mapping property is a container, this attribute sets a `rdf:nodeId` attribute for a produced RDF container.

content (optional) The content can be empty in the case of a datatype property. In the case of a object property `PutInstance` property must be embedded.

Condition

This element puts condition of an enveloping instance into a generated RDF document.

```
<Condition name="name"> content </Condition>
```

name (mandatory) Specifies the corresponding mapping condition from a mapping document. This attribute corresponds with attribute *templateName* in a mapping document.

content (mandatory) The content can be a literal or a *Variable* element.

Variable

This element puts a variable value to a condition.

```
<Variable name="name" id="id" />
```

name (mandatory) Specifies the corresponding mapping variable from a mapping document. This attribute corresponds with attribute *templateName* in a mapping document.

id (mandatory) Specifies the particular *PutInstance* element, from which the variable stems. The *Variable* element must be an ancestor in a subtree of a corresponding *PutInstance*.

5.6 Summary

In this chapter we proposed two XML-based languages that materialise abstract concepts of our data transformation model described in Chapter 4. The *schema mapping language* is able to describe a schema mapping between two schemas (an ontology and a RDB schema).

One can build templates for RDF production by employing the *template language*. The languages cooperate in order to enable data transformation from a relational database into RDF documents based on a schema mapping.

6 Completness of the data transformation

In this chapter we sum up features of our data transformation model and consider its completeness according to the used schema formalisms and data formats.

6.1 Relational completeness

The languages proposed in Section 5 are to transform relational data into RDF. In this section we analyse features of the languages to examine their capability to query relational database.

Since we use SQL queries to identify parts of a relational database schema as well as to select data instances from a database, we can address any part of the database (theorem 4.3.1 and its proof). Schema mapping language does not put any restrictions on SQL *SELECT* queries in its elements (Section 5.3.1).

In this section we will show that the combination of our schema mapping and template language is capable of expressing all operations of the relational algebra and thus we will prove the relational completeness of our approach. In doing this, we show that our data transformation model can be an alternative to the native SQL access to relational data.

Relational completeness of our model was verified using the Elsmasri's and Navathe's [72] approach; we proved that the minimal set $\{\sigma, \pi, \times, \cup, -\}$ of relational operators is supported.

Relational schema and RDFS ontology for the following *Selection*, *Projection* and *Cartesian Product* samples are identical to those used in the running example (Section 5.1).

6.1.1 Selection and Projection

The listings 6.1 and 6.2 show schema mapping and template document that illustrate operations *Selection* and *Projection*.

Listing 6.1: Sample mapping for *Selection* and *Projection*

```
<Class templateName="person" rdfLabel="Person" sql="select * from person
   ">
  <ClassCondition templateName="username" whereString="username ="/>
  <Property templateName="surname" rdfLabel="surname" sqlName="
     family_name" />
</Class>
```

Listing 6.2: Sample template for *Selection* and *Projection*

```
<PutInstance name="person">
   <Condition name="username">svihlm1</Condition>
   <PutProperty name="surname"/>
</PutInstance>
```

The selection

$\sigma_{username="svihlm1"}(Person)$

is enabled by the class condition in the mapping and its application in the template. Only RDF resources of type Person that fulfil the condition will be selected as a result.

The projection

$\pi_{family\_name}(Person)$

is modelled by the mapping property and its application in the template. The mapping property allows us the option to access specified attributes from a relation and to write them as RDF properties.

6.1.2 Cartesian Product

The mapping from Listing 6.3 shows two mapping classes that relate to two database relations (`Project` and `Person`).

Listing 6.3: Sample mapping for *Cartesian Product*

```
<Class templateName="person" rdfLabel="Person" sql="select * from person
    ">
  <Property templateName="surname" rdfLabel="surname" sqlName="
     family_name" />
  <Property templateName="currentProject" rdfLabel="currentProject"/>
</Class>

<Class templateName="project" rdfLabel="Project" sql="select * from
         project p">
  <Property templateName="projectHomepage" rdfLabel="homepage"
    sqlName="web"/>
</Class>
```

Listing 6.4: Sample template for *Cartesian Product*

```
<PutInstance name="person" id="1">
   <PutProperty name="surname"/>
   <PutProperty name="currentProject" ContainerNodeId="projectBagId">
```

```
      <PutInstance name="project">
         <PutProperty name="projectHomepage"/>
      </PutInstance>
   </PutProperty>
</PutInstance>
```

This is equivalent to the cartesian product operation

$Person \times Project$.

It should be noted that a projection was applied to both relations and that only one attribute was assigned to the output for each relation. More precisely, we should write that the listings show complex operation

$\pi_{family\_name}(Person) \times \pi_{web}(Project)$

6.1.3 Set-Union and Set-Difference

Operations *Set-Union* and *Set-Difference* require relations with the same arity and compatible attribute domains. As a sample, we will consider two relations `R(r,s)` and `S(r,s)`.

Our template language does not have the means to express these two operations but they can be easily modelled in a schema mapping. A result of such modelling is a mapping class that contains the result of a particular operation and can be used in a template document to put the results into RDF.

Listing 6.5 shows *set-Union* ($R \cup S$).

Listing 6.5: Sample mapping for *Set-Union*

```
<Class templateName="union" rdfLabel="Union" sql="select * from R union
   select * from S">
  ...
</Class>
```

Listing 6.6 shows *set-difference* ($R - S$).

Listing 6.6: Sample mapping for *Set-Difference*

```
<Class templateName="difference" rdfLabel="Difference" sql="select
   distinct * from R where not exists (select * from S where R.a = S.a
   and R.b = S.b);">
  ...
</Class>
```

6.1.4 Composition of Operations

It is also possible to build more complex expressions using multiple operators in our queries. For instance, Listing 6.2 combines selection and projection and Listing 6.4 combines projection and cartesian product. A more complex query is seen in Listing 5.11 in Section 5.3.2, which describes the template language. This sample also demonstrates how to create *(Equi) Join* operation in our languages.

6.1.5 Summary

The first three relational algebra operators ($\{\sigma, \pi, \times, \}$) are supported directly by features of the template language. The other two operators ($\{\cup, -\}$) are supported by SQL constructions in the schema mapping language. It is also possible to combine multiple operators into more complex expressions. In this section we showed that our languages are able to simulate all basic relational algebra operations.

We cannot say that our data transformation is *relational complete* according to Codd's definition of relational completeness [26], because the result of our query is not relation but RDF graph (Codd's definition requires the result of an operator to be a relation). Due to more relaxed definitions[1] our approach is *relational complete*. However, while supporting all relational algebra operators, our approach can be an alternative to the native SQL access to relational database.

6.2 RDFS support

We based our schema mapping approach on the idea that general schema consists of concepts, relationships and restrictions (definition 3.4.1). Both of our used schema formalisms – relational model and RDFS language – conform this idea. Thus we designed the schema mapping model on mapping of concepts, relationships and restrictions.

The mapping layer is capable of mapping a relational database schema into an RDFS ontology. Here we show that we support all RDFS features in our data transformation model. We will address all RDFS concepts as listed in the RDF Schema specification ([17]) and discuss their support.

[1] A definition of relational completeness provided in [72] states that *query language is relational complete if it can express every query that is expressible in relational algebra.*

6.2.1 RDFS classes

rdfs:Resource

All things described by RDF are called resources, and are instances of the class `rdfs:Resource`. This primitive is abstract and we do not address it directly but all classes, properties, and instances used in our data transformation automatically refer to this primitive.

rdfs:Class, rdf:Property

The concept *mapping class* (definition 4.5.1) is to refer to RDFS classes.

The concept *mapping property* (definition 4.5.2) is to refer to RDFS properties.

rdfs:Literal

The class `rdfs:Literal` is the class of literal values (strings). In Section 4.8, we showed that it is possible to create an RDF triple that include literal value. This can be done by applying a *datatype mapping property* (definition 4.5.3) to the mapping layer.

rdfs:Datatype

`rdfs:Datatype` is the class of datatypes. Any instance of this class can be added to a literal string to create a typed literal. In our model this can be accomplished by specifying a datatype RDF URI reference in the mapping attribute (definition 4.5.5) with the name `rdf:datatype` in the datatype mapping property.

rdf:XMLLiteral

RDF predefines just one datatype: `rdf:XMLLiteral`, which is used for embedding XML in RDF. The support for this feature is discussed in the next Section (6.3).

6.2.2 RDFS properties

rdf:type

Property `rdf:type` is used to state that a resource is an instance of a class. Although it is not directly mentioned in our languages, the data transformation model supports this feature, as an ontology class is automatically assigned to each generated RDF instance. This

is possible because an RDF instance is based on a template instance, which is connected to an ontology class via a mapping class.

rdfs:seeAlso, rdfs:isDefinedBy

Property `rdfs:seeAlso` indicates a resource that might provide additional information about the subject resource, and `rdfs:isDefinedBy` points to a resource that defines the subject resource. In our transformation languages, both properties can be added to any generated resource via mapping attributes assigned to mapping classes.

rdf:value

Property `rdf:value` is the predefined RDF primitive in place to describe structured values. It can be used as any other ontology property in our transformation model.

rdfs:range, rdfs:domain, rdfs:subClassOf, rdfs:subPropertyOf

Properties `rdfs:range`, `rdfs:domain`, `rdfs:subClassOf` and `rdfs:subPropertyOf` do not affect produced RDF graphs directly; they are simply to define inference rules to enable the RDF entailment on these graphs. For this reason they are not a part of our model. However, if a particular ontology is connected to a produced RDF document, further reasoning with these properties is possible.

rdfs:label, rdfs:comment

These properties are used to provide, respectively, a human-readable version of a resource name and a human-readable description of a resource. They are usually used in an ontology and thus are not important for our purposes. However, if necessary, they can be added to any generated resource via mapping attributes assigned to mapping classes.

RDF containers and collections

RDFS specification ([17]) also defines vocabulary for RDF containers and collections. These RDF features are completely supported by our transformation model, as discussed later. Thus, all relevant RDFS properties and classes for containers and collections are supported as well.

Reification vocabulary

The vocabulary for RDF reification contains one predefined class (`rdf:Statement`) and three predefined properties (`rdf:predicate`, `rdf:subject` and `rdf:object`), which can be used as classes and properties from an ontology and which thus are supported directly.

6.3 RDF support

In this section we show that our transformation model supports all basic RDF features, which means that the RDF document can be built as a result of data transformation. RDF concepts and abstract syntax specification [63] enumerates the following RDF key concepts: *(i)* graph data model, *(ii)* datatypes, *(iii)* expression of simple facts and *(iv)* entailment. RDF Primer [70] adds other capabilities: *(v)* RDF containers, *(vi)* RDF collections, *(vii)* RDF reification, *(viii)* structured values and *(ix)* XML literals.

These concepts and their support are discussed in the following text. However, many of these features were detailed in the previous chapters and thus we will refer to the previous descriptions.

Graph data model

The underlying structure of any expression in RDF is a collection of triples [63], each consisting of a subject, a predicate and an object (definition 4.2.7). The subject is a resource being described by the statement, the predicate is a specific property of the subject, and the object is value of the property. In Section 4.8 we show that our data transformation model is capable of creating RDF graph consisting of triples.

Literals

Literals are used to identify values such as numbers and dates by means of lexical representation. A literal may be the object, but not the subject or the predicate, of an RDF statement [63]. There are *plain* and *typed* literals in RDF. In both cases a literal is a string, but in the latter case the datatype URI is attached to this string (datatypes are discussed in the next paragraph). However, the main part of literals is the string value and such a value can be added to an RDF triple addressing *datatype mapping property*, as discussed in sections 4.5, 5.3.1 and 5.3.2.

Datatypes

RDF uses datatypes in the representation of values such as integers, floating point numbers and dates. It is possible to create *typed literals* using datatypes. In RDF/XML serialisation format (which is used in our approach), a datatype is added to a literal string value by a `rdf:datatype` attribute on the property element [5]. This is possible in our model by adding *mapping attribute* `rdf:datatype` to the *mapping property*, as detailed in sections 4.5, 5.3.1 and 5.3.2.

XML Literals

RDF allows XML literals to be given as an object node of a predicate. These are written in RDF/XML as a normal string literal in the content of a property element and are indicated using the `rdf:parseType="Literal"` attribute on the containing property element [5]. Thus, in our approach, it is possible to add a mapping attribute `rdf:parseType` with value `Literal` to a datatype mapping property when expecting XML tags in a literal value.

Structured values

RDF model intrinsically supports only binary relations, but in some cases it is necessary to represent information involving higher arity relationships (relationships between more than two resources). This kind of structured information can be represented in RDF by describing the aggregate element as a separate resource and then by making separate statements about that new resource [70]. Since this approach uses RDF data model based on triples, structured values are directly supported by our data transformation. The intermediate resource is often represented by blank nodes, which are also supported by our model. In addition, the main part of a structural value can be denoted by predefined RDF property `rdf:value`.

Expression of simple facts

The concept of a simple fact expression in RDF, as described in [63], is based on a graph data model and the possibility to build structured values using blank nodes. Since these factors are possible in our data transformation approach, this RDF feature is also supported.

RDF Containers

Containers are designed to enable the description of groups of things in RDF. A container

is a resource that contains elements, which are called members and can be either resources or literals [70]. RDF specification provides several predefined types and properties that can be used to describe containers. There are three types of containers defined: *bag*, *sequence* and *alternative*. In our schema mapping language (see Section 5.4), it is possible to denote a mapping property to be a specific type of container property by the attribute `containerType`. Referring to such a mapping property from a template document will automatically create a container within the property (as illustrated by samples in 5.3.2).

RDF Collections

An RDF collection is a group of things represented as a list structure in the RDF graph [70]. Describing collections is very straightforward in RDF/XML: the enclosed property has the attribute `rdf:parseType="Collection"`. In our work, a mapping property contains a collection when it has the mapping attribute `containerType` with the value `COL`.

RDF Reification

RDF provides a built-in vocabulary intended for describing RDF statements. A description of a statement using this vocabulary is called a reification of the statement [70]. Predefined classes and properties from this vocabulary can be used in our schema mapping language in the same way as any other classes from an ontology. Since reification statements are built using the triple model, we can say that the reification is supported in our data transformation model.

Entailment

The idea of RDF entailment, as described in [51], is beyond the scope of our approach; we simply transform data to RDF document and make no further reasoning with its statements. However, when a proper ontology is connected to generated RDF, reasoning can be performed later by other applications.

XML serialisation syntax

According to [63], RDF has a recommended XML serialisation form called RDF/XML, which can be used to encode the data model in exchange for information among applications. The RDF output of our data transformation is serialised in this format. In addition, our template language is encoded in XML to make it intuitive for a human user to build a template for RDF document (as commented on in Section 5.2). The RDF/XML syntax

specification [5] lists all syntax terms of the language in the chapter titled *Grammar summary*. Our data transformation model supports all of the listed terms except those in the category *oldTerms* (`rdf:aboutEach`, `rdf:aboutEachPrefix` and `rdf:bagID`), which have been withdrawn from the language.

6.4 OWL support in the data transformation model

Our data transformation model was designed to work with of RDFS ontologies. However, the OWL language is a much more powerful tool for ontology design then RDFS and is intended to be the main ontology formalism for the semantic web [71]. According to this, we intend to make our approach OWL-compatible.

OWL is built on top of RDFS and therefore some parts of OWL language are already supported in our model. The following list divides OWL features into four groups from this perspective. The groups are detailed later in the text and their summary can be found in the Appendix B.

- **Directly supported OWL features** – language constructs that can be used in the schema mapping.
- **Indirectly supported OWL features** – language constructs that do not affect the schema mapping and data transformation but can be used later for reasoning on produced RDF documents.
- **Unsupported OWL features** – language constructs that are not yet supported by our approach.
- **Irrelevant OWL features** – language constructs that are irrelevant to our work.

OWL provides two specific subsets of language constructs [31]: *OWL Lite* and *OWL DL*. The third OWL sublanguage is *OWL Full*, which has the same set of constructs as OWL DL and relaxes some of its constraints. When discussing OWL in the following text, we will address OWL Full.

6.4.1 Directly supported OWL features

According to the full RDFS support in our model, we can say that all OWL features related to RDFS are supported automatically. The construct `owl:Class` plays the same role as `rdfs:Class` and thus can be mapped in the same way. Additionally, `owl:ObjectProperty` and `owl:DatatypeProperty` are subclasses of `rdf:Property` [71] and can be used transparently in the schema mapping.

6.4.2 Indirectly supported OWL features

An important task of OWL constructs is to define inference rules to enable the entailment on RDF graphs. These do not affect produced RDF graphs directly and thus we do not use them in the schema mapping. Our approach supports these features indirectly; if a particular ontology is connected to a produced RDF document, further reasoning with asserted statements is possible. These constructs include OWL class axioms (`rdfs:subClassOf`, `owl:equivalentClass`, `owl:disjointWith`), boolean combinations of class expressions (`owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`), RDF Schema property constructs (`rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`), property relationships `owl:equivalentProperty`, `owl:inverseOf`), and logical characteristics of properties (`owl:TransitiveProperty`, `owl:SymmetricProperty`. Property value constraints (`owl:allValuesFrom`, `owl:someValuesFrom` and `owl:hasValue`) are also supported in this way.

6.4.3 Unsupported OWL features

Cardinality restrictions on properties can be set in OWL. However, there is no mechanism in our work to control cardinality of produced RDF instances and therefore the following OWL constructs are not yet supported: `owl:maxCardinality`, `owl:minCardinality`, `owl:cardinality`, `owl:FunctionalProperty`, and `owl:InverseFunctionalProperty`.

6.4.4 Irrelevant OWL features

There is also a set of OWL constructs that provides features for the ontology engineering but do not affect RDF instances of an ontology. These constructs are designed to denote annotations, ontology headers, imports, and version information (see [31] for a list of

constructs) and are not relevant to our work.

6.5 Summary

In this chapter we discussed completeness of our data transformation model according to used formalisms. This discussion proves that we achieved the goals defined at the beginning of our work.

In the definition 3.4.5 we defined the formal goal of our work as follows: Data transformation is an operation $t_{s,t}$, which is able to create a target database D_t conforming to a schema S_t from data stored in a source database D_s conforming to a schema S_s.

Then we specified schema formalisms and data formats (Section 4.1) so that: *(i)* a schema formalism for a source schema is relational model, *(ii)* source data are stored in an RDBMS, *(iii)* a schema formalism for a target schema (i.e. for an ontology) is RDFS, and *(iv)* target data are stored in an RDF document.

In this chapter we showed that:

- the data transformation model can be an alternative to the native SQL access to relational data,
- RDF graph can be composed by our approach and all RDF features are supported,
- all RDFS constructs are supported, and
- a large portion of OWL is supported as well.

This means **our model can map relational schema to RDFS ontology and create RDF document from relational data according to this mapping** (as previously discussed in Section 4.9), which was the goal of our work.

7 Implementation

The chapter describes three tools that were implemented to enable the data transformation described above. The core piece of software is *METAmorphoses*, a processor for our data transformation. *RDF-Shout* is a simple web application that publishes RDF from the processor on the web. *Schema mapping editor* is a GUI editor that helps human users to create mapping between a database schema and ontology. All applications are written in Java for intended platform independence.

7.1 *METAmorphoses* – data transformation processor

This section covers implementation issues of the data transformation processor, including its architecture and transformation algorithm. The usability of our approach is also discussed here as a pratical goal defined in Section 3.3.

METAmorphoses is a processor for data transformation; it transforms data from a relational database into RDF documents according to a mapping. To enable this, *METAmorphoses* processes schema mapping and template documents written in our XML languages (Chapter 5). The processor is implemented using Java and can be used as a standalone CLI application or as a programming library.

7.1.1 Processor architecture

Since our data transformation model is divided into two layers, both the processor architecture and its logic follow this division; there is a *mapping processor* and *template processor*. The former processes mapping documents and the latter processes template documents and drives RDF production.

The mapping layer handles all the complexity and flexibility of the schema mapping while the template processor is a simple programmer interface of the system.

The big picture of the processor architecture in UML is depicted in Figure 7.1. The template processor handles template documents and takes the necessary mapping elements from the mapping processor. Mapping elements also provide SQL queries, which are used in the template layer to fetch data from the relational database. The database is connected via DAO (database access object), which uses JDBC drivers [38]. This means that any

database with a JDBC driver can be connected by *METAmorphoses*. The template processor combine the fetched data with other information in order to create the resulting RDF document. Details of the data transformation process are discussed in the next section.

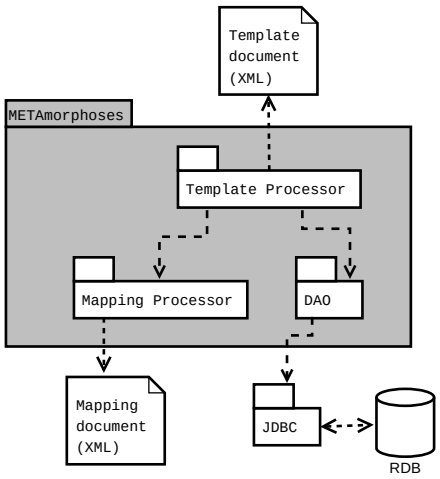


Figure 7.1: *METAmorphoses* architecture

However, this division has a bigger impact than a simple separation of schema mapping and instance transformation. The two-layer architecture is a crucial point in order to build an easy-to-use data transformation system. The server side of common web applications is very often divided into three layers [103] – a presentation layer, a business logic layer, and a database layer. This idea is partially driven by the common design pattern MVC (Model-View-Controller) [41], which strictly separates presentation logic from business logic. This model helps to modularise applications and bring flexibility to their design, but it also separates developer roles in the application development. To reflect this separation in the application development team, there is a presentation layer programmer, a business logic programmer and a person responsible for a database.

An architecture of our framework has many points in common with the MVC model, so the mapping framework can be easily adapted by the web application development process, which is one of our goals (this issue is discussed in Section 7.1.4).

7.1.2 Data transformation process

METAmorphoses are built on the abstract algorithm (4.7.1) proposed in Section 4.4. This algorithm is slightly extended in the implementation:

Algorithm 7.1.1 (METAmorphoses data transformation process)

Input: mapping document M, template document T referencing M, user variables V, relational database D
Output: RDF stream

Step 1: **add** V into T
Step 2: **parse** T
Step 3: **parse** M corresponding to T
Step 4: **write** RDF header to RDF output stream
Step 5: **for each** template element E_t from T do
identify corresponding mapping element E_m from M
create RDF fragment:
– assign RDF attribute names from RDFS ontology stored in M
– assign RDF attribute values from relational database referenced by M
write RDF fragment to RDF output stream
end for
Step 6: **write** RDF footer to RDF output stream

All concepts in the algorithm have already beeb described except for user variables. A user variable is a string, which can be placed to the condition in a template document and data are selected from a database according to this condition. Thus, in this way a user can control RDF production.

The sequence diagram in Figure 7.2 illustrates this process according to system components and their cooperation.

7.1.3 Performance considerations

The question of performance was considered at the beginning of our work and it was one of the goals defined in Section 3.3.

The *METAmorphoses* processor is a stream processor, meaning that it parses elements in a template document one after another and writes corresponding RDF constructs directly to a data stream. This is allowed by the theoretical design of our data transformation

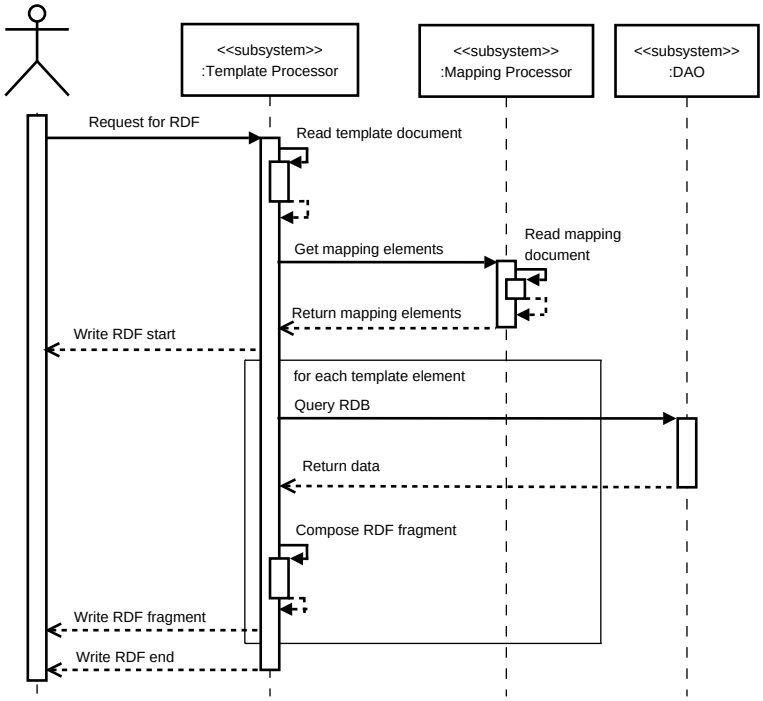


Figure 7.2: Data transformation process

model and by the processor architecture, in which we decided not to use any RDF API in the transformation process. However, this point is a very strong performance issue. The processor does not create an entire resulting RDF model before it is written to an output, thus a RAM footprint of the processor is not affected by the size of the resulting graph (which is not true in the case of tools that use other data transformation approaches). Not using any RDF API also means higher computational performance (as showed in Section 8).

7.1.4 Practical considerations

One goal of our work was to implement a practically useful tool for data transformation. This section provides the answer to the goal *usability* as defined in Section 3.3.

7.1.4.1 Schema mapping layer

As discussed before, our data transformation model uses a relational database schema (i.e. logical schema) as an input rather than an ER schema (i.e. conceptual schema).

This approach is in place for a very practical reason. Even a conceptual schema expresses more semantic information about a mapped database than about a logical one, and we decided on a logical schema primarily because there is usually no conceptual model available for a running database. We wanted to keep our mapping approach as simple as possible, and creating a logical model from a conceptual one would have added more steps to the mapping process. This issue is further detailed in [91] and [80].

7.1.4.2 Template layer

The user interface (and query language) of our data transformation processor is the template document language (Section 5.3.2). A web application programmer designs RDF documents by creating templates. Thus, the usability of our proposal strongly depends on the syntax of this language.

The syntax described in Section 5.5 is not an only possibility – it is only a basic expression of the proposed theoretical concepts. Although the language is standing based on the same principles, its syntax can vary. For example, a *JSP custom tag library* [86] for the language can be deployed so that template tags can be combined with JSP tags. In this way a template can be included directly into a JSP page to publish an RDF document dynamically or to include RDF directly to an (X)HTML page. This is an issue to be explored in future work.

7.1.4.3 Creating data transformation documents

This section provides some usability considerations on how mapping and template documents can be created in *METAmorphoses*.

1. **Building a mapping document**
   In the first step, the concept mapping elements – mapping classes, properties, and attributes – are written. Then, mapping conditions are added to relating mapping classes. Each element has an identifier, which will be used in the template document. It is obviously unnecessary to map all parts of a relational database schema to all

ontology concepts. It is possible to create rules only for necessary concepts and relationships.
The mapping document is created only once and there is only one mapping document for the combination of a relational database schema and an ontology.
The mapping document can be built manually, but in the case of a complex database schema or ontology it can be very difficult to create a valid mapping document. For this purpose we designed a mapping editor (detailed in 7.3); a GUI application for the schema mapping design.

2. **Creating *documentation* for the mapping document**
   Once the mapping document is ready, a sort of documentation can be created for it in *METAmorphoses*. This documentation is a list of possible mapping elements (classes, properties, conditions and variables) that can be used in a template document. The documentation is in human-readable format so that it can be used by a template designer.

3. **Creating template documents**
   Several different template documents can be built on top of one mapping document from the composition of elements from the mapping document (as discussed in Section 5.3.2). Each template document is a template for a set of similar RDF documents. These documents have the same structure as specified by the template but feature different content depending on the corresponding relational data.

When a relational database schema or an ontology is updated, the mapping and/or template document must also be updated. The two-layer architecture adds reliability to this issue. Some changes (typically changes of schema concepts) can be solved by changing only a mapping document without affecting template documents. Only more complex changes of a database model or ontology (typically changes of relationships) must be solved by updating both the mapping document and depending template documents.

7.1.4.4 Developer roles

When we return to our discussion about web application developer roles (in Section 7.1.1), we can see that there are two distinct roles in the mapping creation. The first role, mapping designer, corresponds to a database administrator and/or business logic programmer. This

person must understand relational database schema and the principles under semantic web formats RDF and OWL.

The second role, template designer, is responsible for creating the final RDF document layout. This role is likely filled by the same person as the presentation logic programmer in the web application development. This individual composes elements from the mapping document according to its documentation. Therefore, this person does not necessarily have to know all of the details of the ontology languages or the RDF specification.

This is possible due to the framework architecture, which is based on the MVC model. From the mapping process perspective, we can see the mapping layer as a business logic of our architecture, the template layer as a presentation logic, and the mapping documentation as an API documentation. With the clear separation of developer roles, a development process of semantic web presentation can benefit from the MVC design pattern in the same way that benefits a web application development.

7.1.4.5 Discussion on usability

The practical topics discussed in this section reflect our intention to design and implement a usable tool. Thus, here we include some explanations and recommendations. These considerations were practically tested in the case studies we describe in Chapter 9. Although we performed no formal usability tests, our case study deployments showed that our system is very easy to learn and to use. Formal usability testing is a potential field for future work.

7.1.5 Current status and future work

The *METAmorphoses* described in this section (version 0.2.5) supports all features of data transformation model discussed in this thesis. It is quite a stable and usable software and is available at *http://metamorphoses.sourceforge.net/*.

However, the software presents several issues to improve: the processor lacks a database pool and a good logging system, and some general performance issues could be improved. In the future we plan to provide JSP custom tag lib as an interface for the template processor and standalone PHP implementation of *METAmorphoses* to make this idea available to a larger part of the web community.

7.2 *RDF-Shout* – publishing RDF metadata on the web

RDF-Shout is a simple web application designed to publish data from a relational database as RDF documents on the web. It uses METAmorphoses processor library as a core of the data transformation. The application is written using Java Servlet technology and requires an application server with a servlet container (e.g. Apache Tomcat [44]).

The architecture of *RDF-Shout* is detailed in Figure 7.3. Servlet currently just forwards received HTTP requests to the processor and returns RDF documents in HTTP responses.

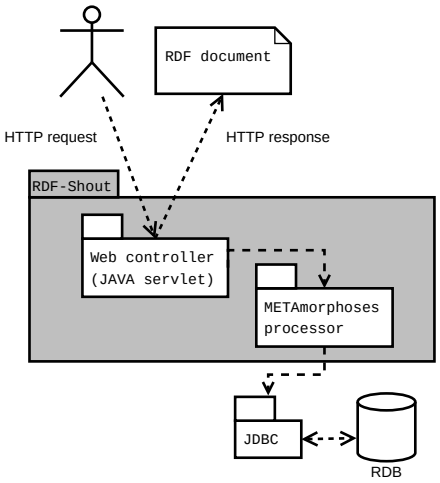


Figure 7.3: *RDF-Shout* architecture

HTTP requests are parsed in *RDF-Shout* in the following way: *(i)* Any *filename.rdf* part of a requested URL is mapped to the template document *filename.xml*. *(ii)* All HTTP parameters are translated to user variables for the template document. For instance, a HTTP request *http://sample.org/person.rdf?username=svihlm1* will result in an RDF document based on the template *person.xml* with the value *svihlm1* for the user variable *username*.

The current version of *RDF-Shout* is very simple and it presents an opportunity for many improvements. We plan to add caching and logging features, web interface for mapping and template document management, and HTTP interface based on the REST architecture [37].

7.3 Schema mapping editor

To simplify the schema mapping process, we provided the GUI editor for creating mapping documents. The application was implemented by Jiří Hofman, a student of Faculty of Nuclear Sciences and Physical Engineering in Prague, who did this work as a research project [55] under the supervision of this author.

The mapping editor is written in Java to provide platform-independent software. It uses JDBC drivers for database connectivity and Jena2 RDF library [22] to handle ontologies.

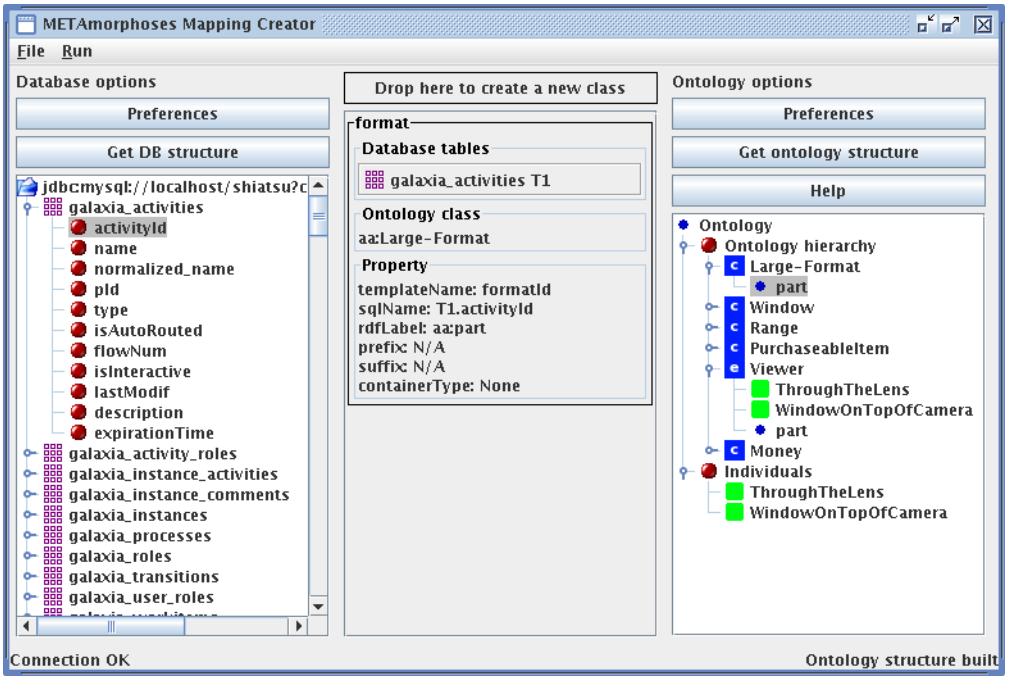


Figure 7.4: Schema mapping editor

The user interface of the application is depicted in Figure 7.4. The editor connects a specified database and provides its structure in the left window. An ontology can be viewed in the right window. The whole application is designed to use the *drag'n'drop* concept. Schema concepts from both sides can be dragged to the centre, where a visual representation of a mapping document is built and the editor asks for values of mandatory fields. This mapping document can be saved in XML format and used directly by the *METAmorphoses* processor.

Features

- All mapping language constructs are supported,
- drag'n'drop based GUI,
- any RDFS or OWL ontology can be loaded,
- any RDBMS with JDBC driver can be connected,
- editor secure consistency of built mapping documents, and
- resulting mapping documents are serialised in our schema mapping language.

Limitations

The main limitation of the application is that it can load only one ontology file at a time. If more ontologies were to be mapped, they would have to be merged into one file. This a is possible area for future work on this tool.

7.4 Summary

The tools described in this chapter were implemented to prove the ideas from the theoretical portion of this work. *METAmorphoses* processor supports all constructs of both XML languages (as described in Chapter 5) and has all of the features of our data transformation model. Another two applications are designed to support the *METAmorphoses*: *RDF-Shout* places the processor library in the context of the (semantic) web and the *schema mapping editor* makes mapping creation a pleasant user experience.

All of this software was also tested and deployed in various case studies. Experimental evaluation of the processor is detailed in Chapter 8, the case studies are described in Chapter 9.

8 Performance analysis

One of the goals of this work, defined in Section 3.3, is *performance*. We designed the theoretical foundations of our data transformation model and processor architecture in order to reach this goal. In this chapter we describe a set of performance tests to compare our transformation processor with similar applications, which are built on other design principles. The tests show how the design affects performance; the test results are discussed at the end of this chapter.

8.1 Experiment overview

8.1.1 Testing environment

The tests were run on an Intel Pentium M processor 1400MHz with 1536 MB of RAM. The operational system was Linux (i386) with kernel version 2.6.12. The Java Virtual Machine was implemented by Sun Microsystems Inc., version 1.5.0_01-b08. The RDBMS for storing data was MySQL server 5.0.30-Debian_1.

All tests were performed within a simple Java benchmarking framework called *JBench* [60]. JBench provides an easy way to compare Java algorithms for speed. In JBench, we used a timer based on the native JVM profiling API to obtain more accurate times. This timer reports the actual CPU time spent executing code in the test case thread rather than the *wall-clock* time, which is affected by CPU load. The granularity of the timer was 10ms.

8.1.2 Compared software

We compared five different systems in our experiments: three tools for the RDB to RDF transformation (METAmorphoses, D2RQ and SquirrelRDF) and two native RDF repositories with RDB back-end (Jena and Sesame1). Moreover, we performed two different tasks with D2RQ and Jena in the most of tests - we queried dataset with both SPARQL and graph API.

8.1.2.1 METAmorphoses v.0.2.5

The *METAmorphoses* processor is the data transformation tool developed in this work and its comprehensive description is found in Chapter 7. We created a schema mapping between the relational schema of the experimental dataset and RDFS ontology and then queried the dataset using our template documents as queries.

8.1.2.2 D2RQ v0.5

D2RQ [85] is a plug-in for the Jena Semantic Web toolkit, which uses the mappings to rewrite Jena API calls to SQL queries and then transmits query results as RDF triples to the higher layers of the Jena framework. Using D2RQ mapping, it is possible to access relational database as a virtual RDF graph via classical Jena API. In this way the relational database can be queried by SPARQL [79] or `find(s p o)` functions and the result is an RDF. When testing D2RQ, we performed two separate experiments: one with `find(s p o)` functions and another with SPARQL. We ran D2RQ in *Jena v2.5.1* in these tests.

8.1.2.3 SquirrelRDF

SquirrelRDF [89] is a tool which allows relational databases to be queried using SPARQL. It provides a tool that creates a rough mapping for a database schema (this is just *the naïve RDB to RDF mapping*, described in [7], which does not consider ontologies) and a set of different SPARQL interfaces. The result of the SPARQL query over RDB is RDF. SquirrelRDF requires *Jena v2.4* and we used its API to perform SPARQL queries in our experiments.

8.1.2.4 Jena v2.5.1 (persistent DB model)

Jena [22] is a Java framework for building Semantic Web applications. It provides a programmer environment for RDF, RDFS, SPARQL and includes a rule-based inference engine. Jena can also store RDF data persistently in relational databases. We stored testing dataset in such a persistent storage (backed by MySQL RDBMS) and then we performed the exact same experiments as we did with D2RQ – we queried stored RDF both by SPARQL and by `find(s p o)` function.

8.1.2.5 Sesame v1.2.6 (persistent DB model)

Sesame [19] is an open source Java framework for storing, querying and reasoning with RDF and RDF Schema. It can be used as a database for RDF and RDF Schema, or as a Java library for applications that need to work with RDF internally. Sesame provides also relational storage for RDF data (so called *RDBMS-Sail*). We uploaded our testing RDF dataset to the Sesame persistent datastore (backed by MySQL RDBMS) and queried it with SeRQL (the internal query language of Sesame) in our experiments.

8.1.3 Testing dataset

The dataset used for the benchmarks is a XML dump of DBLP computer science bibliography [68]. XML was converted into a SQL database dump and into an RDF representation[1].

The SQL version of the dataset consists of six tables (`InProceeding`, `Person`, `Proceeding`, `Publisher`, `RelationPersonInProceeding`, and `Series`) without indices and contains 881,876 records. These relational data were stored in MySQL database and employed while testing *METAmorphoses*, *D2RQ*, and *SquirrelRDF*.

The RDF representation of DBLP contains 1,608,344 statements, and was loaded to relational back-ends of *Jena2* and *Sesame1* to test these systems.

To obtain more granular data, we created the tables `Proceeding500` and `Proceeding1500`, which contain, respectively, 500 and 1500 records from the table `Proceeding`. Then we added these data to the RDF version of the dataset.

8.1.4 Experiment methodology

To compare the tools listed above we used micro-benchmarks. The measured aspect was the time of an RDF production on a given query. Each test task consisted of *(i) preliminary phase*, where the source data, query engine, and query were prepared, and *(ii) measured phase*, where the query was executed and resulting RDF was written to standard output in the RDF/XML syntax (we also decided to measure RDF output because our primary goal is an RDF publishing). According to granularity of the benchmarking tool (10ms) and the high speed of query executions, we executed a query 100 times in a measured phase of each

[1] The data were transformed and provided for our purposes by Richard Cyganiak from Freie Universität in Berlin, Germany.

task.

2-15 tasks (this number was assessed experimentally) were executed before each measured task as a warm-up in order to avoid JVM performance unbalance. After each task, all of its memory references and hardware resources were released.

A test consisted of the same five tasks executed in a row and its result was an arithmetic mean computed from the five task times. Each test was performed for all tested systems.

8.2 Experiments and results

In this section we describe the tests and their results. Testing queries are described in SPARQL formal terminology, although they vary according to the tested system. In the case of SPARQL and SeRQL queries we used *CONSTRUCT* form so that the result was a graph. We also used *CONSTRUCT* form in the cases of METAmorphoses templates and Jena Graph API.

To compare various aspects of RDF production, we divided our tests into three groups. In these experimental sets we tested RDF production according to *(i)* result size, *(ii)* query graph pattern complexity, and *(iii)* query condition complexity.

The results of experiments, described in this section, are evaluated in the next section (8.3).

8.2.1 Experiments with the result size

In this test set we performed a very simple query based on the following general graph pattern:

$$(?s \ < rdf : type > < particular\_RDFS\_class\_URI >)$$

We applied this query to RDFS classes with different amounts of RDF individuals and we analyzed the behaviour of tools according to the size of the resulting RDF graph.

We performed five tests in this group, , proceeding through all resources with type `Series`, `Publisher`, `Proceeding500`, `Proceeding1500` and `Proceeding`. The specific SPARQL queries for these tests with the number of triples in the resulting graphs are displayed in Table 8.1.

Test no.	Query	Result triples
1.1	CONSTRUCT * WHERE {?r rdf:type d:Series.}	24
1.2	CONSTRUCT * WHERE {?r rdf:type d:Publisher.}	64
1.3	CONSTRUCT * WHERE {?r rdf:type d:Proceeding500.}	500
1.4	CONSTRUCT * WHERE {?r rdf:type d:Proceeding1500.}	1500
1.5	CONSTRUCT * WHERE {?r rdf:type d:Proceeding.}	3007

Table 8.1: Tests with the result size: queries

The results of the tests are listed in Table 8.2. Figure 8.1 shows the relation between test time and the amount of resulting data.

	Test no. (number of result triples)				
System	**1.1** (20)	**1.2** (64)	**1.3** (500)	**1.4** (1500)	**1.5** (3007)
METAmorphoses	*84*	*230*	*1840*	*5692*	13124
SquirrelRDF	830	1314	5180	16228	42504
D2RQ SPARQL	522	1332	7730	25530	45704
Jena SPARQL	482	1444	8296	27478	49968
D2RQL Graph API	366	1134	6434	20922	38253
Jena Graph API	368	1028	6902	22874	42588
Sesame1 SeRQL	194	423	2144	6624	*12826*

Table 8.2: Tests with the result size: results (times in ms)

8.2.2 Experiments with the graph pattern complexity

Test queries from this group consist of one graph pattern matching condition, which identifies exactly one RDF resource:

$$(?s \ < my\_ontology : hasTitle > \ "TITLE"\hat{}\hat{}xsd : string)$$

These queries differ in the amount of resources and literals linked by graph patterns in the query. The size of the resulting RDF graph does not differ significantly in these queries and thus we can compare tools according to the complexity of the query graph pattern.

The graph patterns are depicted in Figure 8.2, test results are listed in Table 8.3. Graph in Figure 8.3 shows the relation between the graph pattern complexity and RDF production time.

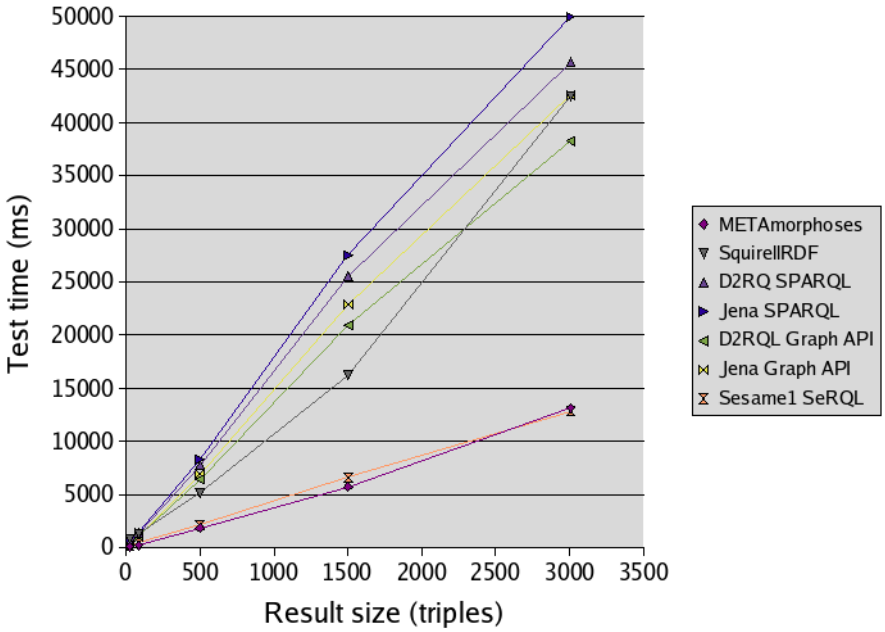


Figure 8.1: Test times rise with the result size

System	**Test no.** (number of result triples)			
	2.1 (2)	**2.2** (4)	**2.3** (6)	**2.4** (8)
METAmorphoses	*28*	*76*	*110*	*124*
SquirrelRDF	640	678	768	808
D2RQ SPARQL	252	426	674	850
Jena SPARQL	212	360	456	552
D2RQL Graph API	106	224	434	506
Jena Graph API	94	150	204	262
Sesame1 SeRQL	100	198	272	324

Table 8.3: Tests with the graph pattern complexity: results (times in ms)

8.2.3 Experiments with the query condition complexity

The tests in the third test set have a very simple graph pattern and they refer to individuals from only one ontology class. These tests differ in number and type of query conditions. We performed these tests only with five systems; we omitted Jena and D2RQ graph API because this API does not allow for straightforward queries with more conditions.

In SPARQL, there are two ways of restricting possible solutions of a query: graph pattern matching and constraining values. This test set contains four tests that combine these

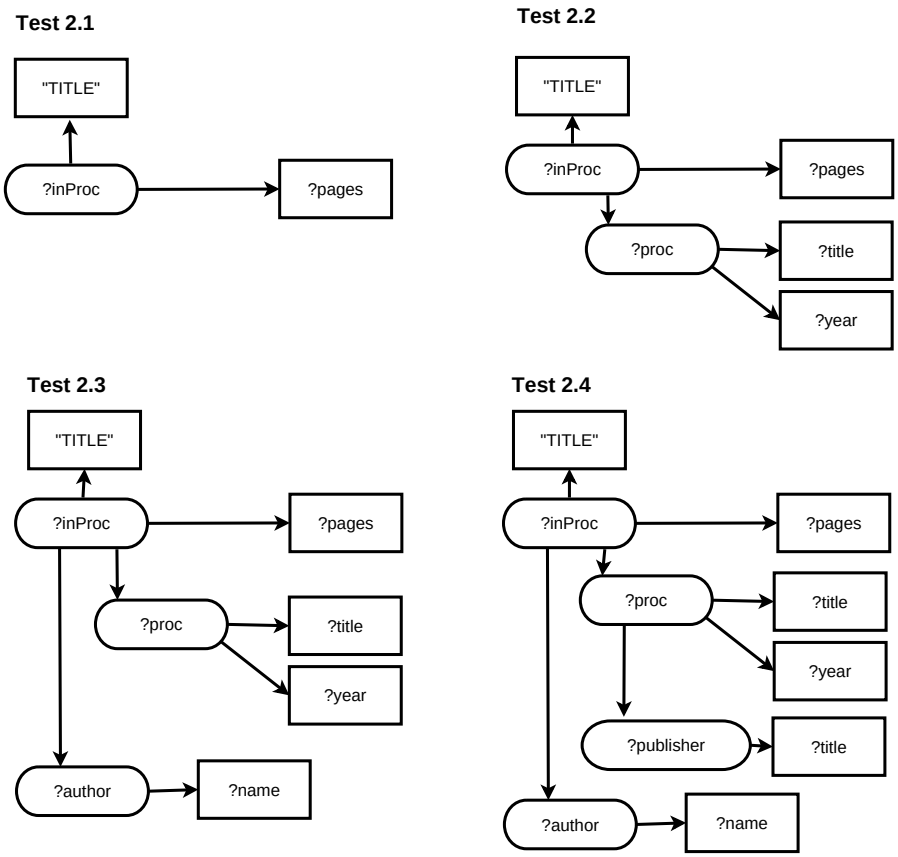


Figure 8.2: Tests with the graph pattern complexity: graph patterns

conditions in various ways. The size of resulting RDF is very small so that the tests are focused on query algorithm performance.

The test queries are depicted in Figure 8.4. The first query (test 3.1) contains a single graph pattern matching condition (similar to queries from the second test set) and the resulting graph contains eight triples. The second query (test 3.2) adds one graph pattern matching condition to the first query and fetches two triples from the dataset. The query in test 3.3 uses a condition with a constraining value to obtain the same result as test 3.2. The last query (test 3.4) combines conditions from test 3.1 and 3.3. The test times are listed in Table 8.4.

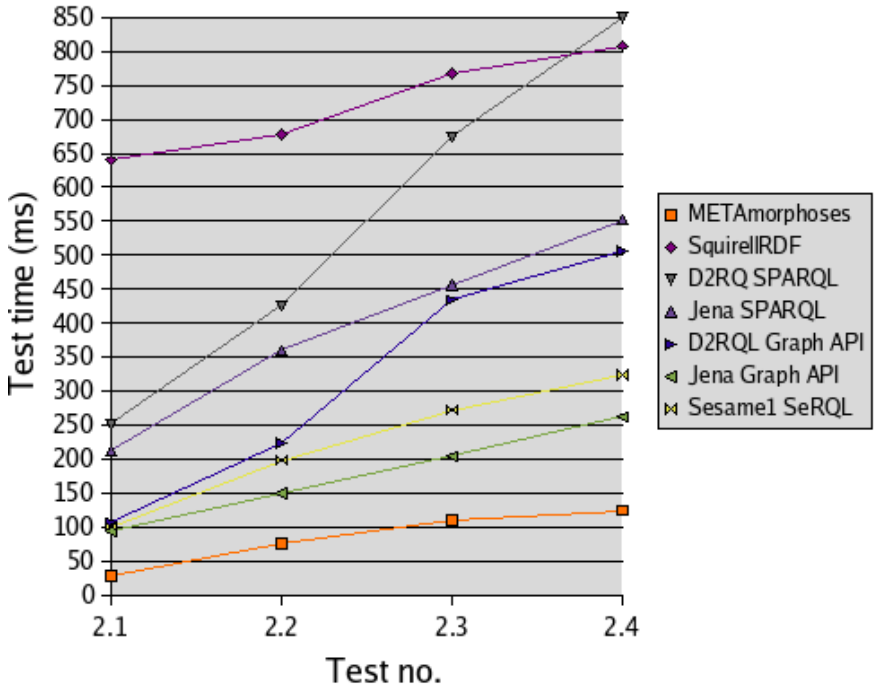


Figure 8.3: Test times rise with the graph pattern complexity

System	**Test no.** (number of result triples)			
	3.1 (8)	**3.2** (2)	**3.3** (2)	**3.4** (2)
METAmorphoses	*78*	*36*	*30*	*32*
SquirrelRDF	636	582	9670	598
D2RQ SPARQL	618	336	17480	360
Jena SPARQL	544	240	30794	232
Sesame1 SeRQL	238	124	110	126

Table 8.4: Tests with the query condition complexity: results (times in ms)

8.3 Discussion

The test results show that our approach (METAmorphoses) was the fastest one in almost all of the tests (except in test 1.5, in which Sesame1 had slightly better performance).

In the first test set, all systems have approximately linear computation performance, as shown in Figure 8.1. The relation between the result size and performance is illustrated in Table 8.5, which contains average times for producing one triple (100 times).

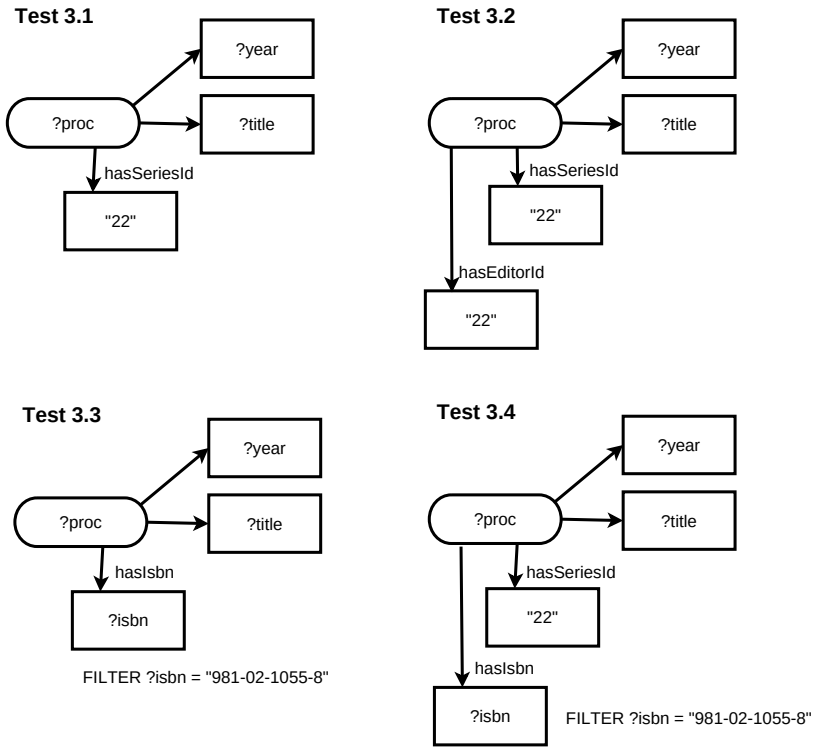


Figure 8.4: Tests with the query condition complexity: graph patterns

	Test no. (number of result triples)				
System	**1.1** (20)	**1.2** (64)	**1.3** (500)	**1.4** (1500)	**1.5** (3007)
METAmorphoses	3,5	2,67	3,68	3,79	4,36
SquirrelRDF	34,58	15,28	10,36	10,82	14,14
D2RQ SPARQL	21,75	15,49	15,46	17,02	15,2
Jena SPARQL	20,08	16,79	16,59	18,32	16,62
D2RQL Graph API	15,25	13,19	12,87	13,95	12,72
Jena Graph API	15,33	11,95	13,8	15,25	14,16
Sesame1 SeRQL	8,08	4,92	4,29	4,42	4,27

Table 8.5: Time (in ms) for producing one triple (100 times), which is based on the first test set

These times are almost identical but do vary between systems. Interestingly, the time-for-one-triple index is slightly higher in test 1.1 than in the other systems. We reason that this is caused by a *starting phase* of the query execution, which does not depend on the result size and is evident in the query with a small resulting RDF (24 triples). The relatively shortest starting phase appears with METAmorphoses and the longest is with

SquirrelRDF (more than twice as high as in the other tests).

Considering this starting phase, there is no reason to compute this index in the second and third test set; the resulting RDF is very small in these tests (2-8 triples).

The METAmorphoses was also the fastest system in the second test set. It maintained its high performance and its small test time growth throughout the increasingly complex graph pattern.

The METAmorphoses also has the best performance in all but the third test set (Table 8.4). Sesame1 is a bit slower but the similarity of its results with those of METAmorphoses is interesting. On the other hand, the other three systems are much slower. This is obvious in test 3.3, where the result times are very high. This is probably caused by non-optimised constraint value handling in the Jena SPARQL query engine, which is used by all of these systems.

It is interesting to observe that all systems based on Jena (SquirrelRDF, D2RQ, and Jena itself) have very similar results, especially in the first and third test sets. We can explain this by the identical algorithms for graph composition (in the first test set) and SPARQL query execution (in the third test set). This means that all solutions built above Jena share its advantages and disadvantages and are limited by its performance. Sesame1 and METAmorphoses had considerably different (and usually much better) test performances. Sesame1 is obviously optimised for querying big amounts of data and METAmorphoses was designed to be a fast data transformation tool.

According to the test results, we can say that our concepts implemented in METAmorphoses show higher performance compared to other tested data transformation systems (D2RQ and SquirrelRDF). Our assumption that RDF API are performance-limiting was correct: our system is faster than those that use RDF API.

METAmorphoses is also faster than tested native RDF persistent storages (persistent DB model in Jena and Sesame1). It does not provide the standard SPARQL interface but it supports all RDF and RDFS features and demonstrates higher performance. This is a very interesting point. We proved that if one needs only to publish relational data in RDF, there is no need to migrate RDB into RDF repository and then query this repository. On-the-fly data transformation (using METAmorphoses) can be achieved faster than queries over RDF repository.

We did not measure RAM footprint of the tested systems. However, METAmorphoses does

not build RDF graph in memory (see Section 7.1.3), thus its memory consumption does not depend on the size of a resulting RDF. All other tested systems first create resulting graph in memory and then serialise it, which means that their RAM footprint does depend on the size of resulting RDF graph.

There are few similar comparison experiments for RDF tools because, as mentioned in [50], the lack of a common query language and access method make benchmarking RDF stores a time-consuming task. However, several attempts are described in [49], [28], or [92]. Due to different methodologies and tested systems, it is very difficult to compare results, but our performance comparison can be considered one of the most complex due to the number of tested systems and performed tests.

8.4 Summary

In this chapter we performed three test sets focused on computational performance to compare our ideas implemented in METAmorphoses with other RDB to RDF transformation tools (D2RQ and SquirrelRDF) and native RDF stores with RDB back-end (Jena and Sesame1). METAmorphoses had the best performance in most tests (12 out of 13) and also showed dominance in the performance aspects discussed in the previous section.

This proves that our concept of data transformation has higher performance than other data transformation solutions as well as native RDF repositories. **This means that on-the-fly data transformation based on our ideas can be done faster than queries over native RDF repository. Thus it is not necessary to migrate relational data into RDF repositories in order to publish them later as RDF.**

Considering this, we can say that we fulfilled the partial goal *performance***, defined in Section 3.3 and proved the performance advantages of our data transformation approach.**

9 Case studies

We conducted various experiments with the implemented system in the process of completing this thesis. These experiments were designed to prove our ideas in a practical environment and to test implementation and usability issues.

Some of the experiments became independent projects based on the METAmophoses. These case studies were supported by grants and their results were also published.

We would like to express gratitude to our supporters and co-workers, who are mentioned in detail later in this chapter in the case studies description.

9.1 SeWebis – department of computer science on the semantic web

The design and implementation of the very first version of METAmorphoses was completed with its experimental deployment. We deployed the system in order to provide RDF metadata about the Department of Computer Science and Engineering at CTU in Prague. The name of the application is *SeWebis*[1].

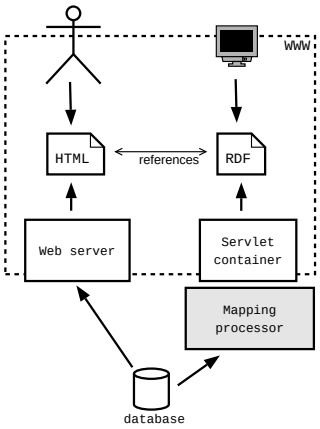


Figure 9.1: *METAmorphoses* extends a dynamic web site

The web portal of the department presents information about people, publications, projects, and education. Data is stored in a relational database and presented as

[1] *Webis* is the name of our department information portal and *Se* stands for *Semantic*

dynamically-generated HTML pages. To extend this web presentation, we first created an OWL ontology that responds to the structure of our department. Then we mapped the existing database schema according to this ontology. We created one mapping document and several templates over this mapping, for example, one template document about persons with their projects and publications, one for publications and their authors, and so on. The metadata generated by the processor were published on the web using *RDF-Shout*.

In doing this, we produced two parallel presentations of our department - one consisting of HTML pages for human users and another consisting of RDF documents for computer applications and software agents (see Figure 9.1). These presentations are created from the same data-source so they carry equivalent information. The presentations are linked together with references.

This project was partially supported by FRVŠ grant agency under grant no. 1804/2005 and by internal grant of Czech Technical University (IGS) under the external number CTU0507513.

9.2 Semantic information retrieval

The RDF data produced in the project SeWebis provided a semantic web platform for further experiments. This environment contains over 200 HTML pages linked to RDF documents with the same information.

In this annotated hyperspace we experimented with semantic information retrieval. We built the search engine, which crawls through HTML pages and digs for linked RDF resources. The crawler downloads RDF, indexes it, and stores it in the RDF knowledge base. RDF statements are indexed by a URL of the web page to which they belong, thus there is a link between RDF data and web pages in the knowledge base.

End-user can query this knowledge base by means of the simple web interface with the semantic search capability. This interface (depicted in Figure 9.2) uses terms from the ontology to question data in the RDF repository. However, a result of the query is not only a set of RDF statements but also a set of links to HTML web pages that are relevant to these statements.

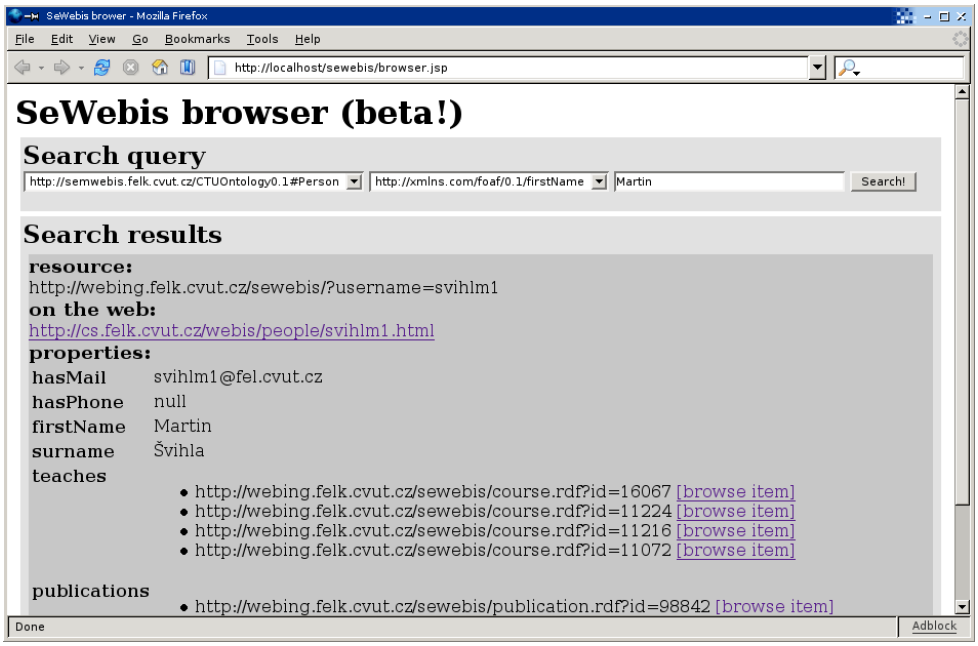


Figure 9.2: Prototype of the semantic search engine user interface

9.3 Publication portal

After establishing the data environment and means for data aggregation and storage, described in the previous sections, we built an information portal capable of aggregating RDF metadata about academic publications.

This portal collected metadata from the SeWebis by using crawlers. Fetched metadata are stored in a knowledge base and published on the portal front-end. The portal serves information about academic publications and their authors. The information can be browsed, sorted, and searched by institutions, authors, publications, dates, and so on. A user can browse or search aggregated information in a single place. Moreover, all of this information is presented in many different formats: HTML, bibTex, plain text, or RDF defined by various publication ontologies. This is possible because RDF can be automatically transformed into any format.

This case is very limited: we fetch information from only one website and thus we cannot speak about a real data integration. However, the experiment proved the usability of the concept and its underlying technologies. We now know it is possible to integrate knowledge from the web using ontologies and semantic web metadata. Moreover, in this case study we

first conducted informal usability tests with the METAmorphoses concept and we received satisfactory results.

This project has been supported by the grant of the Czech Grant Agency no. 201/06/0648. The implementation work was mainly performed by undergraduate students František Pernička and Jan Ledvinka within their bachelor theses ([77], [66]), which were supervised by the author of this work.

10 Conclusion

10.1 Thesis summary

This thesis presents a new model for data transformation from a relational database to an ontology-based RDF metadata. Such transformation allows one to publish relational data without explicit semantics as RDF data with explicit semantics defined by an ontology. The model divides the transformation process into two phases: schema mapping and instance transformation.

The theoretical part of this work includes:

- The identification of correspondences between the relational model and RDFS and also between the structure of relational data and RDF (Section 4.3).
- The design of a formal model architecture (Chapter 4), which is divided into two layers: one for schema mapping and the another for instance transformation based on the schema mapping.
- The proposal of two XML languages based on the formal model (Chapter 5). *Schema mapping language* is capable of describing schema mapping between relational schema and RDFS, while *template language* is for creating RDF documents from a relational database.
- The proposal of a high performance algorithm for the data transformation (Sections 4.7 and 7.1.2).

We verified our theoretical approach by conducting a formal examination of the model capability. We showed that a result of our data transformation is an RDF document with a connection to the used ontology (Section 4.8). We also explored relational completeness of our approach and its compatibility with RDF and RDFS standards. We showed that our model supports all features of used formalisms and data models (Chapter 6). Our model also supports a basic set of OWL features (Section 6.4 and Appendix B).

A data transformation system was designed according to the proposed formal model, was implemented and was verified. Its design, implementation and used algorithm are described in Chapter 7 along with two supporting tools (a server for publishing RDF metadata on the web and a schema mapping editor with GUI). This system was employed in various

case studies to prove its usability and to investigate practical aspects of the semantic web (Chapter 9). The extensive performance tests were executed to compare our tool with other approaches (Chapter 8).

In addition to the main goal of our thesis, which was a data transformation, we strived for partial goals: *performance* and *usability* (Section 3.3). The usability issue is discussed in Section 7.1.4. A high performance of the data transformation is discussed in Section 7.1.3 and proved in the experimental evaluation (Chapter 8).

10.2 Contribution summary

The main theoretical contributions of the thesis are the formal proposal of a novel model for data transformation, two XML languages for transformation description, and high performance algorithm. The model itself was also formally verified.

The practical outcome of the work is the implementation of a data transformation processor and supporting tools that enable web developers to enrich their RDB-backed websites with RDF metadata[1]. This system was deployed in various case studies to prove our concepts and the performance analysis was executed to compare our system with other approaches.

Another very important contribution is a conclusion of the performance analysis. We showed that our approach is faster not only than similar data transformation tools but also than native RDF repositories. This means that an on-the-fly data transformation based on our concepts can be executed faster than queries over current native RDF repositories. Proving this, we can conclude that it is not necessary to migrate relational data to RDF repositories in order to publish them later as RDF.

10.3 Future work

The results of the thesis invite us to continue our research. There are several points that can be improved upon or explored in future work.

The proposed formal model is aimed at specific data formalisms: relational model, RDF, and RDFS. It might be interesting to generalise the model formally so that it can be used for a data transformation between various formalisms (between OODB and RDF, for

[1]The system is called *METAmorphoses* and is available at http://metamorphoses.sourceforge.net/.

example).

Another possibility is to enable reverse data migration, from RDF to RDB. The schema mapping layer provides all necessary features for this migration, but the point is to design a reverse instance transformation layer.

The support for a large portion of OWL is proposed in this thesis but many important features of OWL are not yet supported. Incorporating the support for OWL restrictions is one of the challenges for future research.

The template documents and template language are only one possible interface of our system. It could be interesting to provide query interface for some standard RDF query language on top of the schema mapping layer.

The system implementation also requires some improvement: the processor lacks a database pool and a good logging system, and some performance issues could be improved. In the future, we plan to provide JSP custom tag lib as an interface for the template processor and we are currently working on stand-alone PHP implementation of the model.

Since we put a strong effort into making our system user-friendly, it could be interesting to formally verify its usability. Such *usability testing* is another potential field for future work.

10.4 General conclusion

The semantic web intends to bring more *machine-understandable* meaning to the world wide web content. However, its underlying idea has nothing to do with machines, computers, or software agents. Its very profound purpose is to improve the web so that it is more transparent, accessible, and useful for human beings; in other words, to improve communication and information interchange. We believe in this purpose and thus we dedicate to it this thesis. To achieve the goals of the semantic web it is neccessary to spread semantic web metadata over the web. The aim of this thesis was to create a means for a transformation of traditional data resources (i.e. relational data) to semantic web metadata and to provide tools for such a transformation.

We can observe many changes induced by the evolution of the web, including changes in social structures, business patterns, work processes, science, policy, and art. New tech-

nologies quickly emerge and disappear and leave behind only the changes they introduced. Semantic web technologies may also disappear in time after they accomplish their purpose, but the idea of cooperation and understanding will continue to grow. Technology vastly improves human communication and will do so much more in the future. Our work is intended to be one step in this direction.

11 Bibliography

[1] J. Albert, R. Ahmed, M. A. Ketabchi, W. Kent, and M.-C. Shan. Automatic importation of relational schemas in pegasus. In *RIDE-IMS*, pages 105–113, 1993.

[2] P. Aubrecht. Ontology Transformations Between Formalisms. Dissertation thesis. Czech Technical University in Prague., 2005.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[4] D. Beckett. RDF Test Cases. W3C Recommendation. Available at: http://www.w3.org/TR/rdf-testcases/#ntriples, February 2004.

[5] D. Beckett. RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/, February 2004.

[6] D. Beckett. Turtle - Terse RDF Triple Language. http://www.dajobe.org/2004/01/turtle/, January 2004.

[7] D. Beckett and J. Grant. Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes. SWAD-Europe deliverable, http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report, February 2003.

[8] A. Behm, A. Geppert, and K. R. Dittrich. On the migration of relational schemas and data to object-oriented database systems. In J. Györkös, M. Krisper, and H. C. Mayr, editors, *Proc. 5th International Conference on Re-Technologies for Information Systems*, pages 13–33, Klagenfurt, Austria, 1997. Oesterreichische Computer Gesellschaft.

[9] T. Berners-Lee. Information Management: A Proposal. http://www.w3.org/History/1989/proposal.html, March 1989.

[10] T. Berners-Lee. Relational Databases on the Semantic Web. http://www.w3.org/DesignIssues/RDB-RDF.html, September 1998.

[11] T. Berners-Lee. Semantic Web on XML. In *XML 2000*, Washington DC, USA, December 2000.

[12] T. Berners-Lee. Semantic Web for Industry (Keynote speach). In *5th International Semantic Web Conference (ISWC2005), Industry day*, November 2005.

[13] T. Berners-Lee. Notation 3. http://www.w3.org/DesignIssues/Notation3, March 2006.

[14] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), May 2001.

[15] E. Bertino and L. Martino. *Object-Oriented Database Systems: Concepts and Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[16] E. Bozsak et al. KAON - Towards a Large Scale Semantic web. In *Third International Conference E-Commerce and Web Technologies, EC-Web 2002*, France, 2002.

[17] D. Brickley. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, February 2004.

[18] D. Brickley and L. Miller. FOAF Vocabulary Specification. http://xmlns.com/foaf/0.1/, March 2004.

[19] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proceedings of the First International Semantic Web Conference*, page 5468. Springer, 2002.

[20] W. Bush. As We May Think. *The Atlantic Monthly*, (6), 1945.

[21] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.

[22] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA, 2004. ACM Press.

[23] H. Chen, Z. Wu, G. Zheng, and Y. Mao. Rdf-based schema mediation for database grid. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 456–460, Washington, DC, USA, 2004. IEEE Computer Society.

[24] F. Ciravegna. Adaptive Information Extraction from Text by Rule Induction and Generalisation. In *Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 1251–1256, San Francisco, CA, USA, August 2001.

[25] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[26] E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[27] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, March 2002.

[28] R. Cyganiak. Benchmarking D2RQ v0.2. Technical Report.Freie Universitt Berlin, Germany. http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2rq/benchmarks/, June 2004.

[29] Q. D. and K. D. How to Make a Semantic Web Browser. In *13th international conference on World Wide Web, WWW2004*, pages 255–265, 2004.

[30] DCMI. Dublin Core Metadata Initiative. Information about the initiative available at: http://dublincore.org/about/, March 2001.

[31] M. Dean and G. Schreiber. OWL Web Ontology Language Reference. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, February 2004.

[32] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM Press.

[33] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation. http://www.w3.org/TR/xmlschema-0/, October 2004.

[34] D. Fensel et al. *Spinning the Semantic Web*. The MIT Press, 2003.

[35] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. Oil: An ontology infrastructure for the semantic web, 2001.

[36] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[37] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000. Chair-Richard N. Taylor.

[38] M. Fisher, J. Ellis, and J. C. Bruce. *JDBC API Tutorial and Reference*. Pearson Education, 2003.

[39] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical report, May 1999.

[40] F. Frasincar, G.-J. Houben, R. Vdovjak, and P. Barna. RAL: An Algebra for Querying RDF. *World Wide Web: Internet and Web Information Systems*, (7):83–109, 2004.

[41] E. Gamma et al. *Design Patterns*. Addison-Wesley Professional, 1995.

[42] J. Golbeck, M. Grove, B. Parsia, A. Kalyanpur, and J. Hendler. New Tools for the Semantic Web. In *EKAW 2002*, pages 392–400, 2002.

[43] C. F. Goldfarb and P. Prescod. *The XML handbook*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[44] J. Goodwill. *Apache Jakarta-Tomcat*. Apress, Berkely, CA, USA, 2002.

[45] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 2(5):199–220, 1993.

[46] J.-L. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *WCRE '93: Proceedings of the 1993 Working Conference on Reverse Engineering,* (Baltimore, Maryland; May 21-23, 1993), pages 161–170. IEEE Computer Society Press (Order Number 3780-02), May 1993.

[47] S. Handschuh and S. Staab. Authoring and Annotation of Web Pages in CREAM. In *WWW2002 International Conference*, pages 462–473, Honolulu, Hawaii, USA, May 2002.

[48] S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. In *12th International World Wide Web Conference, WWW 2003*, 2003.

[49] A. Harth and S. Decker. Optimized index structures for querying RDF from the Web. In *Proceedings of LA-WEB 2005*, November 2005.

[50] A. Harth and S. Decker. Yet Another RDF Store: Perfect Index Structures for Storing Semantic Web Data With Contexts. Research Paper, Digital Enterprise Research Institute, Galway, Ireland, 2005.

[51] P. Hayes. RDF Semantics. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/, February 2004.

[52] J. Heflin. Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. Dissertation thesis. University of Maryland., 2001.

[53] J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.

[54] J. Hjelm. *Creating the Semantic Web with RDF*. Wiley Computer Publishing, New York, 2001.

[55] J. Hofman. Mapping relational database schema to rdf. Research project report at Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, CTU in Prague, September 2006.

[56] I. Horrocks. DAML+OIL: a description logic for the semantic web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002.

[57] E. Hyvonen et al. Finnish Museums on the Semantic Web: The user's Perspective on MuseumFinland. In *Museums and the Web Conference 2004*, Arlington, VA, USA, 2002.

[58] ISO (International Organization for Standardization). Information processing – text and office systems – Standard Generalized Markup Language (SGML. ISO 8879:1986(E), 1986.

[59] J. H. Jahnke, W. Schäfer, and A. Zündorf. A design environment for migrating relational to object oriented database systems. In *ICSM '96: Proceedings of the 1996 International Conference on Software Maintenance*, pages 163–170, Washington, DC, USA, 1996. IEEE Computer Society.

[60] Jon Skeet. JBench Manual. http://www.yoda.arachsys.com/java/jbench/docs/, May 2006.

[61] J. Kahan, M.-R. Koivunen, E. Prud'Hommeaux, and R. R. Swick. Annotea: An open rdf infrastructure for shared web annotations. In *The Tenth International World Wide Web Conference (WWW 2001)*, pages 623–632, Hong Kong, 2001.

[62] V. Kashyap. Design and creation of ontologies for environmental information retrieval. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, october 1999.

[63] G. Klyne and C. J. J. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, February 2004.

[64] M. Korotkiy and J. L. Top. From relational data to rdfs models. In *International Conference on Web Engineering 2004*, Munich, 2004.

[65] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222, February 1999.

[66] J. Ledvinka. Mapping of Relational Database Content into Semantic Web Metadata. Bachelor Thesis at FEE, Czech Technical University in Prague, September 2006.

[67] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A Brief History of the Internet. http://www.isoc.org/internet/history/brief.shtml, December 2003.

[68] M. Ley. DBLP Bibliography. http://www.informatik.uni-trier.de/ ley/db/, May 2003.

[69] M. Lytras. An interview with Eric Miller. *SIGSEMIS: Semantic Web and Information Systems*, 1(2), July 2004.

[70] F. Manola and E. Miller. RDF Primer (Revised). http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, February 2004.

[71] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/, February 2004.

[72] S. B. Navathe and R. A. Elsmari. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co. Inc., 2001.

[73] OASIS. RELAX NG Specification. OASIS Committee Specification. http://relaxng.org/spec-20011203.html, December 2001.

[74] ODF Alliance. Open Document Format. Information about the initiative available at: http://www.odfalliance.org/, May 2006.

[75] S. B. Palmer. RDF in HTML: Approaches. http://infomesh.net/2002/rdfinhtml/, 2002.

[76] S. Patrick and C. Jeremy. Trix : Rdf triples in xml. Technical Report HPL-2003-268, 2003.

[77] F. Pernička. Aggregation and Usage of RDF Metadata. Bachelor Thesis at FEE, Czech Technical University in Prague, September 2006.

[78] E. Prud'hommeaux. Optimal RDF access to relational databases (W3C Technical Report). http://www.w3.org/2004/04/30-RDF-RDB-access/, April 2004.

[79] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation. http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, February 2005.

[80] V. Raatikka and E. Hyvonen. Ontology-based semantic metadata validation. In *XML Finland 2002 Conference*, Helsinky, Finland, 2002. HIIT Publications.

[81] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, december 2001.

[82] RSS-DEV working group. RDF Site Summary (RSS) 1.0. Information about the initiative available at: http://web.resource.org/rss/1.0/, June 2000.

[83] A. Sahuguet and F. Azavant. Building intelligent Web applications using lightweight wrappers. *Knowledge Engineering*, 36(3):283–316, 2001.

[84] M. C. Schraefel et al. CS AKTive space: representing computer sci-ence in the semantic web. In *13th international conference on World Wide Web, WWW2004*, pages 384–392, 2004.

[85] A. Seaborne and C. Bizer. D2rq – treating non-rdf databases as virtual rdf graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, 2004.

[86] G. Shachor, A. Chace, and M. Rydin. *JSP tag libraries*. Manning Publications Co., Greenwich, CT, USA, 2001.

[87] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. *VLDB Journal: Very Large Data Bases*, 10(2–3):133–154, 2001.

[88] M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, February 2004.

[89] D. Steer. SquirrelRDF. http://jena.sourceforge.net/SquirrelRDF/.

[90] G. Stoilos, G. Stamou, V. Tzouvaras, J. Pan, and I. Horrocks. Fuzzy owl: Uncertainty and the semantic web. International Workshop of OWL: Experiences and Directions, Galway, 2005, 2005.

[91] N. Stojanovic, L. Stojanovic, and R. Volz. A reverse engineering approach for migrating data-intensive web sites to the semantic web. In *IIP 2002*, Montreal, Canada, 2002.

[92] M. Streatfield and H. Glaser. Report on Summer Internship Work For the AKT Project: Benchmarking RDF Triplestores. Technical Report. Electronics and Computer Science, University of Southampton. http://eprints.aktors.org/437/, November 2005.

[93] R. Studer. The semantic web : Suppliers and customers (keynote). In *Proceedings of the 5rd International Semantic Web Conference (ISWC2006)*, 2006.

[94] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-automatic and Automatic Support for Semantic Markup. In *EKAW 2002*, pages 379–391, 2002.

[95] R. Volz, S. Handschuh, S. Staab, L. Stojanovic, and N. Stojanovic. Unveiling the hidden bride: Deep annotation for mapping and migrating legacy data to the semantic web. *Journal of Web Semantics*, 2004.

[96] W3C HTML Working Group. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). W3C Recommendation. http://www.w3.org/TR/xhtml1/, August 2002.

[97] W3C MathML Working Group. Mathematical Markup Language (MathML) Version 2.0 (Second Edition). W3C Recommendation. http://www.w3.org/TR/2003/REC-MathML2-20031021/, October 2003.

[98] W3C SVG Working Group. Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation. http://www.w3.org/TR/SVG11/, January 2003.

[99] N. Walsh. The DocBook Document Type, v4.5. OASIS Standard. http://www.docbook.org/specs/docbook-4.5-spec.html, October 2006.

[100] Wikipedia authors. Social bookmarking (From Wikipedia, the free encyclopedia). http://en.wikipedia.org/wiki/Social_bookmarking, March 2007.

[101] Wikipedia authors. Wikipedia (From Wikipedia, the free encyclopedia). http://en.wikipedia.org/wiki/Wikipedia, March 2007.

[102] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/2004/REC-xml-20040204/, February 2004.

[103] W. Zhao and D. Kearney. Deriving architectures of web-based applications. In *Web Technologies and Applications: 5th Asia-Pacific Web Conference, APWeb 2003*, Xian, China, 2003. Springer.

12 Refereed publications of the author

[A.1] M. Švihla, I. Jelínek. Benchmarking RDF Production Tools. Accepted for *18th International Conference on Database and Expert Systems Applications - DEXA '07*, Regensburg, 2007. Lecture Notes in Computer Science by Springer Verlag. To appear.

[A.2] M. Švihla, I. Jelínek. Leveraging Semantic Web Technologies to Integrate Information Related to Academic Papers. Proceedings of eChallenges e-2006, Dublin: IIMC International Information Management Corporation Ltd, 2006, ISBN 1-905824-02-5.

[A.3] M. Švihla, I. Jelínek. The Database to RDF Mapping Model for an Easy Semantic Extending of Dynamic Web Sites. *Proceedings of the IADIS International Conference WWW/Internet 2005 - Volume I*, Lisboa: IADIS Press, 2005, vol. I, pp. 27-35. ISBN 972-8924-02-X.

[A.4] M. Švihla, I. Jelínek. Information life-cycle on the semantic web. *DATAKON2005*. Brno: VUT, 2005, pp. 345-355. ISBN 80-210-3813-6.

[A.5] M. Švihla, I. Jelínek. Using Semantic Web Metadata for Advanced Web Information Retrieval. *First International Workshop on Representation and Analysis of Web Space – RAWS-05*, Ostrava - Poruba: VŠB - Technická univerzita Ostrava, 2005, pp. 85-89. ISBN 80-248-0864-1.

[A.6] M. Švihla, I. Jelínek. Improving Web Resources Processing by Distributed Semantic Web Metadata. *E-learning and the Knowledge Society 2005*, Berlin, Germany, 2005, pp. 83-94.

[A.7] M. Švihla, I. Jelínek. Transparent Knowledge Interchange on the Semantic Web. *Proceedings of the International Conference on Computer Systems and Technologies*, Rousse: Bulgarian Chapter of ACM, 2005, pp. 3A 4-1-3A 4-6. ISBN 954-9641-42-2.

13 Unrefereed publications of the author

[A.8] M. Švihla. The Semantic Web. (In Czech) *A chapter in the book New generation of web technologies. Editors M. Bureš, A. Morávek and I. Jelínek.*, WOX, Prague, 2005. 16 pages.

[A.9] M. Švihla, I. Jelínek. Semantic Extension of an University Information Portal. (In Czech) *International Conference on Education, Media, Technology (EMTECH 2005)*, Praha: VUT, 2005, ISBN 80-01-03336-8.

[A.10] M. Švihla, I. Jelínek. Metamorphoses - SQL to RDF Mapping for Semantic Web. *18th International Conference on Systems for Automation of Engineering and Research*, Sofia, 2004, ISBN 954-438-428-6.

[A.11] M. Švihla, I. Jelínek. Two Layer Mapping from Database to RDF. *Sixth International Scientific Conference Electronic Computers and Informatics ECI 2004*, Košice, Slovakia, 2004. ISBN 80-8073-150-0.

[A.12] M. Švihla, I. Jelínek. Semantic Web and Automatic Generation of its Content. (In Czech.) *Tvorba softwaru 2004*, Ostrava, 2004. ISBN 80-85988-96-8.

A Schema mapping and template document listings

This chapter contains a complete mapping document, template document example based on the mapping document and resulting RDF document.

A.1 Schema mapping document

The mapping document from Listing A.1 maps the ontology and database schema from running example described in Section 5.1.

Listing A.1: Mapping document listing

```
<?xml version="1.0" encoding="iso-8859-2"?>
<Mapping xmlns="http://webing.felk.cvut.cz/metamorphoses/mapping">

<DatabaseConnection jdbcURL = "jdbc:mysql://localhost/mm_sample"
   jdbcDriver = "com.mysql.jdbc.Driver" username = "mm_user"
   password="mm_pass">
</DatabaseConnection>

<DocumentHeader>
   <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
   <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns="http://www.sample.org/my/sample/ontology/">
   ]]>
</DocumentHeader>

<DocumentFoot>
   <![CDATA[</rdf:RDF>]]>
</DocumentFoot>

<Class templateName="person" rdfLabel="Person" sql="SELECT * FROM person
   p, department d WHERE d.id = p.id_department">
  <Attribute rdfLabel="rdf:about"
     prefix="http://webing.felk.cvut.cz/people/" sqlName="username"/>
  <ClassCondition templateName="projectId" whereString="p.id =
     pp.person_id and pp.project_id =" tableString="person_project pp" />
  <ClassCondition templateName="username" whereString="p.username =" />
  <Variable templateName="personIdVariable" sqlName="p.id"/>

  <Property templateName="surname" rdfLabel="surname"
     sqlName="family_name">
     <Attribute rdfLabel="rdf:datatype"
       prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
  <Property templateName="department" rdfLabel="hasDepartment"
     sqlName="d.name">
```

```
      <Attribute rdfLabel="rdf:datatype"
        prefix="http://www.w3.org/2001/XMLSchema#string"/>
  </Property>
  <Property templateName="currentProject" rdfLabel="currentProject"
     containerType="Bag" />
</Class>


<Class templateName="project" rdfLabel="Project" sql="SELECT * FROM
   projects pro">
  <Attribute rdfLabel="rdf:about" prefix="http://webing.felk.cvut.cz/
     projects/" sqlName="pro.id"/>
  <ClassCondition templateName="personId" whereString="pro.id =
    pp.project_id AND pp.person_id ="
    tableString="person_project pp">
  </ClassCondition>

  <Property templateName="participants" rdfLabel="participants"
     containerType="Bag" />
  <Property templateName="homepage" rdfLabel="homepage"
     sqlName="pro.web">
     <Attribute rdfLabel="rdf:datatype"
       prefix="http://www.w3.org/2001/XMLSchema#string" />
  </Property>
</Class>

</Mapping>
```

A.2 Template document

The following template document is based on the mapping document from Listing A.1.

Listing A.2: Template document listing for a particular person

```
<?xml version="1.0"?>
<Template xmlns:mmt="http://webing.felk.cvut.cz/metamorphoses/template">
<Mapping url="/url/to/mapping/" />

<PutInstance name="person" id="1">
   <Condition name="username">svihlm1</Condition>
   <PutProperty name="surname" />
   <PutProperty name="currentProject" nodeId="projectBagId">
      <PutInstance name="project">
         <Condition name="personId">
            <Variable id="1" name="personIdVariable" />
         </Condition>
         <PutProperty name="projectHomepage" />
      </PutInstance>
   </PutProperty>
</PutInstance>

</Template>
```

A.3 Resulting RDF

The RDF listing is the result of the template document from the previous section.

Listing A.3: RDF result for the template document A.2

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns="http://www.sample.org/my/sample/ontology/">
<Person rdf:about="http://webing.felk.cvut.cz/people/svihlm1">
  <surname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Svihla
  </surname>
  <currentProject>
   <rdf:Bag rdf:nodeID="projectBagId">
    <rdf:li>
     <Project rdf:about="http://webing.felk.cvut.cz/projects/123">
       <homepage>http://webing.felk.cvut.cz/~svihlm1/metamorphoses/
       </homepage>
     </Project>
    </rdf:li>
   </rdf:Bag>
  </currentProject>
</Person>
</rdf:RDF>
```

B Summary of OWL support

Directly supported OWL features	`owl:Class`, `owl:ObjectProperty`, `owl:DatatypeProperty`

Indirectly supported OWL features	OWL class axioms:
	`rdfs:subClassOf`, `owl:equivalentClass`, `owl:disjointWith`,
	Boolean combinations of class expressions:
	`owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`
	RDF Schema property constructs:
	`rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`
	Property relationships:
	`owl:equivalentProperty`, `owl:inverseOf`
	Logical characteristics of properties:
	`owl:TransitiveProperty`, `owl:SymmetricProperty`
	Property value constraints:
	`owl:allValuesFrom`, `owl:someValuesFrom`, `owl:hasValue`

Unsupported OWL features	`owl:maxCardinality`, `owl:minCardinality`, `owl:cardinality`,
	`owl:FunctionalProperty`, `owl:InverseFunctionalProperty`
	`owl:Restriction`, `owl:onProperty`

Irrelevant OWL features	Versioning:
	`owl:versionInfo`, `owl:priorVersion`,
	`owl:backwardCompatibleWith`, `owl:incompatibleWith`,
	`owl:DeprecatedClass`, `owl:DeprecatedProperty`
	Annotation properties:
	`rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`,
	`owl:AnnotationProperty`, `owl:OntologyProperty`
	Header Information:
	`owl:Ontology`, `owl:imports`

Table B.1: Summary of OWL support

C Acronyms used in the text

All acronyms are defined when first used in the text, with the exception of frequently used ones.

CLI Command Line Interface

DAO Database Access Object

DOM Document Object Model

DLG Directed Labeled Graph

DTD Document Type Definition

GUI Graphical User Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

JDBC (JDBC is a trademarked name and not an acronym, but obviously stands for) Java Database Connectivity

JVM Java Virtual Machine

MVC Model-View-Controller

ODF Open Document Format

OO Object-Oriented

OODB Object Oriented Data Base

OWL Web Ontology Language

PHP PHP Hypertext Preprocessor

RDB Relational Database

RDBMS Relational Database Management System

RDF Resource Description Framework

RDFS RDF Schema

REST Representational State Transfer

SeRQL Sesame RDF Query Language

SPARQL SPARQL Protocol and RDF Query Language

SVG Scalable Vector Graphics

URI Unified Resource Identifier

URL Unified Resource Locator

XML eXtended Markup Language

XHTML Extensible HyperText Markup Language

WWW World Wide Web

W3C World Wide Web Consortium