Master's Thesis

Programming the Semantic Web

- A Microformats Compatible GRDDL Implementation for ActiveRDF

Christian Planck Larsen

Department of Computer Science,
Aalborg University

24th of August 2007

The Faculty of Engineering, Science and Medicine
Aalborg University

Department of Computer Science

Title:
Programming the Semantic Web - A Microformats Compatible GRDDL Implementation for ActiveRDF

Project Period:
DAT6,
1st of February -
24th of August 2007

Project Group:
d619b

Author:
Christian Planck Larsen,
planck@cs.aau.dk

Supervisor:
Lone Leth Thomsen

Number of copies: 5

Total number of pages: 78

Abstract

The Web as we know it today is changing from a Web of documents towards a Semantic Web, where meaning is associated with data allowing the creation of a new generation of novel and intelligent applications. This master's thesis gives a technical state of the art analysis of the W3C standards that will enable the Semantic Web and also considers the significance of the Microformats initiative.

Based on existing solutions in both Java and Ruby, it is discussed how to integrate the standards with object-oriented programming languages. It is found that current implementations of the GRDDL standard have poor support for the majority of microformatted data on today's Web. A GRDDL adapter for the ActiveRDF framework is thus developed and is shown to radically improve Microformats compatibilty.

Development of Sementic Web applications is discussed with regards to how it differs from traditional Web application development. It is also explained how ActiveRDF can be combined with Ruby on Rails to allow rapid development of Semantic Web applications. Using this combination of frameworks and the GRDDL adapter developed in this project, a prototype implementation of a Personal Semantic Organizer is made.

Preface

This master's thesis documents the results of my DAT6 project work during the spring semester of 2007 at the Department of Computer Science at Aalborg University, within the Database and Programming Technologies research unit. The project was supervised by Lone Leth Thomsen, whom I would like to thank especially for her valuable guidance on how to structure my work process.

Familiarity with the Ruby and Java programming languages and Web technologies including XHTML, XML, XSLT and XPath will be of advantage to the reader, but is not assumed.

Aalborg University, 24th of August 2007.

Christian Planck Larsen

Summary

The Semantic Web is a vision of the future Web, where meaning is associated with data in a standardized way to allow the creation of intelligent applications that do not need to be programmed specifically against each source of data. Given that the standards for how to associate meaning with data are beginning to mature, the next step towards the vision is to start describing data using the standards and then build applications using the data. The development of such applications will require an integration of the standards with existing programming languages and frameworks. What motivates this master's thesis is an interest in how this integration can be done with an aspiration of contributing to the work being done within this area.

The report begins with a technical study of the W3C standards that will realize the Semantic Web vision. These standards are organized in a technology stack, where the RDF standard defining a framework for the description of Semantic Web data by means of simple facts is at the bottom. An example of such a fact could be the statement that some person is the author of a specific document. To represent the meaning of such a fact, ontologies can be used to define terms such as "person" and "author" and their relationships. Ontologies are described using the RDF Schema and OWL standards and a key point of the Semantic Web is the sharing and reuse of these. Other standards described include SPARQL, which is both a query language for RDF data and a protocol describing how to communicate these queries and their results between clients and query processors. On the Semantic Web,

logical rules can be used to express how new facts can be inferred from other given facts. The RIF standard defines a rule interchange format but is still in the early stages of development.

Independently of the W3C standardization effort, a group of Web practitioners has defined the so called Microformats, which are widely used standards for how to mark up different kinds of data such as contact or event information in Web documents to enable reuse. The role of Microformats in the Semantic Web is discussed and they are believed to be a very important factor that can help bootstrap the adoption process of the Semantic Web. Using Microformats the wealth of e.g. contact information that is already on the Web today can easily be put into a standardized format, which is what is needed to give developers an incentive to start creating applications using this data. As in what can only be seen as a reaction to the success of Microformats, the W3C has defined the GRDDL standard, which specifies how a Web document can identify XSLT transformations to be used to transform the (X)HTML containing embedded semantic data (e.g. microformatted data) into RDF.

After having described the essential Semantic Web standards focus is directed to how these can be integrated with programming languages. It is considered how RDF can be integrated with different programming paradigms and the object-oriented paradigm is selected and it is found that accessing RDF data from a language of this paradigm requires a mapping between the RDF model and the object model. The Jena framework implemented in Java and the ActiveRDF framework implemented in Ruby are two existing solutions providing such a mapping. These solutions are compared and it is concluded that ActiveRDF provides a more deep integration of the RDF standard because of the dynamic capabilities of the Ruby programming language. How the two frameworks provide integration with the RDF Schema, OWL, SPARQL and GRDDL standards is also discussed. Both Jena and ActiveRDF provide support for the GRDDL standard, but unfortunately the majority of Web documents with Microformats do not use the GRDDL standard to associate the document with XSLT transformations. A test is

conducted on a large number of Web documents containing Microformats and shows that only 6% of these use the GRDDL standard, which leaves almost all of this significant source of semantic data useless in an RDF context.

With the goal of improving on this situation, a Microformats compatible GRDDL adapter is developed for the ActiveRDF framework. The specification, design, implementation and test of the solution is documented. The specification describes how the standard procedure for GRDDL-aware agents is extended to facilitate the discovery of Microformats in Web documents through parsing and subsequent association of these with XSLT transformations. It is also specified how the solution should handle blank nodes in the resulting RDF data as these are not supported by the ActiveRDF framework. Furthermore, the reasoning capabilities of the Jena framework is used by the solution to infer statements about the class memberships of RDF nodes to make data available in ActiveRDF through domain specific Ruby classes. The design composes a solution using existing Ruby libraries and defines a `GrddlAdapter` class and its methods. Tests of the developed solution show that it is able to extract RDF from 67% of the Web documents containing Microformats of which as mentioned only 6% followed the GRDDL standard and thus only had a possibility of working with existing GRDDL implementations.

In the last part of the report, application development for the Semantic Web is discussed. It is described how Semantic Web applications differ from traditional Web applications especially with regards to the characteristics of their data sources. A shift towards the use of distributed data sources can already be seen by the many mashup applications being made today. It is also discussed how popular Web development frameworks such as Ruby on Rails can be reused and accomodated to the creation of Semantic Web applications. Finally, two ideas for Semantic Web applications are proposed of which one of them, a Personal Semantic Organizer, is implemented as a prototype.

Contents

Chapter 1

Introduction

The World Wide Web of today consists of a vast collection of interlinked documents with information structured for presentation in a web browser. To mention an example of the information available on the Web, many people have their own personal website containing their contact information such as e-mail address, phone numbers etc. This information is for human consumption only because we are able to infer the meaning of the data based on the way it is presented to us in the browser. However, it is currently not possible to write software that in general can understand the meaning of the data and e.g. process the contact details of ten different people and compose an address list. This is because the data has not been formatted to reflect its meaning but solely for presentation purposes.

The Semantic Web is a vision proposed by Tim Berners-Lee, the original designer of the World Wide Web, where semantics i.e. meaning are added to data on the Web thus creating a Web of knowledge that can be used to develop novel and intelligent applications. The Semantic Web Activity[46] is led by the W3C organization who in cooperation with researchers and industrial partners develops open standards that will enable the future Web. The standards define a number of XML markup languages for encoding knowledge about data on the Web. These languages can be used to express facts such as "The person named John Smith lives in a city called New York". The languages also allow definitions of ontologies, i.e. a structuring of concepts and their relations within some domain. So in the mentioned fact about John Smith the concepts of a person and a city might be defined in some external ontology that can be shared between applications, which thereby agree on the meaning of facts about where people live. Put another way, the Semantic Web brings a very high level of data interoperability allowing developers to more easily take advantage of data provided by external systems.

The idea of the Semantic Web was first published in 2001[30] and the core enabling standards (RDF and OWL) were finalized in 2004. Now, the next step is to put these standards into use in ways for both producing semantic data and ontologies but also for developing applications utilizing these descriptions of knowledge. However, the technologies of the Semantic Web have not yet reached the mainstream and the Web is still dominated by non-semantic documents. There are indications though of an incremental transition with the emergence of semantic annotated data embedded in existing documents on the web.[16] Microformats[8] is the prime example of this approach and is driven independently of the W3C by a group of Web practitioners, who want to add semantics to today's Web rather than creating a future Web.[32]

1.1 Motivation and Objectives

The motivation behind this master's thesis is a belief that a key driver towards the adoption of the Semantic Web vision, through intermediate steps such as Microformats, is the integration of these technologies with existing programming languages and frameworks making it as feasible as possible for developers to start programming the Semantic Web.

The purpose of this project is to review the current state of the art of the Semantic Web standardization effort and to examine how the standards can be integrated with programming languages. This includes a study of existing solutions with the intent of discovering areas that need further work. The main objective of the thesis is to identify such an area and contribute by developing a solution that will improve on the current level of integration between Semantic Web standards and programming languages.

The project will more specifically involve the following steps, which also dictate the organization of the report:

1. A technical description of the Semantic Web standards, both developed under W3C but also independent ones such as the Microformats initiative, that are needed to establish agreement on how the Semantic Web will function.

2. A discussion of how these standards can be integrated with programming languages. After considering different programming paradigms, a choice of paradigm will be made and focus will be directed at how to integrate the standards with languages of this paradigm. The discussion will be made around existing solutions with the aim of revealing areas that need further work.

3. A specific standard or area of the Semantic Web technology stack that needs more programming language support will be selected. The development of the solution will be documented in terms of specification, design, implementation and test.

4. The last part of the report will discuss application development for the Semantic Web, propose ideas for possible applications and implement one of these using the solution developed earlier.

Chapter 2

Semantic Web Standards

This chapter gives an analysis of the Semantic Web standards that are essential for realizing the vision of the future Web. Throughout the chapter examples will be based on the vCard standard[20] – a widely used standard for electronic business cards.

2.1 The Layer Cake

The W3C Semantic Web standardization effort is quite an extensive activity, which has now been under way for almost 10 years. The so called Semantic Web layer cake diagram in figure 2.1 is often used to give an overview of the entire technology stack. The layer cake is still evolving and the version shown here is the most recent from March 2007.

The grey boxes in the diagram represent Semantic Web standards, that are either already finalized or work in progress. These standards rely on the XML and URI/IRI standards, which respectively provide a text based extensible data format for sharing structured information and internationalized, i.e. Unicode character encoded, resource identification. The following sections will explain each of the stack elements starting from below with RDF, which has been a W3C Recommendation since February 2004. A

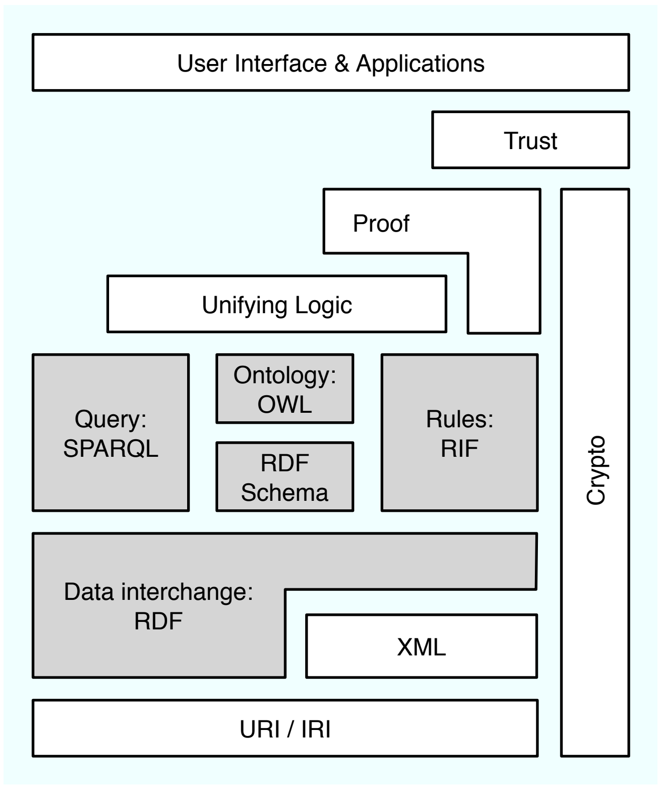


Figure 2.1: *The Semantic Web layer cake. Based on [39].*

standard is called a W3C Recommendation, when it has reached its highest maturity level, have been thoroughly reviewed and is ready for wide deployment. Before a standard becomes a Recommendation it goes through the phases of being a Working Draft, Candidate Recommendation and Proposed Recommendation.[53]

2.2 Resource Description Framework

The Resource Description Framework (RDF)[42] is used to describe knowledge about resources on the Semantic Web. A resource is anything that can be identified by an URI, i.e. ordinary web documents but also people, products, locations etc. that can not necessarily be retrieved and displayed in a web browser. As reflected by is placement in the layer cake of the previous section, RDF is the key W3C Semantic Web standard that other Semantic Web standards build on.

The model underlying RDF is a labeled, directed graph with edges representing facts about the nodes they connect. A fact in RDF is called a triple and consists of a subject, predicate and object and can be interpreted as a statement about the subject having a certain property (the predicate) of some value (the object). The triple elements must all be URIs referencing some resource, except for the object which can also be a plain or typed literal value. Figure 2.2 shows a RDF graph representing part of a person's vCard.

To represent structured information in RDF so called blank nodes are used. Such a node can be seen in figure 2.2 where it is used to represent the aggregate concept of the person's address. The node is blank because it is not identified by an URI like all other nodes.

W3C has defined a standard XML representation language for RDF called RDF/XML so that triples of knowledge can be shared in a standardized format. The triples from figure 2.2 are shown in XML format in listing 2.1 using the RDF representation of the vCard standard[52].

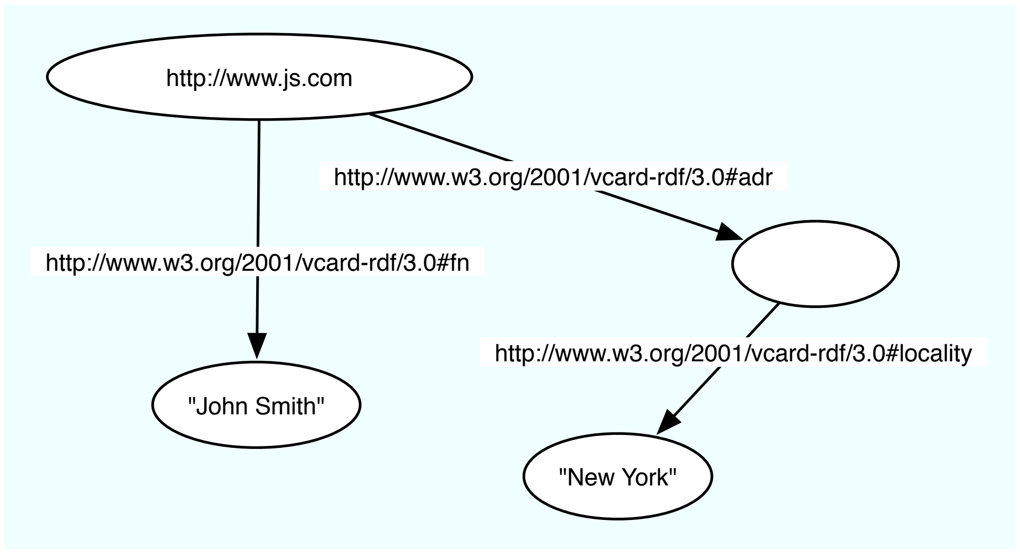

http://www.js.com
http://www.w3.org/2001/vcard-rdf/3.0#adr
http://www.w3.org/2001/vcard-rdf/3.0#fn
http://www.w3.org/2001/vcard-rdf/3.0#locality
"John Smith"
"New York"

Figure 2.2: *RDF graph representing triples stating that a person identified by the URI http://www.js.com is named "John Smith" and is located in "New York".*

Listing 2.1: *John Smith's vCard in RDF/XML*

```
 1  <rdf:RDF
 2    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 3    xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#">
 4
 5    <rdf:Description rdf:about="http://www.js.com/">
 6      <vCard:FN>John Smith</vCard:FN>
 7      <vCard:ADR rdf:parseType="Resource">
 8        <vCard:Locality>New York</vCard:Locality>
 9      </vCard:ADR>
10    </rdf:Description>
11  </rdf:RDF>
```

Line 5 identifies the subject, i.e. John Smith, and line 8 represents the locality predicate and the literal object value. We also notice the use of the terms `FN`, `ADR` and `Locality` which is defined in the external vCard RDF vocabulary imported in line 3. This represents a commitment to a shared way of publishing data, which is a key feature of the Semantic Web. Users of this particalur vocabulary have agreed on the meaning of the terms used to describe personal information and can thus easily share data. In contrast, people not aware of the vCard vocabulary might use the term `City` to mean the same as `Locality`, which would render their data useless to software written only to understand the vCard format. It is of course not realistic that all developers will agree to use the same vocabularies for the same kind of data and as will be described later there are ways of describing that two terms from different vocabularies have the same meaning.

Vocabularies such as these are defined in RDF Schema, a semantic extension to the RDF language, that allows definitions of classes and properties. With RDF Schema one can create a class hierarchy using the `subClassOf` property, which essentially makes RDF Schema a language for defining simple instances of what was earlier referred to as ontologies.

2.3 Microformats

This section moves outside what is explicitly considered by the W3C layer cake and considers an important source of semantic data, which however (as will be clear later) can be transformed into RDF. As was mentioned in chapter 1, annotating existing data in Web documents with semantics is an emerging approach taking one of the first noticeable steps towards a Semantic Web. This idea can be credited to the Microformats initiative, who on their website defines microformats as:

> *"Designed for humans first and machines second, microformats are a set of simple, open data formats built upon existing and widely adopted standards."*[8]

Instead of waiting for the Semantic Web standards to enter the mainstream, the Microformats community decided to start creating common data formats for sharing knowledge on the existing Web using already available technologies. The idea is to mark up e.g. contact information in a Web document using agreed upon values for attributes of the XHTML elements. An example of the hCard format[31] can be seen in listing 2.2, which contains the same contact details as the RDF/XML example on page 8 in listing 2.1.

Listing 2.2: *Example hCard microformat.*

```
<head profile="http://www.w3.org/2006/03/hcard"> ... </head>

<div class="vcard">
  My name is <span class="fn">John Smith</span>
  <div class="adr">
    and I live in
    <span class="locality">New York</span>.
  </div>
</div>
```

The hCard microformat is a 1:1 representation of the vCard standard and the standardized property names such as `url`, `fn`, `adr` and `locality` that are used as values for the `class` attributes in the example above are adopted directly. These property names are specified in vocabularies called microformat profiles, which themselves are expressed in a microformat named XMDP (XHTML Meta Data Profile)[17]. Additional microformats defining how to semantically mark up other types of content in XHTML include hCalender, hAtom, hReview and hResume.

2.3.1 The Role of Microformats in the Semantic Web

The Microformats approach lowers the barrier for putting semantic data on the web as it requires no understanding of new standards and data models. Microformats acknowledge the fact that today and most likely many years ahead in time a lot of information will still be published for human consumption on web sites. Given this fact, the most straightforward solution to making data on these web sites accessible for machine processing is applied. W3C, on the other hand, went for a more universal strategy for knowledge representation on the web by means of the RDF and OWL standards.

However, the problem with these standards is that they are rather complex and hence require more initial understanding to use compared to microformats. The incentive for applying RDF to publish semantic data on the web today is quite limited because the world has yet to see a killer application demonstrating the usefulness of the RDF and OWL standards. On the other hand, why should someone develop such an application if there is not a sufficient amount of data published in RDF? This is an example of the well known chicken and egg problem, which currently slows down the mainstream adoption of the Semantic Web standards. Microformats can help to bootstrap the Semantic Web adoption process because they require a minimum of effort to use. The microformatted data is however not immediately compatible with RDF data and so it would be desirable to bridge the two representations of semantic data to ensure future interoperability.

In July 2006 a new W3C Working Group was formed to address the issue of transforming semantic data expressed in XML languages different from RDF/XML (such as microformatted XHTML) into RDF. The standard is called Gleaning Resource Descriptions from Dialects of Languages (GRDDL)[36] and defines a way to state that a document contains data that can be extracted and transformed into RDF. GRDDL also specifies how to reference the concrete algorithm that will perform the transformation, which will typically be expressed in XSLT[15], a language for specifying transformations of XML documents into other XML documents. The GRDDL standard has by July 2007 received the status of Proposed Recommendation.

2.3.2 Alternative Microformats

The RDFa standard[44] by W3C defines how to embed RDF triples in XHTML documents without repeating existing content. The a in RDFa refers to the fact that RDF data is embedded in XHTML via certain [a]ttributes. To continue the running examples of vCards listing 2.3 shows how John Smith's vCard is expressed in RDFa.

Listing 2.3: *John Smith's vCard in RDFa*

```
1 <html xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#">
2
3 <p about="http://www.js.com/">
4   My name is <span property="vCard:fn">John Smith</span>
5   <div rel="vCard:adr">
6     and I live in
7     <span property="vCard:locality">New York</span>.
8   </div>
9 </p>
```

As can be seen, this looks almost identical to the hCard microformat example in listing 2.2. The most distinct difference is that RDFa markup is a direct representation of RDF triples. As described earlier, triples require a subject, which in this case is identified in line 3. The `about`, `rel` and `property` attributes are introduced in the Metainformation Attributes Module[40] that can be used with XHTML 1.1.

It is also worth noting that in line 1, the same vocabulary as used in the previous listed RDF/XML example (see listing 2.1) is imported, so RDFa has the benefit of being able to reuse existing RDF vocabularies. Furthermore, terms from multiple vocabularies can be mixed and new custom vocabularies can be created. The RDFa approach thus provides a great deal of flexibility compared to the Microformats approach. This difference in degree of flexibility reflects that the idea behind microformats is to define a number of domain-specific formats rather than a generic model that can be used to express anything about anything, which is what RDF provides. Nevertheless, the centralized method used by the Microformats community to develop the formats is not necessarily a bad thing. It ensures that everyone uses precisely the same format to semantically annotate domain-specific data on their web sites, which can not be guaranteed in the RDFa approach. Due to the mentioned flexibility of RDFa, different people might define different vocabularies for describing the same data. Organizations like the W3C might help to prevent this by defining RDF vocabularies for widespread standards as they have done with vCard and iCalender.

Because microformats do not follow a underlying data model such as RDF there is no consistent syntax for expressing microformats and they are all individually defined formats using XHTML markup. As a consequence a different parser must be written for each microformat and also a different transformation for when a microformat needs to be converted into e.g. RDF. On the other hand, all RDFa markup is expressed using a fixed syntax based on the RDF model, and RDF triples can thus be extracted from any embedded RDFa using a single tranformation mechanism.

Another alternative approach to microformats is eRDF (embedded RDF)[24] that as RDFa embeds RDF data in XHTML but without using any new elements or attributes making it complaint with current versions of both XHTML and HTML. As a result of this, only part of the RDF model is supported and e.g. blank nodes and non-plain literals can not be expressed.

2.4 SPARQL

The RDF triples on the Semantic Web can be contained in a RDF/XML document identified by an URI or kept in a so called RDF store, which is a database optimized for storage of large amounts of RDF data. Access to the data in a RDF store is provided through a service endpoint using a query language. A number of different RDF stores with different query languages have emerged, so it was realized that a standard was necessary to ensure interoperability just as SQL did for relational databases.[47]

SPARQL is a recursive acrynom for SPARQL Protocol And RDF Query Language. The work on this W3C standard began in 2004 and is now a Candidate Recommendation. The goal of this standardization effort is twofold, namely to create a standard query language[49] for RDF data and also a protocol[48] describing how to communicate these queries and their results between clients and query processors.

The query language is used to extract information from RDF graphs consisting of a number of triples. The syntax resembles that of SQL used for relational databases and is based on pattern matching. Listing 2.4 shows a simple SPARQL query using a triple pattern for finding all persons who live in New York.

Listing 2.4: *A simple SPARQL query.*

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?x
WHERE { ?x vCard:Locality "New York" }
```

A query is sent to a query processor according to the protocol specification. The SPARQL protocol standard defines an abstract interface containing a single query operation that can be described as a request-response message exchange pattern. The standard defines the contents of these messages along with additional fault messages and shows how the protocol can be supported via either HTTP or SOAP.

2.5 Web Ontology Language and Rules

The Web Ontology language (OWL)[41] takes RDF Schema a step further and allows description of more complex ontologies. It can for example be stated that two properties are equivalent, which could be used in cases such as described in the previous section where “City” and “Locality” were two terms meaning the same thing. OWL can also be used to specify that properties are transitive, symmetric, the inverse of another property, that they have a certain cardinality constraint with respect to some class etc. OWL is like RDF one of the mature Semantic Web standards and also became a W3C Recommendation in February 2004.

Ontologies in RDF Schema and OWL can be used to perform reasoning and infer new facts based on what is already known. A simple example of this, reminiscent of the concept of polymorphism from object-oriented programming languages, is that if a class A is a subclass of another class B, then instances of B can be inferred to be instances of A as well.

A recent proposal called the Semantic Web Rule Language (SWRL)[51] suggests a combination of OWL with the RuleML language to facilitate the creation of user defined inference rules. With SWRL it could for example be expressed in an OWL ontology that the combination of a hasParent and a hasBrother property would imply a hasUncle property. This could be expressed in a rule such as the following:

$$hasParent(x1, x2) \wedge hasBrother(x2, x3) \rightarrow hasUncle(x1, x3)$$

Building on the SWRL proposal, a Rules Interchange Format (RIF) Working Group[45] was started under the W3C by the end of 2005. Once defined, RIF will be a standardized format for rules allowing them to be shared between different rule systems. A core version of the rule format was supposed to be a Recommendation by May 2007, but is still a Working Draft at the time of this writing. The entire RIF effort with extensions to the core format is expected to be finished by 2009.

2.6 Logic, Proof and Trust

The elements at the top of the Semanic Web layer cake have not yet entered the process of standardization and can thus only be discussed at a conceptual level.

The unifying logic layer refers to the need of a logic that defines rules for inference of new knowledge based on statements made using the standards in lower layers. Technology is also needed for constructing proofs, i.e. finding a sequence of rules that can be used to infer that some statement is true. Representations of these proofs can then be shared providing short cuts to new facts in the system and thereby avoiding the high cost of reconstructing the proofs.

Trust is a very important issue in the future Semantic Web, because of the fact that everyone is able to make statements about everything. Users of the Semantic Web will thus need a way of knowing that some document or statement comes from a source that the user trusts. This can be implemented using digital signatures establishing the authenticity of sources of information, which is what the vertical Crypto block on the right side of the layer cake diagram in figure 2.1 refers to. It would be quite limiting for a user only to rely on the sources of information digitally signed by someone that he personally knows that he can trust. The concept of "Web of trust" can help on this issue by trusting sources that are trusted by some source that the user already trusts making a web of trust with possible various degrees of trust.

2.7 Summary

With the Semantic Web layer cake as a starting point, this chapter has progressively described the technology stack enabling the future Web. The RDF standard for representing knowledge on the Semantic Web was described in terms of its graph model, its XML serialization syntax and how terms in vocabularies can be used to share an agreement on meaning of data. The creation of more advanced vocabularies as ontologies with terms organized using classes and properties was introduced in terms of the RDF Schema standard.

Moving outside the W3C standards stack, the popular Microformats initiative for structuring existing data in Web documents using simple data formats was discussed with regards to its role in the W3C Semantic Web vision. Microformats could have a great importance in the mainstream adoption process of the Semantic Web, since it is an easy way of making a lot of existing data semantic. It was also shown how microformatted data can be brought into RDF using the new W3C GRDDL standard and how RDFa and eRDF provide alternatives to Microformats using syntax following the RDF model.

Moving further up the Semantic Web layer cake, the SPARQL standard providing a query language for RDF was described. The OWL standard for creating more complex instances of ontologies and the not yet matured

SWRL and RIF standards for defining logical rules were explained. The upper layers of logic, proof and trust were briefly discussed at a conceptual level since no standardization process addressing these areas have yet begun.

An important point is that despite the fact that many of the upper layer Semantic Web standards are still under development, the fundamental RDF, RDF Schema, OWL and SPARQL standards are mature and ready to be implemented and used for creating the first applications for the Semantic Web.

Chapter 3

Programming the Semantic Web

This chapter will focus on how the Semantic Web standards introduced in the previous chapter can be integrated with programming languages. An integration is required to allow developers to start creating applications that make use of data represented according to the Semantic Web standards, both those of W3C but also independent and widespread standards such as Microformats. The discussion will be based on existing solutions and will identify areas that need further integration effort.

RDF was in the last chapter described as the foundational Semantic Web standard. The integration of programming languages and data represented using RDF will thus be the starting point of this chapter. However, the characteristics of the integration depends on the programming paradigm employed by the language with which RDF data integration is desired.

3.1 Programming Paradigms and RDF

Every programming paradigm offers a particular data model and a way to manipulate data. To provide an integration with RDF requires a mapping between the RDF data model and the model of the paradigm. In the following we consider the logical, functional, imperative and object-oriented paradigms – the four major programming paradigms.Terrence W. Pratt [33, p. 25]

The paradigm which is conceptually closest to the RDF model is the logical programming paradigm. In the logical programming language Prolog[57] a program consists of facts and rules expressed using predicates, which form a knowledge base that can be queried by stating goals that the Prolog reasoning engine attempts to prove. An example of a fact in Prolog could be `lives_in(JohnSmith, "New York")` which would correspond to the RDF triple `(JohnSmith, lives_in, "New York")`. As can be seen the difference between Prolog facts and RDF triples is quite minimal and thus the mapping required to derive facts from triples is a simple one. This suggests Prolog as a suitable programming language for accessing RDF data and for this purpose a Semantic Web library[55] for the SWI-Prolog environment exists. Swish[23] is a Semantic Web framework for Haskell, which is a purely functional programming language. The functional programming paradigm also seems to be a good fit with the RDF model since it is known to especially shine when it comes to representing and manipulating tree based data structures.[58]

While the declarative nature of both the logical and functional programming paradigms matches RDF data very well, it is likely that RDF data in most cases will be needed in an object-oriented or imperative programming context as these paradigms are by far the most widely used paradigms for application development.[29] In the object-oriented paradigm, real world objects are modelled as classes of objects sharing certain attributes. An instance of a `Person` class could for example have an attribute called `lives_in` with the value “New York”. Again, the correspondence between such an object and the RDF triple `(JohnSmith, lives_in, "New York")` is straightforward. Furthermore, the RDF Schema standard allowing the description of RDF data in terms of classes and properties suggests a very close relationship to the object-oriented model.

To make RDF accessible to the majority of programmers and to speed up adoption, Semantic Web frameworks for object-oriented languages are needed. The rest of this chapter will thus focus on how to integrate RDF with languages employing the object-oriented paradigm. This paradigm is chosen with the recognition that the logical and functional paradigms might also prove to be good matches with the RDF data model. In particular, logical programming languages such as Prolog could be of great value when the upper layers of logic and proof in the Semantic Web layer cake have matured into standards and need to be implemented as e.g. reasoning engines.

3.2 Accessing Data of a Different Model

The problem of creating a mapping to the object-oriented programming model in order to provide access to data based on a different model is not a new problem. It has long been necessary to access data in relational databases from object-oriented languages and ways of handling the difference between the models have been found. For example, in the relational model, relations are associated to other relations using foreign keys, whereas objects use object references to associate with each other. Also, objects implement both data and behaviour acting on that data whereas relations just represent data. These and other differences between the object and relational models are referred to as the object-relational impedance mismatch.[59]

The mismatch requires a mapping from one model to the other, referred to as an object-relational mapping, which can be realized in several ways. The most simple approach, commonly found in languages that have support for relational database access, is to provide an API with functions for executing SQL queries on the database and receiving result sets. It is then up to the programmer to manually extract the data into a suitable array, object or other data structure. More advanced object-relational mapping frameworks use meta information about the database to automatically generate a class hierarchy that allows the programmer to access table rows and their field values through objects and their attributes. Typically the framework also provides a number of class methods to e.g. perform queries, generating the necessary SQL behind the scenes. This makes working with relational data feel more natural to the object-oriented programmer and with simple applications the use of SQL can be completely avoided making the relational database appear as a virtual object-oriented database.

3.3 Bringing RDF into Object-Oriented Languages

Like with object-relational mappings, solutions mapping the RDF data model to the object model vary in their approach. In the rest of this chapter, focus will be on two object-oriented Semantic Web frameworks, which both povide a mapping but in different ways. The first framework is called ActiveRDF[19] and is implemented in the scripting language Ruby. It is a fairly young open source project with the first version released around the beginning of 2006 and is still in active development by a group of researchers from the Digital Enterprise Research Institute (DERI) of the National University of Ireland, Galway. The second framework called Jena[6] is implemented in Java and is now an open source project based on work by the HP Labs Semantic Web Programme[3]. The Jena framework has been available since 2000 and is thus quite mature, feature rich and well documented. The two frameworks are chosen because they respectively represent dynamically and statically typed object-oriented languages which is reflected in the way they provide the mapping from RDF triples to programming objects.

The mapping solution employed by Jena is to provide a general API for processing RDF data. Listing 3.1 shows how a RDF representation of a person's vCard is opened, how his name is extracted and finally how a new property can be added to the vCard.

Listing 3.1: *Example Jena code.[25]*

```
Model model    = ModelFactory.createDefaultModel();
InputStream in = FileManager.get().open(rdf_data);

model.read(in, "");

Resource vcard = model.getResource(johnSmithURI);
Resource name  = (Resource) vcard.getProperty(VCARD.FN)
                                 .getObject();

vcard.addProperty(VCARD.NICKNAME, "Smithy");
```

We note that in this approach to working with RDF the API is generic in terms of the `Model` and `Resource` classes and also the `getProperty()`, `addProperty()` and `getObject()` methods and that the actual resource identifiers are supplied as arguments.

The solution offered by the ActiveRDF framework offers a more deep integration with RDF. ActiveRDF maps RDF Schema classes to programming classes, RDF resources to programming objects and RDF predicates to attributes on those objects. Thus, a more domain specific API is provided compared to Jena. An example of ActiveRDF code corresponding to the Jena code in listing 3.1 can be seen below:

Listing 3.2: *ActiveRDF code.*

```
 1  adapter = ConnectionPool.add_data_source :type => :rdflite
 2  adapter.load "rdf_data"
 3
 4  Namespace.register(:NS, 'http://urlofnamespace')
 5
 6  ObjectManager.construct_classes
 7
 8  john = NS::Person.new(johnSmithURI)
 9  puts john.fn
10  john.nickname = "Smithy"
```

After loading the RDF data, classes for each of the RDF Schema classes found in the data is automatically created on line 6 using Ruby's metaprogramming facilities that allow runtime generation of new classes. Suppose that the `johnSmithURI` identifies a resource that is of class `Person` defined in the namespace registered on line 4, then a proxy object for that resource can be created as in line 8. This proxy object has attributes corresponding to the predicates used on the resource, e.g. `fn` and `nickname`. The API to RDF data thus becomes much less cumbersome and more natural to the object-oriented programmer compared to generic APIs such as that offered by Jena and the majority of other Semantic Web frameworks.[18]

RDFReactor[35] for Java builds on the Jena framework adding an RDF API identical to that of ActiveRDF. However, because Java is a statically typed language, RDFReactor requires that the proxy classes are created by a code generator at compile time based on a specific RDF Schema. This conflicts with the open-world semantics of RDF because in RDFReactor resources are required to adhere to a specific RDF Schema definition from which the proxy classes were originally generated. This is an example of one of the mismatches between the object-oriented programming model and the RDF model. In traditional object-oriented languages such as Java, an object only belongs to a certain class if it follows the structure imposed by the class definition. In contrast, an RDF resource which is an instance of a class is not constrained by the class definition and may for example be described further using other properties than those implied by the RDF Schema. The mismatch can however be handled in ActiveRDF because dynamic object-oriented languages such as Ruby are flexible enough to accommodate the individual properties of a resource on run time.

3.4 Ontologies and Reasoning

As described in section 2.5, OWL is an extension of RDF Schema that gives more expressiveness with regards to classes, properties and their relationships in ontology specifications. This added expressiveness can be utilized by Semantic Web frameworks to allow the development of even more intelligent applications. The Jena framework supports OWL through an ontology model, that has functionality for working with data described in OWL. For example, there are methods to find sub- and super-classes of a class in an ontology, setting cardinality constraints, finding equivalent classes from other ontologies etc.

The Jena framework also includes reasoning capabilities, which is what makes an ontology really useful. Based on what is known from an ontology specification new facts can be derived using so called entailment rules. An example of such a rule could be that if a property B is a sub-property of A and a triple (X, A, Y) exists, then the triple (X, B, Y) can be entailed. The function of a reasoning engine is thus to apply all rules to a base of RDF statements deriving a number of new statements representing inferred knowledge. Apart from the rules given by the semantics of the RDF Schema and OWL languages developers can define their own rules in Jena.

Not all Semantic Web frameworks support reasoning; ActiveRDF is such a framework, which has chosen to leave reasoning up to the RDF data stores and focus only on data access. Unfortunately, most of the data stores supported by ActiveRDF does not perform any reasoning. One solution is to use SPARQL to access a Jena based data store on which reasoning has already been applied, but this does not allow one to e.g. perform reasoning on RDF data collected from multiple sources. The developers of ActiveRDF are presumably working on an adapter to Jena that would allow developers to perform reasoning at run time.[21]

A final matter regarding the support of OWL in Semantic Web frameworks is how the restrictions that can be expressed in OWL, e.g. cardinality or value contraints on properties, is handled. This is again a question of how to map the concepts to the object-oriented model. In a paper entitled "Automatic Mapping of OWL Ontologies into Java"[22] suggestions are made on how to create a mapping of OWL by generating corresponding Java classes, similar to the approach of RDFReactor mentioned in the previous section. To enforce property contraints, listeners are registered on field accessor methods in the Java class to check that the constraints are satisfied upon access. ActiveRDF currently has no such support for OWL constraints and the OWL standard in general.

3.5 Querying Semantic Data

Both ActiveRDF and Jena has support for SPARQL – the standard for accessing and querying RDF data on the Semantic Web. In the Jena framework SPARQL query strings can be sent to endpoints via generic query methods to obtain the desired RDF data. ActiveRDF has an abstract Query API that via adapters can be used against different data sources, including SPARQL endpoints. Building on the Query API, ActiveRDF provides so called dynamic finders, which makes it easy to perform simple queries on resources of a given type identified by a proxy class in the Ruby code. Listing 3.3 shows the use of a dynamic finder on the `Person` class to find all person resources, that has a locality property with the value "New York" and print their names.

Listing 3.3: *Dynamic finders in ActiveRDF.*

```
persons_in_ny = Person.find_by_locality("New York")

persons_in_ny.each do |person|
  puts person.fn
end
```

Through the Ruby meta programming facilities, the undefined method `find-_by_locality` is dynamically translated into a query using the abstract query language and then into a query that can be executed on the data source in use, e.g. a SPARQL endpoint.

3.6 Accessing Semantic Data in Web Documents

As described in section 2.3.1, semantic data encoded in web documents using standards such as Microformats must be considered an important part of and a catalyst for the Semantic Web vision. Even though the Microformats initiative started in early 2004, standardization efforts for aligning this kind of semantic data with RDF data through transformation mechanisms have only recently been made.

The GRDDL standard working group was as mentioned started in July 2006 and is expected to have a Recommendation ready in July 2007, which makes framework support for this standard naturally limited in level of maturity. One implementation of the GRDDL specification is the GRDDL Reader[7] for the Jena framework, which was first released in January 2007. In ActiveRDF, it is possible to get RDF data through GRDDL by using the adapter to the Redland RDF data store, which apart from being a data store also has functionality for parsing RDF from multiple sources including web documents containing e.g. Microformats.

For GRDDL to be useful there must exist transformations to transform various dialects of XML into RDF/XML. XLST transformations are already available for RDFa, eRDF and some the most widely deployed microformats such as hCard and hCalender.[56] However, for GRDDL to work it is required that the web documents containing semantic data are linked to these transformations e.g. via profiles such as that shown in line 1 of listing 2.2 on page 10. The use of profiles is however not encouraged by the Microformats community and is only mentioned in a remote part of their wiki.[28] It is clear that Microformats are not considered part of the RDF based Semantic Web vision by its current users, who as mentioned in section 2.3 consider RDF too complex and general to be used for what can be achieved with exisiting and widely adopted standards. Tools that consume the microformatted data on today's Web thus parse the web document and discover any used microformats. As a consequence, the GRDDL implementations offered by both Jena and ActiveRDF are useless with microformatted web documents that lack information associating them with transformations. To document the extent of this problem a Ruby script, see listing A.2 in appendix A.2, has been made to check the lists of hCard and hCalender "examples in the wild".[27][26] The result of running the script shows that only 13 out of 228 Web documents with microformatted data contained GRDDL links to trans-

Subject	**Predicate**	**Object**
_:r1	w3.org/1999/02/22-rdf-syntax-ns#type	w3.org/2006/vcard/ns#VCard
_:r1	w3.org/2006/vcard/ns#fn	John Smith
_:r1	w3.org/2006/vcard/ns#adr	_:r2
_:r2	w3.org/1999/02/22-rdf-syntax-ns#type	w3.org/2006/vcard/ns#Address
_:r2	w3.org/2006/vcard/ns#locality	New York

Table 3.1: *Triples extracted from a hCard. 'http://www.' has been removed from the URIs for brevity.*

formations that could extract RDF from the hCard and hCalender data.

Another problem with ActiveRDF and its current ability to handle microformats is that blank nodes are not supported by the framework. Loading RDF data via GRDDL from a web document containing the hCard of listing 2.2 on page 10 using the Redland adapter in ActiveRDF produces the triples seen in table 3.1.

The first three triples have the blank node identifier _:r1 and not an URI as its subject because the information in the hCard is not related to any specific resource representing for example the person who the contact details are about. The second blank node identifier _:r2 is used to locally identify the composite structure of the address information. Not having a top-level URI identifiable resource, to which all other information is related, is a common pattern for microformats as they do not follow the triple based data model. Semantics encoded using RDFa or eRDF do not have this problem as they are both based on the RDF model.

Blank nodes as those described above are problematic in ActiveRDF because an URI is required to create the proxy object for a resource (see for example line 6 in listing 3.2 on page 23). Thus there is no way to access the data of a microformatted page as the kind described above because no subject URIs are available.

A third problem that inhibits the access to RDF data gleaned from microformatted data in ActiveRDF is the fact that classes to be used for creating proxy objects are only constructed based on the available resources that are described as being of the type `Class`. In the first row of table 3.1 the subject _:r1 is described as being of the type VCard, which happens to be of the type `Class`, which is defined in the vCard ontology definition found at w3.org/2006/vcard/ns. However, ActiveRDF would not recognize VCard as a class because there is no triple making a statement that it is. Such a statement would however be inferred if reasoning was performed on the data. Unfortunately the Redland RDF data store, which is the only data store providing GRDDL functionality to ActiveRDF, does not support reasoning.

3.7 Summary

On the basis of existing frameworks, different approaches to an integration of the Semantic Web standards with object-oriented programming languages have been shown. Both Jena and ActiveRDF have strong support for the RDF standard, though ActiveRDF benefits from the dynamics of the Ruby language to provide a deeper and more domain specific mapping.

With regards to OWL and reasoning, the Jena framework has comprehensive support whereas ActiveRDF is focused only on RDF data access. Relying on external components such as data stores to perform reasoning seems like a fair choice, however enforcement of OWL constraints and the possibility of making user defined rules and online reasoning could be of value.

Both frameworks support the SPARQL standard for performing queries on RDF data, however the dynamic finders of ActiveRDF makes it easier for the developer to perform simple queries.

Implementations of the relatively new GRDDL standard for bringing e.g. microformatted data into RDF are still in their early stages and face many problems making the majority of the semantics encoded in web documents using the popular microformats practically useless in an RDF context.

Chapter 4

A GRDDL Adapter for ActiveRDF

Given the perceived importance of Microformats and other encodings of semantics in XHTML as a means for helping to bootstrap the Semantic Web, support for the GRDDL standard is essential. As became evident in the previous chapter, the current implementations of the standard have problems that make especially microformats difficult to glean into RDF. This chapter documents the specification, design, implementation and test of a solution extending the ActiveRDF framework with a new GRDDL adapter that solves the problems of existing solutions.

4.1 ActiveRDF Architecture

To be able to extend ActiveRDF with new functionality an understanding of the architecture of the framework is needed. As described in [19], the framework has a layered architecture as depicted in figure 4.1.

The four toned layers in the middle gradually exposes RDF data from data sources to Ruby objects accessible from an application. When accessing data, the object manager handles the mapping of proxy object manipulations

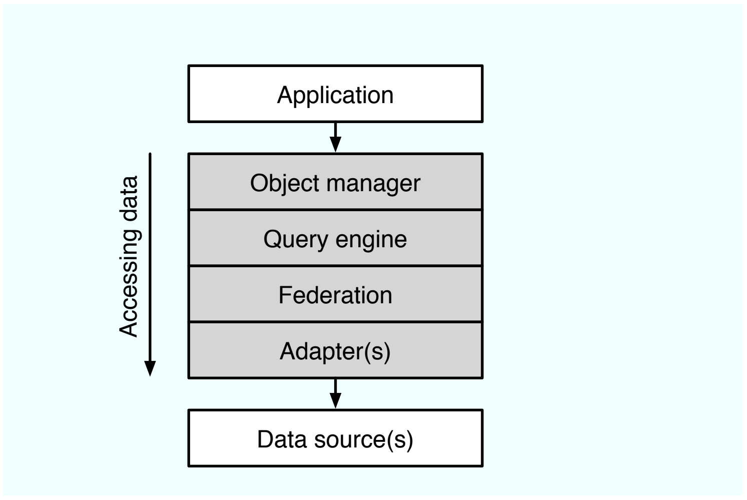


Figure 4.1: *ActiveRDF architecture.[19]*

to queries. The queries are expressed using the abstract query API provided by the query engine just below the object manager. Apart from the mapping functionality, the object manager also allows the construction of domain specific Ruby classes for the RDF Schema classes present in the data. The federation manager handles distribution of queries over the available data sources and combines the results by merging similar objects. Finally, the adapters give access to specific RDF data stores by mapping the abstract queries given by the federation manager to specific queries understood by the data store.

4.2 Specification

The ActiveRDF framework can be extended to work with new types of data sources by creating new adapters. All adapters implement a simple API including methods like `query()`, `add()` and `delete()` to be issued by the federation manager. Existing ActiveRDF adapters provide access to different RDF data stores, but adapters can also wrap different kinds of data sources as long as the result of the `query` method is RDF data.

4.2.1 RDF Data Through GRDDL

A GRDDL adapter will wrap semantically marked-up data in web documents and expose this as RDF. Following the pattern of other ActiveRDF adapters (see e.g. listing 3.2 on page 23), the GRDDL adapter should implement a `load()` method that given an URI will load any RDF data that can be extracted from the resource. The `load()` method will implement the procedure for GRDDL-aware agents[37] recommended by the GRDDL standard.

What makes this GRDDL implementation different from others is that additional steps will be added to the procedure to ensure a higher level of compatibility with microformatted web documents. As was shown in section 3.6, 13 out of 228 web documents with microformatted data did not contain links to transformations, making these 13 documents useless with current GRDDL implementations. The success criteria of this GRDDL implementation will thus be to improve on the low level of compatibility with Microformats exhibited by current GRDDL implementations.

Specification of the `load()` Method

Given an URI of a web document, the `load()` method of the GRDDL adapter should:

1. Find all transformations that can be applied to extract RDF from the document, i.e.:
    (a) Look for information linking directly to transformations.
    (b) Look for microformatted data by parsing the document and finding data marked-up according to known microformats such as hCard, hCalender etc. Based on the found microformats, select the appropriate transformations based on a list associating microformats with transformations.

2. Apply the transformations.
3. Merge and return the results in RDF.

The above is based on the mentioned procedure for GRDDL-aware agents but with step (b) as the addition to the standard that should ensure a higher level of compatibility with microformatted Web documents without profile information.

The GRDDL standard expands on step (a) in the above procedure and makes four special cases considering how transformation information can be included in different kinds of XML documents. The first case is for general XML documents, the second for XML namespace documents, which can implicitly assign a GRDDL transformation to all documents sharing a namespace. Third and fourth, there are cases for XHTML documents with explicitly assigned transformations and XHTML documents that are linked to transformations through shared profile documents.

To initially limit the scope of the GRDDL adapter, only the special cases regarding XHTML documents will be handled. These special cases of the GRDDL standard will cover the situations where semantic data is encoded in Web documents using e.g. eRDF or RDFa, which will not be detected as Microformats in step (b) in the procedure for the `load()` method, but through links to transformations. This limitation is in accordance with the purpose of the GRDDL adapter, which as mentioned is to allow encodings of semantic data in existing Web documents to be extracted into RDF, as this source of data is believed to be what will help bootstrap the Semantic Web.

Sample Application

Listing 4.1 shows an example of XHTML document that might be loaded using the GRDDL adapter.

Listing 4.1: *Example XHTML containing Microformats.*

```
<html>
<head profile="http://www.w3.org/2006/03/hcard">
  <title>John Smiths's Homepage</title>
</head>
<body>

<div class="vcard" id="me">
  My name is <span class="fn">John Smith</span>
  <div class="adr">
    and I live in
    <span class="locality">New York</span>.
  </div>
</div>

I will be attending the following events in the near future:
<div class="vevent">
  <span class="summary">Web 2.0 Conference</span>:
  <abbr class="dtstart" title="2007-10-05">October 5</abbr>-
  <abbr class="dtend" title="2007-10-20">19</abbr>,
  at <span class="location">San Francisco, CA</span>
</div>

</body>
</html>
```

The document contains the same hCard microformat as previously shown in listing 2.2 but adds a description of an event that this person will be attending using the hCalender microformat.

Given the URI of this document, the GRDDL adapter will start looking for transformations to extract RDF from the available data. Following step (a) from the `load()` procedure described above, the profile attribute in the head element will be found and the referenced profile document will be retrieved. The profile document identifies the XSLT transformation via a link of type `profileTransformation` as according to the standard. In step (b) of the procedure the document will be parsed and the hCalender microformat will be recognized and the approriate transformation selected. The two found transformations will then be applied to the document to transform the hCard and hCalender microformatted data into RDF.

4.2.2 Storing and Querying the RDF Data

The `load` method specified in the last section will save the resulting RDF data in a data store, so that it can be queried later as a result of proxy object invocations made in the application code. For storage of RDF the Redland data store has been chosen, because an ActiveRDF adapter for this data store is already available. During initialization of the GRDDL adapter, an instance of the Redland adapter should be created and subsequently used to store the RDF data extracted from web documents by the GRDDL adapter.

When the RDF is needed to answer queries from the application, the federation manager of the ActiveRDF framework will invoke the `query` methods of all registered adapter instances. As the GRDDL adapter will have forwarded all data to the Redland data store, its `query` method should return an empty result set upon invocation. The Redland adapter will then answer the queries as it will have access to all RDF data produced by the GRDDL adapter.

4.2.3 Blank Nodes and Reasoning

In section 3.6 two problems with the existing GRDDL implementation available in ActiveRDF were described. The first was the issue of blank nodes in an RDF graph, which cannot be accessed because of the lack of URIs. To solve this problem, the `load` method of the GRDDL adapter of this project will replace blank nodes with normal URI nodes. This is done by renaming blank node identifiers to URIs before saving any RDF data in the data store. The renaming will be done by concatenating the URI of the document from which the data originates with the given blank node identifier. In this way, all blank nodes will be replaced by unique URI based nodes, that can be accessed from ActiveRDF.

The second problem which was mentioned earlier, is that without reasoning on the RDF data extracted from a document, domain specific classes cannot not be used in ActiveRDF. Being able to work with specific classes is important because it should be possible for the developer to get immediate access to specific kinds of data objects. For example, given a web document containing many different kinds of semantic data including hCalender events, the program in listing 4.2 should be working if reasoning is applied to the data by the `load` method of the GRDDL adapter before it is saved to the RDF store.

Listing 4.2: *ActiveRDF code with the domain specific Vevent class.*

```ruby
adapter = ConnectionPool.add_data_source :type => :grddl
adapter.load "http://www.js.com/my_events.html"

Namespace.register :ical, 'http://www.w3.org/2002/12/cal/icaltzd#'
ObjectManager.construct_classes

ICAL::Vevent.find_all.each do |event|
  puts event.summary
end
```

Nodes in the RDF data of type `Vevent` will with reasoning be inferred to be of the class `Vevent`, which can then be used as in the listing above to find all events and list the summaries of these.

4.3 Design

This section will describe the design of the GRDDL adapter fulfilling the specification made in the previous section. Figure 4.2 shows a `GrddlAdapter` class with its methods and the surrounding components that the methods of the class interacts with.

In the top right part of the figure, it is shown how the components of the ActiveRDF architecture described in section 4.1 interact with the adapter. The application will create an instance of the `GrddlAdapter` class and call

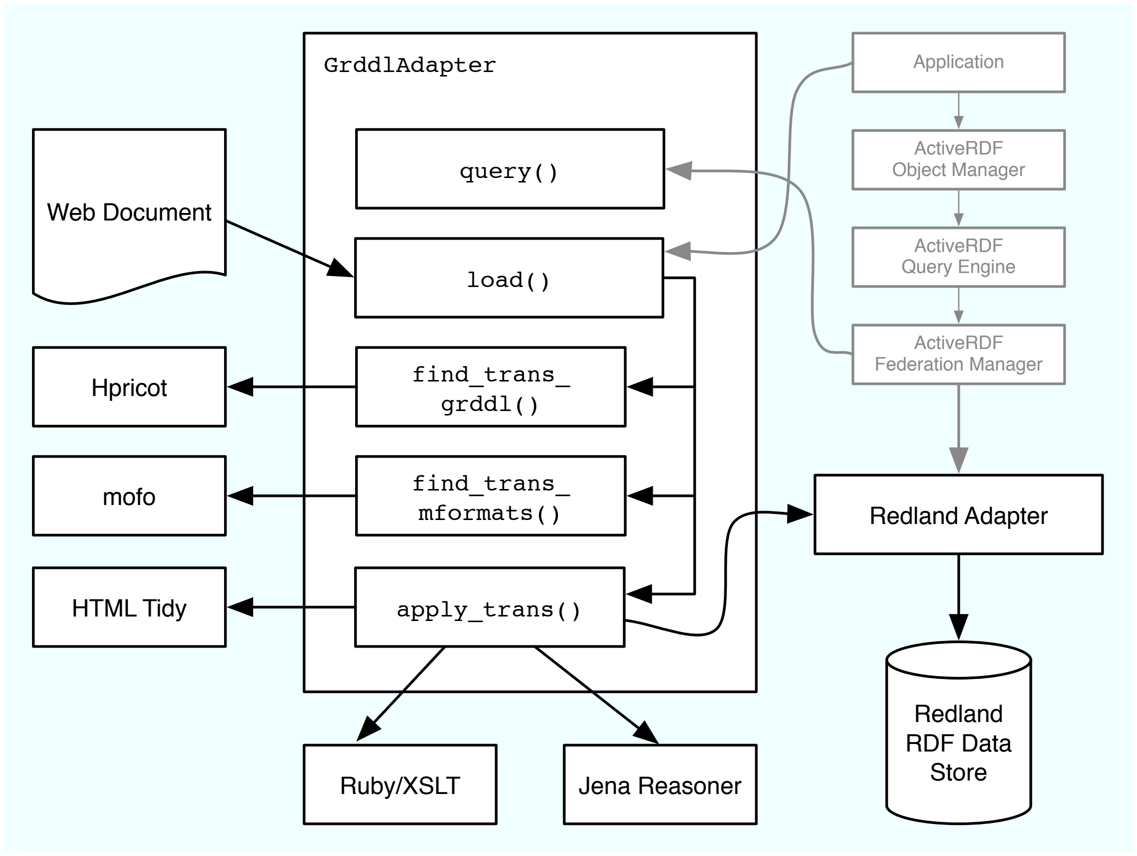


Figure 4.2: *Design components of the GRDDL adapter along with components of the ActiveRDF framework (shown in grey).*

the `load()` method supplying an URI. The `load` method fetches the web document and subsequently calls the three methods `find_trans_grddl()`, `find_trans_mformats()` and `apply_trans()` that collectively implement the procedure for the `load()` method specified earlier in section 4.2.1.

The three methods called by the `load()` method utilizes the following existing software components to carry out their individual tasks:

- **Hpricot**[4] is a HTML parser, that is used by the `find_trans_grddl()` method to find profile information and links to transformations in web documents following the specifications of the GRDDL standard.
- **mofo**[9] is a Microformats parser, that can discover and extract microformatted data in web documents. Only its capability to discover microformats will be utilized by the `find_trans_mformats()` method to determine the transformations needed to extract RDF from any microformatted data present in the document that is not revealed by profile links.
- **HTML Tidy** is a utility that can correct mark-up errors of HTML documents and convert them to valid XML. It is used by the `apply_trans()` method to ensure that a document is correct XML before an XSLT transformation is applied to it.
- **Ruby/XSLT**[12] is a library for processing XSLT and is used to apply XSLT transformations to the web document.
- **Jena Reasoner.** None of the data stores, including Redland, currently supported by ActiveRDF has reasoning capabilities. In contradiction, the Jena framework for Java is much more mature when it comes to reasoning support, and will thus be used to perform reasoning on the RDF data before the `apply_trans()` method saves it to the data store.

The diagram in figure 4.2 furthermore shows how the ActiveRDF federation manager uses the `query` method of both the GRDDL adapter and the Red-

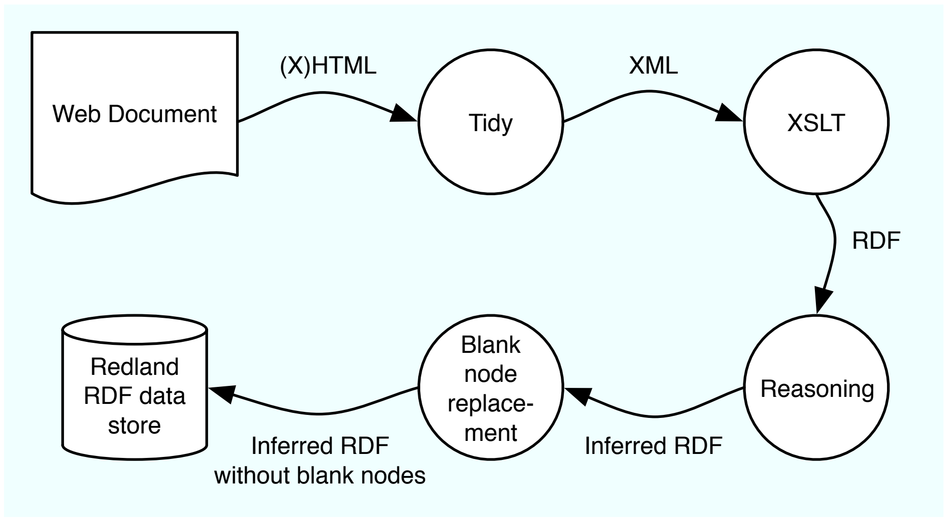


Figure 4.3: *Data flow diagram.*

land adapter. The GRDDL adapter will as mentioned in the specification always return an empty result set, whereas the Redland adapter will answer all queries using the RDF data that has been given by the `apply_trans()` method as the final step of the `load()` procedure of the GRDDL adapter.

Figure 4.3 is a data flow diagram of the system, showing how an (X)HTML document with one associated transformation ends up as RDF through the data transforming processes of the GRDDL adapter. In the case where more transformations need to be applied to the document, the processes after the Tidy process in the diagram are repeated for each transformation.

4.4 Implementation

This section describes selected implementation details of the GRDDL adapter. Focus will be on the parts of the implementation that represent the additional steps taken by this GRDDL implementation both to increase the compatibility with Microformats and to make the RDF data available as programming objects through the ActiveRDF framework.

First the method for finding Microformats independent of any profile information and secondly the implementation of the last two processes of the data flow diagram in figure 4.3, i.e. reasoning and blank node replacement, will be described. The full source code of the GRDDL adapter can be found in appendix A.1.

4.4.1 Finding Microformats

The method `find_trans_mformats()` of the `GrddlAdapter` class is shown in listing 4.3.

Listing 4.3: *Method for finding microformats.*

```
 1  def find_trans_mformats
 2    xslts = Hash["HCard" => "http://www.w3.../hcard2rdf.xsl",
 3                 "HCalendar" => "http://www.w3.../glean-hcal.xsl"]
 4
 5    formats = Set.new
 6    Microformat.find(@doc).each do |format|
 7      formats << format.class.to_s
 8    end
 9
10    formats.each do |f|
11      if xslts.include? f
12        @transformations << xslts[f]
13      end
14    end
15  end
```

Line 6 iterates over each of the microformats in the document found by the `find` method of the `Microformat` class from the mofo library. In the next line, the string value of the class of each found microformat is added to the `formats` object of class `Set`, which is used to avoid duplicates ocurring when the same microformat is used more than once in a document.

In line 2 a variable `xslts` is defined as a hash data structure and is used to associate mofo class names of microformats with URIs of XSLT transformations, that can transform the particular microformats into RDF. This is used in lines 10-14, where it is checked if each of the found microformats is included in the `xslts` hash and thus has a known associated transformation. If so, this transformation is added to the instance variable `@transformations`, which is the set of all transformations, including those found by the `find_grddl_trans()` method, that needs to be applied to the document.

4.4.2 Reasoning

To perform reasoning on the RDF data resulting from the applied XSLT transformations an external Java application utilizing the reasoning capabilities of the Jena framework is invoked from the Ruby code. Before this, the RDF data is written out to a file, which name will be passed as an argument to the Java program. Listing 4.4 shows the implementation of the class `JenaReasoner`.

Listing 4.4: *The JenaReasoner class.*

```java
public class JenaReasoner {

  public static void main (String args[]) {
    String inputFileName  = args[0];

    Model model = ModelFactory.createDefaultModel();

    InputStream in = FileManager.get().open(inputFileName);
    if (in == null) {
      throw new IllegalArgumentException("File not found.");
    }

    model.read(in, args[1]);
    InfModel inf = ModelFactory.createRDFSModel(model);

    try {
      OutputStream os = new FileOutputStream(args[0]);
      inf.write(os);

    } catch (FileNotFoundException ex) {
      System.out.println("File not found.");
    }
  }
}
```

In line 6 a default RDF model is created using the `ModelFactory` class of the Jena framework. Line 8 opens the file containing RDF data and in line 13 this data is read into the model, where the `args[1]` argument is the URI of the original web document and is provided as a base URI to be used by Jena to convert relative URIs to absolute. Afterwards, in line 14, a new inference model is created using the `createRDFSModel()` method of the `ModelFactory` class. This creates a new inference model based on the previous model, but now also containing all RDF triples that can be inferred using RDF Schema entailment rules. Finally, the inferred RDF is written out as RDF/XML in line 18.

Reasoning on the RDF data is performed before blank node replacement, because blank nodes can be expressed in RDF/XML in different ways. The Jena framework will always output RDF in a consistent way based on the internal RDF representation model, thus making the blank node replacement process simpler.

4.4.3 Blank Node Replacement

As the last step before the RDF data is added to the data store ready for application queries, the blank nodes must be replaced by nodes with URIs. This step is implemented using XSLT to transform the RDF/XML outputted by the Jena Reasoner to the same RDF/XML, but with blank nodes replaced. A sample of the RDF/XML that is produced by Jena can be seen in listing 4.5.

Listing 4.5: *The vCard of John Smith in RDF/XML as produced by the Jena Reasoner.*

```
1 <rdf:Description rdf:nodeID="A1">
2  <v:fn>John Smith</v:fn>
3  <v:adr rdf:nodeID="A0"/>
4  <rdf:type rdf:resource="http://www.w3.../vcard/ns#VCard"/>
5 </rdf:Description>
```

This RDF/XML is a representation of John Smith's vCard as known from previous examples. In line 1 and 3 it can be seen how Jena uses the **`rdf:nodeID`** attribute to refer to blank nodes of the vCard itself and a another blank node representing the composite value of John Smith's address. These attributes must be replaced by **`rdf:about`** or **`rdf:resource`** attributes depending on whether the element represents a RDF subject or property node as according to the RDF/XML syntax specification.[43] The value of the attributes will be the concatenation of the URI of the document from which the RDF data has been extracted and the blank node identifier given by Jena. Given that the vCard has been extracted from the URI `http://www.js.com/contact`, the XML in listing 4.5 should thus be transformed to that shown in listing 4.6.

Listing 4.6: *The vCard of John Smith with blank nodes replaced.*

```
<rdf:Description rdf:about="http://www.js.com/contact/A1">
 <v:fn>John Smith</v:fn>
 <v:adr rdf:resource="http://www.js.com/contact/A0"/>
 <rdf:type rdf:resource="http://www.w3.../vcard/ns#VCard"/>
</rdf:Description>
```

A XSLT transformation is expressed in a so called stylesheet, which defines templates of content to be generated as the result of the transformation. Each template matches a certain part of the source document and the template output replaces only this part. The matching criteria is expressed using XPath[54], which is a syntax for navigating through and selecting elements and attributes of a XML document. The developed XSLT stylesheet for replacing blank nodes will now be presented and explained. The first part of the stylesheet can be seen in listing 4.7.

Listing 4.7: *Part one of the XSLT stylesheet replacing blank nodes.*

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet
3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4    version="1.0">
5
6  <xsl:output method="xml" encoding="utf-8" indent="yes"/>
7  <xsl:param name="uri"/>
8
9  <xsl:template match="@*|node()">
10   <xsl:copy>
11     <xsl:apply-templates select="@*|node()"/>
12   </xsl:copy>
13 </xsl:template>
```

Lines 9-13 is the so called identity template, which matches all attributes (`@*`) and all nodes (`node()`) of the XML document tree and recursively copies them to the result of the transformation.[34] In line 11 the `xsl:apply-templates` element is used to call other templates that applies to the current node or attribute including itself – hence the recursion. If there are other templates in the stylesheet matching the current attribute or node, they will be applied and the copying behavior will be overruled. Using the identity template is thus a way to copy everything from the source document, but with exceptions defined by the other templates in the stylesheet. Such an exception is needed whenever an attribute with the name `rdf:nodeID` is encountered, which indicates the presence of a blank node that needs to be replaced. The template for this exception can be seen in listing 4.8 (note that indentation is not preserved to avoid line breaks).

Listing 4.8: *Part two of the XSLT stylesheet replacing blank nodes.*

```
1  <xsl:template match="@nodeID">
2  
3  <xsl:variable name="attr_name">
4  <xsl:choose>
5  
6  <xsl:when test="contains(name(parent::node()),’Description’)">
7    <xsl:value–of select="’:about’"/>
8  </xsl:when>
9  
10 <xsl:otherwise>
11   <xsl:value–of select="’:resource’"/>
12 </xsl:otherwise>
13 
14 </xsl:choose>
15 </xsl:variable>
16 
17 <xsl:variable name="prefix"
18               select="substring–before(name(),’:’)"/>
19 
20 <xsl:variable name="qname"
21               select="concat($prefix, $attr_name)"/>
22 
23 <xsl:attribute name="{$qname}" namespace="{namespace–uri()}">
24   <xsl:value–of select="$uri"/>
25   <xsl:value–of select="’/’"/>
26   <xsl:value–of select="."/>
27 </xsl:attribute>
28 
29 </xsl:template>
30 </xsl:stylesheet>
```

In line 1 all the relevant attributes are matched using the XPath expression `@nodeID`. From line 3 to 15 a variable named `attr_name` is defined and initialized. The value of this variable depends on whether the node is a subject or property node as previously described. Using the `xsl:choose` element, `attr_name` is given the value `:about`, when the name of the attribute's parent node, i.e. its element node, contains the string `'Description'`, which signifies that it is a subject node. Otherwise (line 10-12), the `attr_name` variable is given the value `':resource'`.

Lines 17-18 defines a `prefix` variable as the value of the string before the colon in the name of the attribute. It is necessary to extract the prefix in this way, because not all transformations used to extract RDF from web documents use `rdf` as the prefix value. Then in line 20-21 a new qualifying name to be used as the name of the replacing attribute is created by concatenating the values of the `prefix` and `attr_name` variables.

The result of applying the template is specified in lines 23-27, where a new attribute node is created using the `xsl:attribute` element. The value of the new attribute contains the value of the `uri` variable, which was declared in line 7 of listing 4.7, and is given as a XSLT parameter by the Ruby script executing the transformation. The `uri` value is concatenated with a slash and the value of the current attribute node (line 26), which is the blank node identifier.

4.5 Test

The following documents tests of the developed solution's Microformats compatibility and GRDDL standard compliance.

4.5.1 Microformats Compatibility

As mentioned in the specification (see section 4.2.1) of the developed GRDDL adapter, the success criteria was to improve on the low level of compatibility with Microformats exhibited by current GRDDL implementations. The test script of listing A.3 in appendix A.2 loads the same URIs with hCard or hCalender data as used during the tests of the GRDDL functionality of the Redland adapter described in section 3.6, but now using the developed GRDDL adapter instead. The result of the test is that it is now possible to access microformatted data as RDF in 152 out of 228 Web documents, i.e.

in 67% of the cases which is a radical improvement compared to the Redland adapter, which only had a possibility of working in 6% of the cases because of the lack of information associating the Web documents with transformations.

The test script for the GRDDL adapter revealed that in 32 cases the adapter was not able to extract RDF from a Web document because of XML parsing errors exhibited by the XSLT processing library. This means that the HTML Tidy library was not able to convert the (X)HTML into valid XML, which happens in cases where the mark up contains such irregularities that this is impossible. Unfortunately this is an inherent property of the Web, because browsers are extremely forgiving with regards to Web documents marked up using poor, non-standard compliant X(HTML). Another 12 Web documents were simply no longer available on the given URI. The remaining Web documents that failed during the test were found to do so because they either no longer contained any Microformats or was marked up incorrectly causing the mofo library not to detect them.

It must be noted though, that this large improvement in Microformats compatibility is limited to the specific microformats that are both recognized by the mofo Microformats parsing library and have associated XSLT transformations. To adapt to a new microformat, a new XSLT transformation has to be written and the GRDDL adapter code must be updated to accomodate to the new format. The last requirement could however be avoided if a dynamic web accessible resource associating microformats with transformations was used instead of the static hash variable of the current solution (see line 2 in listing 4.3). This is essentially a problem of scaling, and is inherent to all usage of Microformats. More general approaches such as RDFa, which was described in section 2.3.2, does not suffer from this condition as it uses a fixed syntax to encode data in XHTML and thus only one transformation for all RDFa data is needed.

4.5.2 GRDDL Test Cases

The W3C GRDDL Working Group provides a set of test cases for developers of so called GRDDL-aware agents that can be used to ensure that an implementation fulfills the standard.[38] Each test case has an input document and an output document, which represents the expected result of the GRDDL transformation.

A test driver written in Python is also provided, which automatically runs all tests given a so called test manifest containing a description of the test cases in RDF/XML and the path to the GRDDL implementation. The test driver executes the GRDDL implementation providing the input document and compares the output RDF graph with the expected using graph isomorphism testing.

The set of test cases are devided into categories each covering some part of the GRDDL standard. As was specified in section 4.2.1, the developed GRDDL adapter implements two of the four special cases of the GRDDL standard concerning how information about transformations can be embedded in XHTML documents. Hence, only the tests relating to the categories of "Nominating Transformations via GRDDL Metadata Profile" and "Identifying Metadata Profile Transformations" are contained in the test manifest.

While running the set of test cases, some tests failed and revealed details of the standard that the implementation did not consider. To give an example, the test case named "2 profiles: eRDF and hCard" exercises the rule that the `profile` attribute of the `head` element can contain references to multiple profiles seperated by spaces and that references do not necessarily have to be absolute, but can be relative to the URI of the containing document. This is shown in listing 4.9 showing the `head` element of the input document for this test case.

Listing 4.9: *Head element of test case intput document.*

```
<head profile="erdf hcard"> ... </head>
```

As can be seen this needs to be handled differently than e.g. the `profile` attribute in line 1 of listing 2.2 on page 10. In this and other cases, the implementation was adjusted to make the tests pass. However, some tests checking the robustness of the implementation in the face of various odd cases have been left to be passed by future versions of the implementation.

4.6 Summary

Based on the identified need for an implementation of the GRDDL standard with higher Microformats compatibility, an adapter for the ActiveRDF framework fulfilling this need has been developed. This chapter has documented the specification, design, implementation and test of the developed GRDDL adapter. The tests showed that the solution worked with 67% of a set of microformatted Web documents, which is a big improvement compared to existing solutions that only had a possiblity of working in 6% of the cases.

The developed GRDDL thus provides a solution to bring microformatted data into RDF, so that it can be used together with other semantic data based on e.g. eRDF or RDFa encoded in Web documents as well as other sources of RDF. The designed solution furthermore solves the problem of blank nodes and performs reasoning on the extracted RDF data to enable object-oriented access to the RDF through the ActiveRDF framework.

Chapter 5

Semantic Web Applications

This chapter first discusses how Semantic Web applications differ from traditional web applications and also considers the use of supporting development frameworks. Then two ideas for Semantic Web Applications are proposed. One of the ideas is implemented as a prototype using the ActiveRDF framework together with the GRDDL adapter developed in this project.

5.1 Application Development for the Web

Semantic web applications fundamentally differ from traditional web applications with regards to the properties of the data used. Furthermore, a semantic application might get its data from many different kinds sources, including for example SPARQL endpoints and Web documents containing semantic data that can be extracted through GRDDL. These challenges need to be handled by Web development frameworks to allow developers to start creating the next generation of applications for the Web.

5.1.1 Data Sources

Traditional Web applications are most often driven by a single database following a specific database schema. This means that the developer will always know exactly what kind of data he has available in the application logic to generate the Web documents requested by the user of the application.

Over the last couple of years a new type of Web applications called mashups have emerged in great numbers. A mashup application moves away from the traditional Web application with only one underlying data source and instead integrates data from multiple sources and presents the result of this mashup of data to the user. A classic example of a mashup is how the Google Maps service can be combined with real estate information to show the locations of houses for sale. The data providers each has an API that the developer of the mashup must program specifically against to obtain the data. As with the traditional web application, the developer of a mashup still has complete knowledge of the representation and structure of the data he receives from the sources, but he may have to deal with different data representation for each of the sources.

This tight coupling is loosened in Semantic Web applications or what could also be called semantic mashups, because all data is assumed to be in the Semantic Web data representation language, RDF. Furthermore, a semantic application will to some degree be able to assume that certain kinds of data adheres to some well accepted ontology, that defines the structure and meaning of the data. There are however no guarantees in this matter. For example, some data representing people's contact details might be described using the vCard vocabulary while other data is described using the FOAF[2] (Friend-of-a-Friend) vocabulary. Also, semantic data does not need to have values for all terms described in a chosen vocabulary, e.g. some will have lattitude and longitude values associated with the location in their vCard data and others will not. In addition, the flexibility of semantic web data allows users to customize vocabularies to their own needs, which means that a user could for example add a new field to his vCard to describe how he can be contacted through instant messaging.

5.1.2 Development Frameworks

Developers of web applications use frameworks to increase productivity through reuse of common functionality and overall system design. However, current web development frameworks do not handle the distributed and semi-structured nature of semantic data as described in the previous section.

Take for example the popular Ruby on Rails framework[11], which imposes the Model-View-Controller (MVC) design pattern on the application. The model holds the domain objects of the application together with their business logic and ensures persistency of the objects through an object-relational mapping to the underlying relational database. The view is responsible for the user interface of the application for which it uses templates that are populated with data objects from the model. The controller handles incoming requests generated from user actions by interacting with the model and directing the data to the view. Similar frameworks also exists for other programming languages such as Struts[13] for Java and Django[1] for Python.

Both the view and controller functionality of these frameworks can be readily applied for semantic web applications, but the model layer needs to be accommodated to the properties of semantic data. In the case of Ruby on Rails, ActiveRDF is able to substitute the default model and its object-relational mapping with an object-RDF mapping and also add support for different semantic data sources by means of adapters such as the GRDDL adapter developed in this project.

One of the most prominent features of the Ruby on Rails framework is its so called scaffolding capabilities, which allows the developer to have basic view and controller code automatically generated based on a model object. The scaffolding will generate a simple interface to manipulate model objects via the CRUD (create, read, update and delete) operations allowing the developer to get quickly started on the application development. The developers of ActiveRDF have created similar scaffolding functionality but for RDF Schema classes, so that a basic skeleton application to manipulate

RDF data can be automatically generated. The views generated are however dynamic and based on specific instance data due to the flexible nature of semantic data described in the previous section. Thus the view of a person's vCard data would contain e.g. custom added instant messaging information even though it is not defined in the vCard vocabulary. The scaffold generators and ActiveRDF are collectively described as Semantic Web Application Framework (SWAF)[14] extending the Ruby on Rails framework to allow rapid development of Semantic Web applications.

5.2 Application Ideas

In this section two ideas for semantic web applications are presented. The first idea, a personal semantic organizer, uses semantic data which is already present on the Web today and is thus realistic to implement. The second idea, a semantic vacation planner, is less realistic as it is based on non-existing ontologies, but is nevertheless still

5.2.1 Personal Semantic Organizer

As is clear from the list of hCard and hCalender examples in the wild[27][26] this kind of semantic data is already present in a lot of web documents around the Internet. This data could be utilized in a semantic web application that would collect and organize calender and contact information. What separates such an application from a traditional non-semantic application is that it would be able to pull semantic contact and calender data from existing sources and use this data in interesting ways.

To illustrate, consider a user browsing the Internet looking for something to do for a Friday night out. He checks the local theater's website and finds a show that he wants to go see. He presses an "Organize" button is his browser, which takes him to his personal semantic organizer. To his luck, the theater website has marked up all shows as events using the hCalender microformat. The organizer recognizes the microformat and asks the user if he wants to add the event to his calender. The user agrees to add the event and then moves on to something else. Considering the case, where the web document of the particular theater show does not contain any semantically marked up data, the organizer would announce this to the user and let him manually add the event to the calender.

Another scenario describing how data is imported to the application could be the following: Upon visiting the website of for example a person or a company, the user might decide to add the contact details to his organizer as he needs to contact that person or company later on. Again, if the website contains semantic encodings of the contact details using e.g. the hCard microformat, the organizer will automatically extract and save the information.

Once semantic data has been imported to the application it can be used in several ways. Say the user wants to arrange a meeting with one of the contacts in his organizer. If this contact has published his calender information in a standardized semantic format, the organizer would be able to find and suggest times where both are available for a meeting. An email suggesting the meeting time could then be generated and sent automatically to the contact from the application based on contact information previously retrieved.

The organizer could also provide an overview to the user of the upcoming events that a group of contacts labeled e.g. "friends" are planning to attend. Each event might have an associated location as part of its description, which could be used to present the data in the form of a map with the locations plotted in using services such as Google Maps. The user might also want to recommend an event that he is attending to the contacts in the "friends" group, which would also be a completely automated function.

5.2.2 Semantic Vacation Planner

Planning a longer vacation to an unfamiliar destination can be quite a cumbersome exercise. Many things have to be considered, including where to stay, what to see and how to get around. A wealth of information is already available on the current Web to help in this planning effort, but on the future Semantic Web the user could be assisted to a much higher degree using a semantic vacation planner.

Imagine that a vocabulary (ontology) existed for hotels to semantically mark up the information about their services that most hotels already have on their websites today. This information would e.g. include how many stars the hotel has, rates and facilities (types of rooms, pool, wireless Internet, etc.). While browsing the Web for hotels, the user would add individual hotels or maybe a whole list of hotels located in a particular city to the vacation planner. By virtue of its knowledge about the meaning of the hotel data, the planner application would be able to assist the user in selecting a hotel by e.g. sorting the hotels after price, removing all those without wireless Internet etc. To further assist in the selection of a hotel, the application might present reviews of the hotels made by previous visitors and extracted from an external hotel review website. In fact, the hReview[5] microformat, which is one of the more widely adopted microformats, could readily be used for the purpose of hotel reviews.

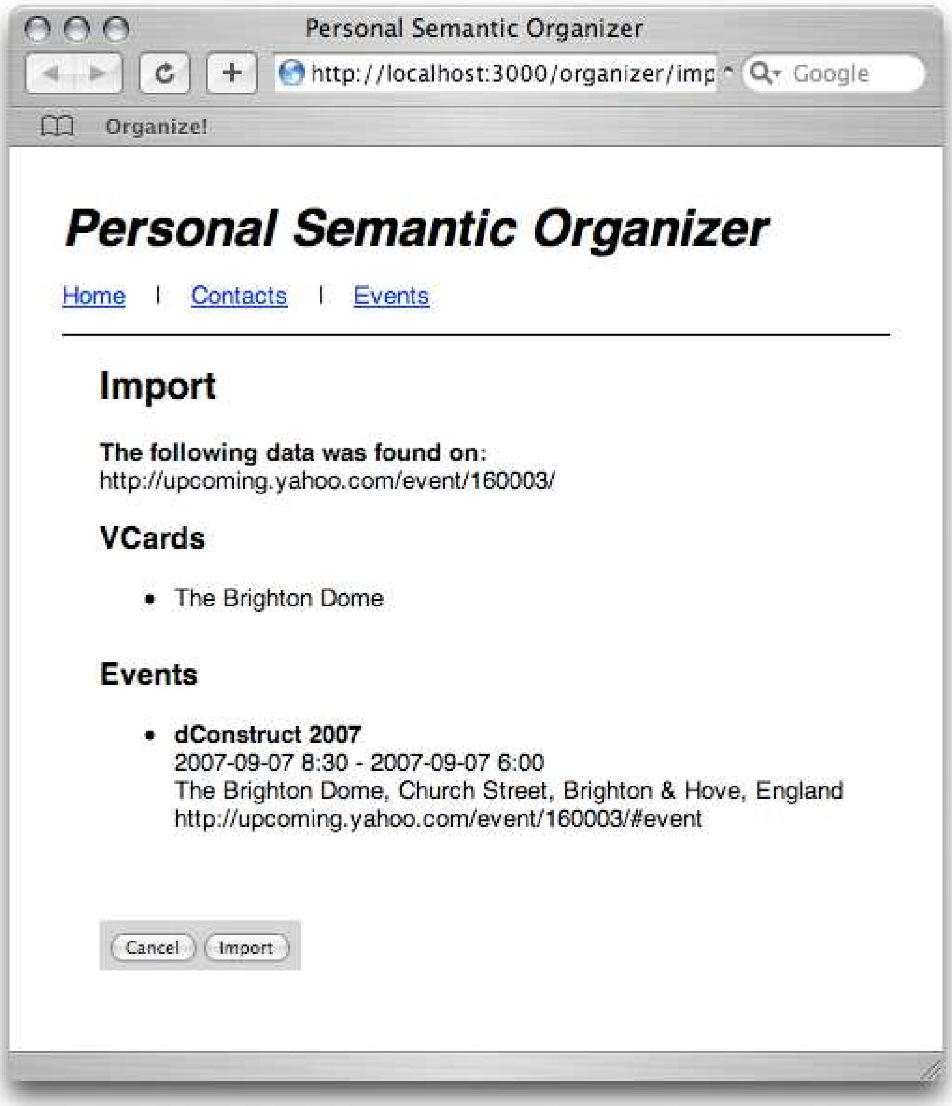
To find out which sights to see in a city, the user would browse different tourist information web sites with descriptions of the various sights. If any of the described sights catches the user's interest, he would add them to his vacation planner, which would further assists in the planning. The application would do this by for example sorting out all places with an entrance fee above some user set limit or any attractions, which are closed during the time of the user's stay in the city. Each sight might also have associated geographical information, which would make it possible to plot the sights on a map and maybe even have the planner suggest the most optimal route between the sights. Furthermore, the user may like to see photos taken by other people of the sights before deciding whether to see them himself or not. The user's of online photo sharing communities like Flickr.com have already begun doing so called geo-tagging, which assoicates location information with the photos. Using such a resource, the semantic vacation planner would be able to present the user with photos of each of the sights under consideration.

5.3 Implementing the Personal Semantic Organizer

This section describes the prototype implementation providing only the import functionality of the Personal Semantic Organizer application, which was described above.

The mentioned "Organize" button that the user will press to import any semantic data found on the current Web document to his Organizer can be implemented in a simple way using the bookmarks toolbar available in most browsers. This involves creating a bookmark for the toolbar, where the URI of the bookmark is set to contain a piece of JavaScript code that redirects the user to the Organizer application providing the current URI as a parameter. To provide a more user friendly solution, a plugin for the browser could be developed to add the "Organize" button, which might also graphically indicate if there is any semantic data to be imported from the Web document. However in this prototype the solution using the bookmarks toolbar is employed.

After including ActiveRDF in the Ruby on Rails environment file, it can immediately be used in controllers to work with semantic data as a replacement for the normal interaction with the model layer. Listing A.4 in appendix A.3 shows the `OrganizerController` class and its `import` method, that given an URI finds any events or vCards in the Web document using the GRDDL adapter developed in this project. The screenshot in figure 5.1 shows the resulting page of an import showed to the user after clicking on the "Organize" button. The user can then choose to import the data found by clicking "Import" or return to the Web site from which he came by clicking "Cancel".


Personal Semantic Organizer
http://localhost:3000/organizer/imp
Google
Organize!

Personal Semantic Organizer

Home | Contacts | Events

Import

The following data was found on:
http://upcoming.yahoo.com/event/160003/

VCards

- The Brighton Dome

Events

- **dConstruct 2007**
  2007-09-07 8:30 - 2007-09-07 6:00
  The Brighton Dome, Church Street, Brighton & Hove, England
  http://upcoming.yahoo.com/event/160003/#event

Figure 5.1: *The screen presented to the user of the Personal Semantic Organizer after clicking on the "Organize" button when browsing an event page under Yahoo's event listings service called Upcoming.*

5.4 Summary

In this chapter the topic of Semantic Web applications has been discussed in terms of how they are different from traditional Web applications. The main difference is the characteristics of the data sources used. Traditional Web applications use centralised databases, whereas Semantic Web applications use distributed and semi-structured data. This poses a requirement to Web application development frameworks, and it was described how the Ruby on Rails framework can be used in combination with ActiveRDF to facilitate rapid creation of Semantic Web applications.

Two proposals for Semantic Web applications were made of which one relies only on already existing semantic data in the form of contact and calender data. This Personal Semantic Organizer has been implemented as a prototype demonstrating how the GRDDL adapter for the ActiveRDF framework developed in this project can be applied to create a Semantic Web application.

Chapter 6

Conclusion

Programming the Semantic Web has been the subject of study in this master's thesis. The subject was approached by giving a state of the art technical analysis of the Semantic Web standards developed under the W3C organization. In this context, the independent Microformats initiative defining data formats for structuring e.g. contact and calender information in Web documents using existing standards was considered. It can be concluded that Microformats will have a very significant role in the bootstrapping process of the Semantic Web, as it is a huge source of semantic data. Given the recent GRDDL standard, microformatted data can be transformed into RDF thus ensuring interoperability.

Based on the Jena and ActiveRDF frameworks it was discussed how the Semantic Web standards can be integrated with object-oriented programming languages. It was shown that ActiveRDF provides a more deep and natural integration due to the dynamic properties of the Ruby language which accomodates well to the characteristics of RDF data. It can be concluded that programming language support for the RDF, RDF Schema and SPARQL standards is very good, whereas the higher layers of the Semantic Web layer cake are less supported, which is reasonable given the state of standardization concerning these layers. The relatively new GRDDL standard is supported both by ActiveRDF and Jena, but as was shown the standard specification as it stands makes no value of the many microformatted Web documents without links to transformations.

The main contribution of this thesis has been the development of a Microformats compatible GRDDL adapter for the ActiveRDF framework. Tests showed that the adapter was able to extract RDF from microformatted Web documents in 67% of the cases, which is a big improvement considering to the fact that only 6% of these test cases would have worked with other GRDDL implementations. Apart from improving on the Microformats compatibility, the adapter was designed to solve the problem of blank nodes and to perform reasoning on the RDF data making it available through domain specific classes in ActiveRDF.

Application development for the Semantic Web has been discussed and compared to traditional Web application development. The most notable difference is the change in the kind of data that applications use. With Semantic Web applications comes a shift towards distributed and semi structured data. This shift is already visible by means of the many available mashups, which however do not use semantic data but are tailored to handle each of its data source's specific data structures and formats. When looked at from the application's point of view, the Semantic Web standards provide a way of sharing data on the Web with a very high level of interoperability.

With the Semantic Web a lot of interesting and novel applications will emerge. Two ideas for such applications were proposed in this report and the Personal Semantic Organizer, which uses actual semantic data already available on the Web, has been implemented as a prototype. This was done using a combination of the Ruby on Rails and the ActiveRDF frameworks, which shows that large parts of the functionality of model-view-controller frameworks can be reused in the development of Semantic Web applications.

To really get the Semantic Web going more applications are needed to demonstrate the benefits of the technologies and to foster the publishing of more semantic data. The idea of the Semantic Web has still not reached mainstream and a commercial breakthrough is yet to be seen. However, there are several Semantic Web startups such as Radar Networks[10] currently working in stealth-mode and promising to realease revolutionary products very soon. Recently, the W3C has started a Semantic Web Education and Outreach (SWEO) Interest Group[50], which has been established to develop strategies and materials to increase awareness among the Web community of the need and benefit for the Semantic Web, and educate the Web community regarding related solutions and technologies. It will definitely be exciting to follow the developments within this area in the coming years.

Bibliography

[1] Django. http://www.djangoproject.com/.

[2] The Friend of a Friend (FOAF) project. http://www.foaf-project.org/.

[3] Semantic Web Research at HP Labs. http://www.hpl.hp.com/semweb/.

[4] Hpricot, a fast and delightful HTML parser. http://code.whytheluckystiff.net/hpricot/.

[5] hReview Microformat. http://microformats.org/wiki/hreview.

[6] Jena Semantic Web Framework, . http://jena.sourceforge.net/.

[7] Jena GRDDL Reader, . http://jena.sourceforge.net/grddl/.

[8] Microformats. http://www.microformats.org.

[9] mofo - a ruby microformat parser. http://mofo.rubyforge.org/.

[10] Radar Networks. http://www.radarnetworks.com/.

[11] Ruby on rails. http://rubyonrails.org/.

[12] ruby-xslt. http://raa.ruby-lang.org/project/ruby-xslt/.

[13] Struts. http://struts.apache.org/.

[14] SWAF - semantic web application framework. http://www.semanticdesktop.org/xwiki/bin/view/Wiki/SWAF.

[15] XSL Transformations (XSLT). http://www.w3.org/TR/xslt.

[16] Danny Ayers. The Shortest Path to the Future Web. *IEEE Internet Computing*, 10(6):76–79, Nov/Dec 2006.

[17] Tantek Çelik. XMDP - XHTML Meta Data Profiles. http://gmpg.org/xmdp/.

[18] Daniel Westphal Chris Bizer. Developers Guide to Semantic Web Toolkits for different Programming Languages. http://sites.wiwiss.fu-berlin.de/suhl/bizer/toolkits/.

[19] Sebastian Gerke Armin Haller Stefan Decker Eyal Oren, Renaud Delbru. ActiveRDF: Object-Oriented Semantic Web Programming. In *WWW*, Banff, Alberta, Canada, May 8-12 2007.

[20] T. Howes F. Dawson. RFC 2426: vCard MIME Directory Profile. http://www.ietf.org/rfc/rfc2426.txt.

[21] Benjamin Heitmann. Fun with Java and ActiveRDF. http://blog.activerdf.org/articles/2007/02/05/fun-with-java-and-active-rdf.

[22] Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian A. Padget. Automatic Mapping of OWL Ontologies into Java. In *SEKE*, pages 98–103, 2004.

[23] Graham Klyne. Swish: Semantic Web Inference Scripting in Haskell. http://www.ninebynine.org/RDFNotes/Swish/Intro.html.

[24] Talis Information Ltd. Embedded RDF. http://research.talis.com/2005/erdf/wiki.

[25] Brian McBride. An Introduction to RDF and the Jena RDF API. http://jena.sourceforge.net/tutorial/RDF_API/index.html.

[26] Microformats. hCalender Examples in the wild, . http://microformats.org/wiki/hcalendar-examples-in-wild.

[27] Microformats. hCard Examples in the wild, . http://microformats.org/wiki/hcard-examples-in-wild.

[28] Microformats. Profile URIs, . http://microformats.org/wiki/profile-uris.

[29] TIOBE Software. IOBE Programming Community Index. http://www.tiobe.com/tpci.htm.

[30] J. Hendler T. Berners-Lee and O. Lassila. The Semantic Web. *Scientific American*, 279(5):34–43, May 2001.

[31] Brian Suda Tantek Çelik. hCard, . http://microformats.org/wiki/hcard.

[32] Kevin Marks Tantek Çelik. Real World Semantics, . http://www.tantek.com/presentations/2004etech/realworldsemanticspres.html.

[33] Marvin V. Zelkowitz Terrence W. Pratt. *Programming Languages: Design and Implementation*. Prentice-Hall, 4th edition, 2001.

[34] Jesper Tverskov. Identity Template: xsl:copy with recursion. http://www.xmlplease.com/xsltidentity.

[35] Max Völkel. RDFReactor – From Ontologies to Programmatic Data Access. HP Bristol, Jena User Conference 2006, May 2006.

[36] W3C. Gleaning Resource Descriptions from Dialects of Languages (GRDDL), . http://www.w3.org/TR/grddl/.

[37] W3C. GRDDL-Aware Agents, . http://www.w3.org/TR/grddl/#sec_agt.

[38] W3C. GRDDL Test Cases, . http://www.w3.org/TR/grddl-tests/.

[39] W3C. Semantic Web Layer Cake. http://www.w3.org/2007/03/layerCake.png.

[40] W3C. XHTML Metainformation Attributes Module, . http://www.w3.org/TR/xhtml2/mod-metaAttributes.html.

[41] W3C. OWL Web Ontology Language Overview, . http://www.w3.org/TR/owl-features/.

[42] W3C. RDF Primer, . http://www.w3.org/TR/rdf-primer/.

[43] W3C. RDF/XML Syntax Specification (Revised), . http://www.w3.org/TR/rdf-syntax-grammar/.

[44] W3C. RDFa Primer 1.0, . http://www.w3.org/TR/xhtml-rdfa-primer/.

[45] W3C. Rule Interchange Format, . http://www.w3.org/2005/rules/.

[46] W3C. Semantic Web Activity, . http://www.w3.org/2001/sw/.

[47] W3C. RDF Data Access WG Charter, . http://www.w3.org/2003/12/swa/dawg-charter.

[48] W3C. SPARQL Protocol for RDF, . http://www.w3.org/TR/rdf-sparql-protocol/.

[49] W3C. SPARQL Query Language for RDF, . http://www.w3.org/TR/rdf-sparql-query/.

[50] W3C. Semantic Web Education and Outreach (SWEO) Interest Group. http://www.w3.org/2001/sw/sweo/.

[51] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, . http://www.w3.org/Submission/SWRL/.

[52] W3C. Representing vCard Objects in RDF/XML, . http://www.w3.org/TR/vcard-rdf.

[53] W3C. W3C Technical Report Development Process. http://www.w3.org/2005/10/Process-20051014/tr.html#maturity-levels.

[54] W3C. XML Path Language (XPath). http://www.w3.org/TR/xpath.

[55] Jan Wielemaker. SWI-Prolog Semantic Web Library. http://www.swi-prolog.org/packages/semweb.html.

[56] ESW Wiki. Custom RDF Dialects. http://esw.w3.org/topic/CustomRdfDialects.

[57] Wikipedia. Prolog, . http://en.wikipedia.org/wiki/Prolog.

[58] Wikipedia. Purely functional, . http://en.wikipedia.org/wiki/Purely_functional.

[59] Wikipedia. Object-relational impedance mismatch, . http://en.wikipedia.org/wiki/Object-Relational_impedance_mismatch.

Appendix A

Source Code

A.1 GRDDL Adapter

Listing A.1: *GRDDL Adapter source code.*

```ruby
require 'rubygems'
require 'active_rdf'
require 'queryengine/query2sparql'
require 'set'
require 'digest/md5'
require 'open-uri'
require 'hpricot'
require 'mofo'
require 'tidy'
require 'sablot'
require 'xml/xslt'

class GrddlAdapter < ActiveRdfAdapter
  ConnectionPool.register_adapter(:planck, self)
  Tidy.path = '/usr/lib/libtidy.dylib'

  def initialize(params = {})
    @reads = true
    @writes = false

    @rdf_store = ConnectionPool.add_data_source :type => :redland
  end

  def load(uri)
    @transformations = Set.new
    @uri = uri
    @doc = Hpricot(open(@uri))

    find_trans_grddl
    find_trans_mformats
    apply_trans
  end

  def find_trans_grddl
    if profile_uri = @doc.at("head")['profile']
      if profile_uri == "http://www.w3.org/2003/g/data-view"
        @doc.search("//a|link[@rel]") do |t|
          if t['rel'].split(%r{\s+}).include?("transformation")
            @transformations << URI.join(@uri, t['href'])
          end
        end
      else
        profile_uri.split(%r{\s+}).each do |p|
          if p.include?("http://")
            profile = Hpricot(open(p))
          else
            path = @uri.slice(%r{(.*/).*$}, 1)
            profile = Hpricot(open(URI.join(path, p)))
          end

          profile.search("//a[@rel='profileTransformation']") do |t|
            @transformations << URI.join(path, t['href']).to_s
          end
        end
      end
    end
  end

  def find_trans_mformats
```

```
    xslts = Hash["HCard" => "http://www.w3.org/2006/vcard/
        hcard2rdf.xsl",
                 "HCalendar" => "http://www.w3.org/2002/12/cal/
                     glean−hcal.xsl"]

    formats = Set.new
    Microformat.find(@doc).each do |format|
      formats << format.class.to_s
    end

    formats.each do |f|
      if xslts.include? f
        @transformations << xslts[f]
      end
    end
  end

  def apply_trans
    html = @doc.to_s

    xml = Tidy.open do |tidy|
      tidy.options.output_xml = true
      tidy.options.wrap = 0
      tidy.options.input_xml = true
      xml = tidy.clean(html)
    end

    @transformations.each do |t|
      xsl = open(t).read

      filename = "rdf_data/" + Digest::MD5.hexdigest(@uri + Time.
          new.to_s) + ".rdf"
      file = File.new(filename, "w")
      file << Sablot::process_strings(xsl, xml)
      file.close

      ‘java JenaReasoner "#{filename}" "#{@uri}"‘

      rename_bnodes(filename)

      @rdf_store.load(filename, "rdfxml")
    end
  end

  def rename_bnodes(filename)
    xslt = XML::XSLT.new
    xslt.xml = filename
    xslt.xsl = "replace_bnodes.xsl"

    xslt.parameters = { "uri" => String.new(@uri) }

    file = File.new(filename, "w")
    file << xslt.serve
    file.close
  end

  def query(query)
    return Array.new
  end

  def delete(s,p,o)
    @rdf_store.delete(nil, nil, nil)
  end

  def dump
    @rdf_store.dump
  end
end
```

A.2 Test Scripts

Listing A.2: *Test script documenting the the lack of links to transformations in microformatted web documents.*

```
#!/usr/local/bin/ruby
require 'rubygems'
require 'mofo'
require 'hpricot'
require 'open-uri'
```

```

addrs = IO.readlines("microformatted_docs.txt")

total_count = 0;
profile_count = 0;
profiles = Hash.new

addrs.each do |addr|
  puts addr

  begin
    doc = Hpricot(open(addr))

    if profile = doc.at("head")['profile']
      profile_count += 1

      if profiles.has_key?(profile)
        profiles[profile] = profiles[profile] + 1
      else
        profiles[profile] = 1
      end

      puts "Profile found: " + profile
    end
    total_count += 1
    puts "\n"

  rescue Exception
    puts "Error: " + $!
  end
end

puts "\n\nTest results: " + profile_count.to_s + " profiles found
    in " +
     total_count.to_s + " pages containing microformats.\n\n"

profiles.each do |profile, count|
  puts profile + " – " + count.to_s + "\n"
end
```

Listing A.3: *Test script for the GRDDL Adapter.*

```
#!/usr/local/bin/ruby
require 'rubygems'
require 'active_rdf'

Namespace.register(:ns1, 'http://www.w3.org/2006/vcard/ns#')
Namespace.register(:ns2, 'http://www.w3.org/2002/12/cal/icaltzd#')

@errors = Hash.new
@working = 0;
@not_working = 0;

def test_microformats(filename, type)
  addrs = IO.readlines(filename)

  addrs.each do |addr|
    adapter = ConnectionPool.add_data_source :type => :planck

    begin
      adapter.load addr
    rescue Exception
      if @errors.has_key?($!.to_s)
        @errors[$!.to_s] = @errors[$!.to_s] + 1
      else
        @errors[$!.to_s] = 1
      end
    end

    ObjectManager.construct_classes

    if type == "hcard"
      objects = NS1::Vcard.find_all
    elsif type == "hcal"
      objects = NS2::Vevent.find_all
    end

    if objects.nil? or objects.empty?
      @not_working += 1
    else
      @working += 1
    end

    ConnectionPool.clear()
  end
end

test_microformats("hcard.txt", "hcard")
test_microformats("hcal.txt", "hcal")

total = @working + @not_working
puts "Test results: Data could be extracted from " + @working.to_s
     +
      " out of " + total.to_s + " documents."

@errors.each do |error, count|
  puts error + " — " + count.to_s + "\n"
end
```

A.3 Personal Semantic Organizer

Listing A.4: *Organizer controller.*

```
class OrganizerController < ApplicationController

  def initialize
    @adapter = ConnectionPool.add_data_source :type => :grddl
  end

  def index
  end

  def import
    @uri = params[:uri]
    @adapter.load @uri

    Namespace.register(:ns1, 'http://www.w3.org/2006/vcard/ns#')
    Namespace.register(:ns2, 'http://www.w3.org/2002/12/cal/
        icaltzd#')

    ObjectManager.construct_classes

    @vcards = NS1::Vcard.find_all
    @events = NS2::Vevent.find_all
  end

  def events
  end

  def contacts
  end
end
```